

LNAI 6472

Noriaki Ando Stephen Balakirsky
Thomas Hemker Monica Reggiani
Oskar von Stryk (Eds.)

Simulation, Modeling, and Programming for Autonomous Robots

Second International Conference, SIMPAR 2010
Darmstadt, Germany, November 2010
Proceedings



Springer

Lecture Notes in Artificial Intelligence 6472

Edited by R. Goebel, J. Siekmann, and W. Wahlster

Subseries of Lecture Notes in Computer Science

Noriaki Ando Stephen Balakirsky
Thomas Hemker Monica Reggiani
Oskar von Stryk (Eds.)

Simulation, Modeling, and Programming for Autonomous Robots

Second International Conference, SIMPAR 2010
Darmstadt, Germany, November 15-18, 2010
Proceedings

Series Editors

Randy Goebel, University of Alberta, Edmonton, Canada

Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Wolfgang Wahlster, DFKI and University of Saarland, Saarbrücken, Germany

Volume Editors

Noriaki Ando

AIST Tsukuba Central 2, RT-Synthesis R.G., ISRI, AIST

Tsukuba, Ibaraki 305-8568, Japan

E-mail: n-ando@aist.go.jp

Stephen Balakirsky

National Institute of Standards and Technology, Intelligent Systems Division

100 Bureau Drive, Gaithersburg, MD 20899-8230, USA

E-mail: stephen@nist.gov

Thomas Hemker

Oskar von Stryk

Technische Universität Darmstadt, Department of Computer Science

Hochschulstr. 10, 64289 Darmstadt, Germany

E-mail: {hemker, stryk}@sim.tu-darmstadt.de

Monica Reggiani

University of Padua, Department of Management and Engineering (DTG)

Stradella San Nicola, 3, 36100 Vicenza, Italy

E-mail: monica.reggiani@unipd.it

Library of Congress Control Number: 2010939068

CR Subject Classification (1998): I.2.9-11, I.2, I.6, H.3, F.1, D.2, C.2, H.4-5

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743

ISBN-10 3-642-17318-7 Springer Berlin Heidelberg New York

ISBN-13 978-3-642-17318-9 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

springer.com

© Springer-Verlag Berlin Heidelberg 2010

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

Why are the many highly capable autonomous robots that have been promised for novel applications driven by society, industry, and research not available today despite the tremendous progress in robotics science and systems achieved during the last decades? Unfortunately, steady improvements in specific robot abilities and robot hardware have not been matched by corresponding robot performance in real world environments. This is mainly due to the lack of advancements in robot software that master the development of robotic systems of ever increasing complexity. In addition, fundamental open problems are still awaiting sound answers while the development of new robotics applications suffers from the lack of widely used tools, libraries, and algorithms that are designed in a modular and performant manner with standardized interfaces. Simulation environments are playing a major role not only in reducing development time and cost, e.g., by systematic software- or hardware-in-the-loop testing of robot performance, but also in exploring new types of robots and applications. However, their use may still be regarded with skepticism. Seamless migration of code using robot simulators to real-world systems is still a rare circumstance, due to the complexity of robot, world, sensor, and actuator modeling.

These challenges drive the quest for the next generation of methodologies and tools for robot development. The objective of the International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR) is to offer a unique forum for these topics and to bring together researchers from academia and industry to identify and solve the key issues necessary to ease the development of increasingly complex robot software.

The second SIMPAR 2010 was held November 15–18, in Darmstadt at the darmstadtium Conference Center. It continues the work of the first SIMPAR 2008 conference in Venice which was initiated to offer a selected number of researchers the possibility to discuss, in a highly stimulating atmosphere, how to identify and solve the key issues necessary to ease the development of software for autonomous robots.

This book collects 45 contributed papers of which 28 were presented as talks and 17 as posters in Darmstadt, selected among a total of 74 that were submitted to the main single-track conference. Thirteen papers address methodologies and environments of robot simulation, 14 refer to methodologies about autonomous robot programming and middleware, and 18 describe applications and case studies. Each submitted paper received at least two reviews by the members of a carefully selected international Program Committee.

In addition, to enlarge the scientific attention towards particularly challenging topics, seven workshops were offered: Biomechanical Simulation of Humans and Bio-Inspired Humanoids (BH)²; Brain – Computer Interface; Domestic Service Robots in the Real World; International Workshop on Dynamic languages for

RObotic and Sensors systems (DYROS); Simulation Technologies in the Robot Development Process; Standards and Common Platforms for Robotics; Teaching Robotics, Teaching with Robotics. Papers presented at these workshops were collected in a CD-ROM edited separately by Emanuele Menegatti. Additionally, two tutorials on Model-Driven Software Development in Robotics and An Introduction to the OpenSim API were offered. Four invited talks were presented by Peter Fritzson, Brian Gerkey, Takayuki Kanda, and Oussama Khatib.

We want to gratefully thank the Graduate School of Computational Engineering and the Research Training Group on Cooperative, Adaptive and Responsive Monitoring of Technische Universität Darmstadt for their support. We also express our gratitude to the Program Committee members and all other supporters, organizers, and volunteers who contributed in making SIMPAR possible. Without their effort, it would not have been possible to run SIMPAR!

November 2010

Noriaki Ando
Stephen Balakirsky
Thomas Hemker
Monica Reggiani
Oskar von Stryk

Organization

Executive Committee

General Chair	Oskar von Stryk, TU Darmstadt, Germany
International Program Co-chairs	
America	Stephen Balakirsky, NIST, USA
Asia	Noriaki Ando, AIST, Japan
Europe	Monica Reggiani, University of Padua, Italy
Local Chair	Thomas Hemker, TU Darmstadt, Germany
Award Chair	Itsuki Noda, AIST, Japan
Exhibition Chair	Martin Friedmann, TU Darmstadt, Germany
Tutorial Chair	Davide Brugali, University of Bergamo, Italy
Workshop Chair	Emanuele Menegatti, University of Padua, Italy

Steering Committee

America	Maria Gini, University of Minnesota, USA Lynne Parker, University of Tennessee, USA
Asia	Tamio Arai, University of Tokyo, Japan Xiaoping Chen, University of Science and Technology of China
Europe	Herman Bruyninckx, Katholieke Universiteit Leuven, Belgium Enrico Pagello, University of Padua, Italy

Program Committee

America	Jeff Durst, US Army Corps of Engineers, USA Matei Ciocarlie, Willow Garage, Inc., USA Michael Lewis, University of Pittsburgh, USA Michael Quinlan, University of Texas at Austin, USA Bill Smart, Washington University in St. Louis, USA Mohan Sridharan, Texas Tech University, USA
---------	---

VIII Organization

Asia	Sang Chul Ahn, KIST, Korea Joschka Boedecker, Osaka University, Japan Takayuki Kanda, ATR Intelligent Robotics & Communication Labs, Japan Hyun Kim, ETRI, Korea Yongmei Liu, Sun Yat-sen University, China Bruce MacDonald, University of Auckland, New Zealand Takashi Minato, Osaka University, Japan Oliver Obst, CSIRO, Sydney, Australia Jun Ota, The University of Tokyo, Japan Masaharu Shimizu, Chiba Institute of Technology, Japan Masayuki Shimizu, Shizuoka University, Japan Yuki Suga, Waseda University, Japan Masaki Takahashi, Keio University, Japan Sasaki Takeshi, Shizuoka University, Japan Mary-Anne Williams, The University of Technology, Sydney, Australia
Europe	Jan Bender, TU Darmstadt, Germany Antonio Chella, University of Palermo, Italy Antonio Dominguez-Brito, University of Las Palmas de Gran Canaria, Spain Alessandro Farinelli, University of Southampton, UK Martin Friedmann, TU Darmstadt, Germany Holger Giese, Hasso Plattner Institute, Germany Giuseppina Gini, Politecnico di Milano, Italy Martin Huelse, University of Wales, UK Luca Iocchi, University of Rome La Sapienza, Italy Alexander Kleiner, Albert-Ludwigs-Universität Freiburg, Germany Gerhard Kraetzschmar, Bonn-Rhein-Sieg University, Germany Jochen Maas, TU Braunschweig, Germany Tekin Meriçli, Boğaziçi University, Turkey Olivier Michel, Cyberbotics, Switzerland Rezia Molfino, University of Genoa, Italy Maurizio Morisio, Politecnico di Torino, Italy Antonio Sgorbissa, University of Genoa, Italy Sergiu-Dan Stan, Technical University of Cluj-Napoca, Romania Sebastian Wrede, Bielefeld University, Germany Cezary Zielinski, Warsaw University of Technology, Poland

Additional Reviewers

R. Balan	R. Hebig	A. Pretto	Y. Suga
D. Calisi	B. Johnston	C.-R. Rad	F. Tomassetti
M. Coman	R. C. Keely	M. Sartori	T. Vogel
S. Ghidoni	T. Lens	A. Scalimato	H. Wang
M. Göbelbecker	F. Mastrogiovanni	F. Schaefer	

Sponsoring Institutions

Graduate School of Computational Engineering and Research Training Group
on Cooperative, Adaptive and Responsive Monitoring in Mixed Mode Environments
of Technische Universität Darmstadt

Table of Contents

Invited Talks

Building Blocks for Mobile Manipulation (Abstract)	1
<i>Brian P. Gerkey</i>	
Natural Human-Robot Interaction (Abstract)	2
<i>Takayuki Kanda</i>	
Robots and the Human (Abstract)	3
<i>Oussama Khatib</i>	
The Modelica Object-Oriented Equation-Based Language and Its OpenModelica Environment with MetaModeling, Interoperability, and Parallel Execution	5
<i>Peter Fritzson</i>	

Simulation

Blender for Robotics: Integration into the Leuven Paradigm for Robot Task Specification and Human Motion Estimation	15
<i>Koen Buys, Tinne De Laet, Ruben Smits, and Herman Bruyninckx</i>	
Simulating the C2SM ‘Fast’ Robot	26
<i>Robert Codd-Downey, Michael Jenkin, Matthew Ansell, Ho-Kong Ng, and Piotr Jasiobedzki</i>	
Extending Open Dynamics Engine for Robotics Simulation	38
<i>Evan Drumwright, John Hsu, Nathan Koenig, and Dylan Shell</i>	
Virtual Robot Experimentation Platform V-REP: A Versatile 3D Robot Simulator	51
<i>Marc Freese, Surya Singh, Fumio Ozaki, and Nobuto Matsuhira</i>	
Evaluation and Enhancement of Common Simulation Methods for Robotic Range Sensors	63
<i>Martin Friedmann, Karen Petersen, and Oskar von Stryk</i>	
High Fidelity Sensor Simulations for the Virtual Autonomous Navigation Environment	75
<i>Chris Goodin, Phillip J. Durst, Burkman Gates, Chris Cummins, and Jody Priddy</i>	
GPS/Galileo Testbed Using a High Precision Optical Positioning System	87
<i>Robin Heß and Klaus Schilling</i>	

Validating Vision and Robotic Algorithms for Dynamic Real World Environments	97
<i>Tobias Kotthäuser and Bärbel Mertsching</i>	
OpenGRASP: A Toolkit for Robot Grasping Simulation	109
<i>Beatrix León, Stefan Ulbrich, Rosen Diankov, Gustavo Puche, Markus Przybylski, Antonio Morales, Tamim Asfour, Sami Moisio, Jeannette Bohg, James Kuffner, and Rüdiger Dillmann</i>	
NERD Neurodynamics and Evolutionary Robotics Development Kit	121
<i>Christian Rempis, Verena Thomas, Ferry Bachmann, and Frank Pasemann</i>	
Simulation and Evaluation of Mixed-Mode Environments: Towards Higher Quality of Simulations	133
<i>Vinay Sachidananda, Diego Costantini, Christian Reirl, Dominik Haumann, Karen Petersen, Parag S. Mogre, and Abdelmajid Khelil</i>	
Evaluating a Physics Engine as an Ingredient for Physical Reasoning ...	144
<i>Erik Weitnauer, Robert Haschke, and Helge Ritter</i>	
Simulating Vehicle Kinematics with SimVis3D and Newton	156
<i>Jens Wettach, Daniel Schmidt, and Karsten Berns</i>	
Programming	
Coordinating Software Components in a Component-Based Architecture for Robotics	168
<i>Geoffrey Biggs, Noriaki Ando, and Tetsuo Kotoku</i>	
Native Robot Software Framework Inter-operation	180
<i>Geoffrey Biggs, Noriaki Ando, and Tetsuo Kotoku</i>	
Run-Time Management of Component-Based Robot Software from a Command Line	192
<i>Geoffrey Biggs, Noriaki Ando, and Tetsuo Kotoku</i>	
Modelling Behaviour Requirements for Automatic Interpretation, Simulation and Deployment	204
<i>David Billington, Vladimir Estivill-Castro, René Hexel, and Andrew Rock</i>	
Implementing Automated Robot Task Planning and Execution Based on Description Logic KB	217
<i>Joonmyun Cho, Hyun Kim, and Joochan Sohn</i>	

On the Way to High-Level Programming for Resource-Limited Embedded Systems with Golog	229
<i>Alexander Ferrein and Gerald Steinbauer</i>	
RobustHX – The Robust Middleware Library for Hexor Robots	241
<i>Konrad Kułakowski and Piotr Matyjasik</i>	
RoboComp: A Tool-Based Robotics Framework	251
<i>Luis Manso, Pilar Bachiller, Pablo Bustos, Pedro Núñez, Ramón Cintas, and Luis Calderita</i>	
Improving a Robotics Framework with Real-Time and High-Performance Features	263
<i>Jesús Martínez, Adrián Romero-Garcés, Luis Manso, and Pablo Bustos</i>	
Implementation of Distributed Production System for Heterogeneous Multiprocessor Robotic Systems	275
<i>Yosuke Matsusaka and Isao Hara</i>	
Robot Programming by Demonstration	288
<i>Krishna Kumar Narayanan, Luis Felipe Posada, Frank Hoffmann, and Torsten Bertram</i>	
Design Principles of the Component-Based Robot Software Framework Fawkes	300
<i>Tim Niemueller, Alexander Ferrein, Daniel Beck, and Gerhard Lakemeyer</i>	
Handling Hardware Heterogeneity through Rich Interfaces in a Component Model for Autonomous Robotics	312
<i>Olena Rogovchenko and Jacques Malenfant</i>	
Design Abstraction and Processes in Robotics: From Code-Driven to Model-Driven Engineering	324
<i>Christian Schlegel, Andreas Steck, Davide Brugali, and Alois Knoll</i>	

Applications

Using Simulation to Assess the Effectiveness of Pallet Stacking Methods	336
<i>Stephen Balakirsky, Fred Proctor, Tom Kramer, Pushkar Kolhe, and Henrik I. Christensen</i>	
Analysing Mixed Reality Simulation for Industrial Applications: A Case Study in the Development of a Robotic Screw Remover System	350
<i>Ian Yen-Hung Chen, Bruce MacDonald, Burkhard Wünsche, Geoffrey Biggs, and Tetsuo Kotoku</i>	

A Parameterless Biologically Inspired Control Algorithm Robust to Nonlinearities, Dead-Times and Low-Pass Filtering Effects	362
<i>Fabio DallaLibera, Shuhei Ikemoto, Takashi Minato, Hiroshi Ishiguro, Emanuele Menegatti, and Enrico Pagello</i>	
Exploration Strategies for a Robot with a Continuously Rotating 3D Scanner	374
<i>Elena Digor, Andreas Birk, and Andreas Nüchter</i>	
Validating an Active Stereo System Using USARSim	387
<i>Sebastian Drews, Sven Lange, and Peter Protzel</i>	
Performance Analysis for Multi-robot Exploration Strategies	399
<i>Sebastian Frank, Kim Listmann, Dominik Haumann, and Volker Willert</i>	
Dynamic Modeling of the 4 DoF BioRob Series Elastic Robot Arm for Simulation and Control	411
<i>Thomas Lens, Jürgen Kunz, and Oskar von Stryk</i>	
Static Balance for Rescue Robot Navigation: Discretizing Rotational Motion within Random Step Environment	423
<i>Evgeni Magid and Takashi Tsubouchi</i>	
Discovery, Localization and Recognition of Smart Objects by a Mobile Robot	436
<i>Emanuele Menegatti, Matteo Danielletto, Marco Mina, Alberto Pretto, Andrea Bardella, Andrea Zanella, and Pietro Zanuttigh</i>	
Simulation for the Optimal Design of a Biped Robot: Analysis of Energy Consumption	449
<i>Federico Moro, Giuseppina Gini, Milos Zefran, and Aleksandar Rodic</i>	
Efficient Use of 3D Environment Models for Mobile Robot Simulation and Localization	461
<i>Andreu Corominas Murtra, Eduard Trulls, Josep M. Mirats Tur, and Alberto Sanfeliu</i>	
Decision and Coordination Strategies for RoboCup Rescue Agents	473
<i>Maitreyi Nanjanath, Alexander J. Erlandson, Sean Andrist, Aravind Ragipindi, Abdul A. Mohammed, Ankur S. Sharma, and Maria Gini</i>	
Swarm Dispersion via Potential Fields, Leader Election, and Counting Hops	485
<i>Anuraag Pakanati and Maria Gini</i>	

Compliant Robot Actuation by Feedforward Controlled Emulated Spring Stiffness	497
<i>Katayon Radkhah, Stefan Kurowski, Thomas Lens, and Oskar von Stryk</i>	
Different Approaches in Feeding of a Flexible Manufacturing Cell	509
<i>Claudia Raluca Tudorie</i>	
On the Stability of Bipedal Walking	521
<i>Pieter van Zutven, Dragan Kostić, and Henk Nijmeijer</i>	
An Approach to Close the Gap between Simulation and Real Robots ...	533
<i>Yuan Xu, Heinrich Mellmann, and Hans-Dieter Burkhard</i>	
Accelerating Point-Based POMDP Algorithms via Greedy Strategies ...	545
<i>Zongzhang Zhang and Xiaoping Chen</i>	
Author Index	557

Building Blocks for Mobile Manipulation

Brian P. Gerkey

Willow Garage

68 Willow Road, Menlo Park, California, USA

gerkey@willowgarage.com

<http://www.willowgarage.com>

Abstract. Following a long period of separate development, the twin fields of mobile robots and robotic manipulators have in recent years combined to form a new area of study, mobile manipulation. Research in mobile manipulation offers the promise of robots that can operate in workplaces and homes, with little or no modification, assisting people with everyday tasks. To realize that promise, there are significant scientific and engineering challenges to overcome.

In this talk, I will discuss how we, at Willow Garage, approach both the research and development of mobile manipulation. At the core of our approach are two engineering principles: (i) build reusable capabilities, and (ii) test what you build. While these are basic, even obvious, ideas in a development context, we apply them to great effect in our research projects. We make extensive use of simulation, building automated regression tests that run in simulation to verify the functionality of key capabilities (e.g., Fig. ①). We follow the maxim, “it’s tested or it’s broken.”

I will also discuss the importance of openness in research, with a particular focus on sharing, validating, and reusing each others’ work. By default, code that we write, notably the ROS infrastructure (<http://ros.org>), is distributed under an open source license. By releasing the code that supports our published experimental results, and encouraging others to do the same, we aim to improve scientific practice in our field.

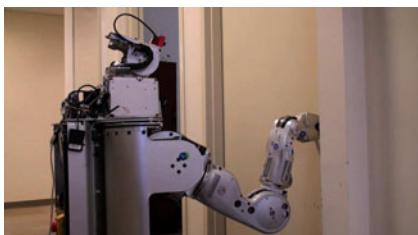


Fig. 1. Opening doors is a key capability. To ensure that the door-opening code will continue to work on the real robot (left), we test every change in simulation (right).

Natural Human-Robot Interaction

Takayuki Kanda

ATR Intelligent Robotics & Communication Labs
Kyoto 619-0288, Japan

There are a lot of human-like robots developed recently. Researchers have started to explore the way to make human-like robots interact with humans in a similar way as humans do, that is, natural human-robot interaction (HRI). We have faced the lack of knowledge on human behavior in interaction. There are existing literatures for modeling humans' information processing in conscious, such as about language understanding, and generation of utterance and gesture. However, when we try to build a model of natural interaction for robots, we faced a difficulty from the fact that natural interaction involves a lot of humans' unconscious information processing. Without much conscious efforts, people communicate with each other with naturally using body properties, such as gaze and gestures; but without explicit model of thorough information processing, we cannot build robots that engage in natural HRI. This talk introduces a series of studies for modeling human behavior for natural human-robot interaction. Built on existing knowledge in psychology and cognitive science, we have started to build models where observation of people's behavior were also contributed a lot. We have modeled behaviors for robots for interacting with people as well as for behaving in an environment. Moreover, we have also explored a case where behavior in interaction is inter-related with environments.

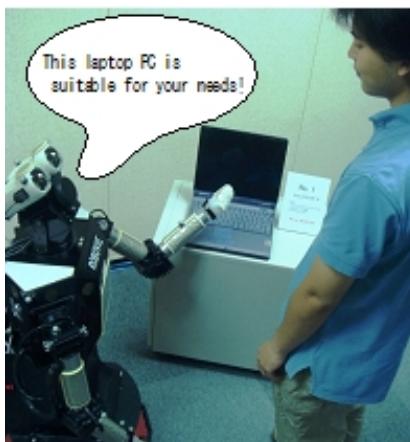


Fig. 1. A scene of interaction with a robot

Robots and the Human

Oussama Khatib

Artificial Intelligence Laboratory
Department of Computer Science
Stanford University, Stanford, CA 94305

Abstract. Robotics is rapidly expanding into the human environment and vigorously staying engaged in its emerging challenges. From a largely dominant industrial focus, robotics has undergone, by the turn of the new millennium, a major transformation in scope and dimensions. This expansion has been brought about by the maturity of the field and the advances in its related technologies to address the pressing needs for human-centered robotic applications. Interacting, exploring, and working with humans, the new generation of robots will increasingly touch people and their lives, in homes, workplaces, and communities, providing support in services, entertainment, education, manufacturing, personal health care, and assistance. The successful introduction of robots in human environments will rely on the development of competent and practical systems that are dependable, safe, and easy to use. This presentation focuses on the efforts to develop human-friendly robotic systems that combine the essential characteristics of safety, human-compatibility, and performance with emphasis on (i) new design concepts and novel sensing modalities; (ii) efficient planning and whole-body robot control strategies; and (iii) robotic-based synthesis of human motion and skills.

In human-friendly robot design, our effort has focused on the development of intrinsically safe robotic systems that possess the requisite capabilities and performance to interact and work with humans. The novel design concept was based on a hybrid actuation approach that consists of biomimetic pneumatic muscles combined with small electric motors. The flexible muscles and the lightweight mechanism allow for human safety, while the electric motors compensate for the slower dynamics and nonlinearities of the pneumatics. This concept was shown to significantly decrease the inherent danger of robotic manipulators, as measured in terms of the reflected mass perceived at the point of impact. Safety can be further enhanced by the addition of robot skin to provide impact reduction and tactile sensing capabilities for advanced sensor based behaviors.

Redundancy is a major challenge in the planning and control of humanoid robots. Inspired by human behaviors, our early work in robot control encoded tasks and diverse constraints into artificial potential fields capturing human-like goal-driven behaviors. To implement such behaviors on robots with complex human-like structures we developed a unified whole-body task-oriented control structure that addresses dynamics in the context of multiple tasks, multi-point contacts, and multiple constraints. The performance and effectiveness of this approach have

been demonstrated through extensive robot dynamic simulations and implementations on physical robots for experimental validation. The new framework provides a multi-task prioritized control architecture allowing the simultaneous execution of multiple objectives in a hierarchical manner, analogous to natural human motion.

Initially motivated by the development of human-like skills in robotics, our extensive study of human musculoskeletal system has brought insights and results that proved extremely valuable in human biomechanics. Understanding human motion is a complex procedure that requires accurate reconstruction of movement sequences, modeling of musculoskeletal kinematics, dynamics, and actuation, and suitable criteria for the characterization of performance. These issues have much in common with the problems of articulated body systems studied in robotics research. Building on methodologies and techniques developed in robotics, a host of new effective tools have been established for the synthesis of human motion. These include efficient algorithms for the simulation of musculoskeletal systems, novel physio-mechanical criteria and performance measures, real-time tracking and reconstruction of human motion, and accurate human performance characterization. These developments are providing new avenues for exploring human motion – with exciting prospects for novel clinical therapies, athletic training, and performance improvement.

The Modelica Object-Oriented Equation-Based Language and Its OpenModelica Environment with MetaModeling, Interoperability, and Parallel Execution

Peter Fritzson

PELAB Programming Environments Lab, Department of Computer Science
Linköping University, SE-581 83 Linköping, Sweden
peter.fritzson@liu.se

Abstract. Modelica is a modern, strongly typed, declarative, equation-based, and object-oriented language for modeling and simulation of complex systems. Major features are: ease of use, visual design of models with combination of lego-like predefined model building blocks, ability to define model libraries with reusable components, support for modeling and simulation of complex applications involving parts from several application domains, and many more useful facilities. This paper gives an overview of some aspects of the Modelica language and the OpenModelica environment – an open-source environment for modeling, simulation, and development of Modelica applications. Special features are MetaModeling for efficient model transformations and prototype generation of parallel code for multi-core architectures. Ongoing work also include UML-Modelica integration and interoperability support through the new FMI standard.

1 Introduction

A large number of modeling and simulation tools are available on the market. Most tools are very specialized and only address a specific application domain. These tools usually have incompatible model formats.

However, certain tools are of a more general nature. For modeling and simulation of dynamic systems, we have e.g. the following de-facto standards:

- Continuous: Matlab/Simulink, [28], MatrixX/ SystemBuild, Scilab/Scicos, ACSL,... for general systems, SPICE and its derivates for electrical circuits, ADAMS, DADS/Motion, SimPack, SimMechanics,... for multi-body mechanical systems, ANSYS, FEMLAB, etc. for finite-element analysis, ...
- Discrete: general-purpose simulators based on the discrete-event GPSS line, VHDL- and Verilog simulators in digital electronics, etc.
- Hybrid (discrete + continuous): Modelica, gPROMS [19], AnyLogic, VHDL-AMS and Verilog-AMS simulators (not only for electronics but also for multi-physics problems), etc.

The insufficient power and generality of the former modeling languages has stimulated the development of Modelica (as a true object-oriented, multi-physics language) and VHDL-AMS/Verilog-AMS (multi-physics but primarily for electronics, and therefore often not general enough).

1.1 Modelica Background

In 1996 several first generation object-oriented mathematical modeling languages and simulation systems (ObjectMath [25], Dymola [23] [24], Omola [29], NMF [32], gPROMS [19], Allan, Smile, etc.) had been developed.

However, the situation with a number of different incompatible object-oriented modeling and simulation languages was not satisfactory. Therefore, in the fall of 1996, a group of researchers from universities and industry started work towards standardization and making this object-oriented modeling technology widely available. This language is called Modelica [2][1][33] and designed primarily for modeling dynamic behavior of engineering systems; moreover, meta-modeling extensions have been developed [26]. The language is intended to become a de facto standard.

The language allows defining models in a declarative manner, modularly and hierarchically and combining various formalisms expressible in the more general Modelica formalism. The multidomain capability of Modelica allows combining electrical, mechanical, hydraulic, thermodynamic, etc., model components within the same application model.

Compared to most widespread simulation languages available today this language offers several important advantages:

- *Object-oriented mathematical modeling.* This technique makes it possible to create model components, which are employed to support hierarchical structuring, reuse, and evolution of large and complex models covering multiple technology domains.
- *Acausal modeling.* Modeling is based on equations instead of assignment statements as in traditional input/output block abstractions. Direct use of equations significantly increases re-usability of model components, since components adapt to the data flow context in which they are used. Interfacing with traditional software is also available in Modelica.
- *Physical modeling of multiple application domains.* Model components can correspond to physical objects in the real world, in contrast to established techniques that require conversion to “signal” blocks with fixed input/output causality. In Modelica the structure of the model naturally correspond to the structure of the physical system in contrast to block-oriented modeling tools such as Simulink. For application engineers, such “physical” components are particularly easy to combine into simulation models using a graphical editor.
- A *general type system* that unifies object-orientation, multiple inheritance, components/connectors, and templates/generics within a single class construct.

Hierarchical system architectures can easily be described with Modelica thanks to its powerful component model. Components are connected via the connection mechanism realized by the Modelica system, which can be visualized in connection diagrams. The component framework realizes components and connections, and ensures that communication works over the connections.

For systems composed of acausal components with behavior specified by equations, the direction of data flow, i.e., the causality is initially unspecified for connections between those components. Instead the causality is automatically deduced by the compiler when needed. Components have well-defined interfaces consisting of ports, also known as connectors, to the external world. A component may internally consist of other connected components, i.e., hierarchical modeling. Fig. 1 shows hierarchical component-based modeling of an industry robot. More information on Modelica robot models can for example be found in [35], [36], [37], and [38]. Methods for real-time simulation, e.g. in [39].

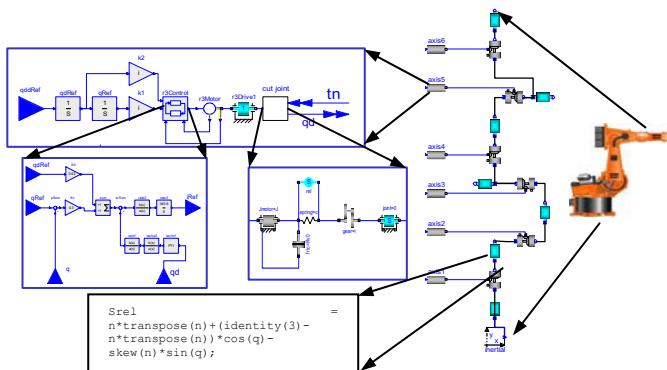


Fig. 1. Hierarchical model of an industrial robot, including components such as motors, bearings, control software, etc. At the lowest (class) level, equations are typically found.

The language design and model library development has proceeded through a number of design meetings, a nonprofit Modelica Association was started in Linköping, Sweden year 2000, and a conference series was started the same year, with the 5th conference in Vienna Sept. 2006. Several commercial implementations of Modelica (i.e., subsets thereof) are available, including Dymola [22], MathModelica [27], and IDA [20]. This presentation primarily focuses on the OpenModelica, which is currently the major Modelica open-source tool effort and most complete open source environment.

1.2 An Example Modelica Model

The following is an example Lotka Volterra Modelica model containing two differential equations relating the sizes of rabbit and fox populations which are represented by the variables `rabbits` and `foxes`: The rabbits multiply; the foxes eat rabbits. Eventually there are enough foxes eating rabbits causing a decrease in the rabbit population, etc., causing cyclic population sizes. The model is simulated and the sizes of the rabbit and fox populations as a function of time are plotted in Fig. 2.

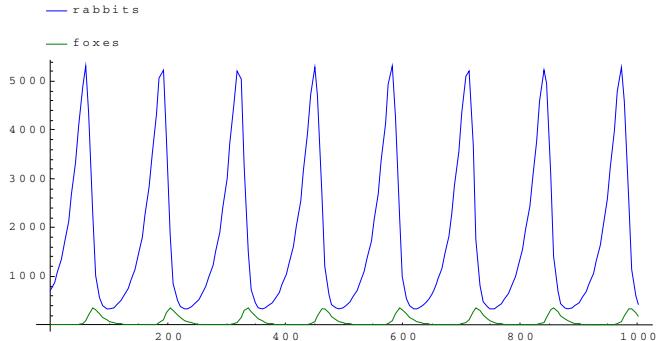


Fig. 2. Number of rabbits – prey animals, and foxes - predators, as a function of time simulated from the PredatorPrey model

The notation `der(rabbits)` means time derivative of the `rabbits` (population) variable.

```

class LotkaVolterra
  parameter Real g_r =0.04      "Natural growth rate for rabbits";
  parameter Real d_rf=0.0005  "Death rate of rabbits due to foxes";
  parameter Real d_f =0.09    "Natural deathrate for foxes";
  parameter Real g_fr=0.1   "Efficiency in growing foxes fr rabbits";
  Real rabbits(start=700) "Rabbits,(R) with start population 700";
  Real foxes(start=10)    "Foxes,(F) with start population 10";
equation
  der(rabbits) = g_r*rabbits - d_rf*rabbits*foxes;
  der(foxes)   = g_fr*d_rf*rabbits*foxes - d_f*foxes;
end LotkaVolterra;
```

2 The OpenModelica Environment

The OpenModelica open source environment [4], [5] project supported by the Open Source Modelica Consortium has several goals, including, but not limited to the following:

- Providing an efficient interactive computational environment for the Modelica language, for both academic and industrial usage.
- Development of a complete reference implementation of Modelica in MetaModelica, an extended version of Modelica itself.
- Providing an environment for teaching modeling and simulation. It turns out that with support of appropriate tools and libraries, Modelica is very well suited as a computational language for development and execution of both low level and high level numerical algorithms, e.g. for control system design, solving nonlinear equation systems, or to develop optimization algorithms for complex applications.
- *Language design*, e.g. to further extend the scope of the language, e.g. for use in diagnosis, structural analysis, system identification, etc., as well as modeling

problems that require extensions such as partial differential equations, enlarged scope for discrete modeling and simulation, etc.

- *Language design* to improve abstract properties such as expressiveness, orthogonality, declarativity, reuse, configurability, architectural properties, etc.
- *Improved implementation techniques*, e.g. to enhance the performance of compiled Modelica code by generating code for parallel hardware.
- *Improved debugging support* for equation based languages such as Modelica, for improved ease of use.
- *Easy-to-use* specialized high-level (graphical) *user interfaces* for certain application domains.
- *Application usage* and model library development by researchers in various application areas.

3 MetaModelica – A Unified Equation-Based Modeling Language

The idea to define a unified equation-based mathematical and semantic modeling language started from the development of the OpenModelica compiler [5]. The entire compiler is generated from a Natural Semantics specification written in RML [34], a compiler generator for Natural Semantics. The open source OpenModelica compiler has its users in the Modelica community which have detailed knowledge of Modelica but very little knowledge of RML and Natural Semantics. In order to allow people from the Modelica community to contribute to the OpenModelica compiler we retargeted the development language from RML to MetaModelica, which is based on the Modelica language with several extensions. We already translated the OpenModelica compiler from RML to the MetaModelica using an automated translator. We also developed a compiler which can handle the entire OpenModelica compiler specification (~160 000 lines of code) defined in MetaModelica.

The basic idea behind the unified language is to use equations as the unifying feature. Most declarative formalisms, including functional languages, support some kind of limited equations even though people often do not regard these as equations, e.g. single-assignment equations.

Using the meta-programming facilities, usual tasks like generation, composition and querying of Modelica models can be automated. MetaModelica supports e.g. efficient pattern matching and transformation of models represented as abstract syntax trees.

4 Integrated UML-Modelica for Whole-Product Modeling

Complex products are increasingly made of both software and hardware components which are closely interacting. Thus, modeling tools and processes need to support co-design of software and hardware in an integrated way. This is the main goal addressed by the OPENPROD project [3].

Currently, UML is the dominant graphical modeling notation for software, whereas Modelica is the major object-oriented mathematical modeling language for component-oriented modeling of complex physical systems, e.g., systems containing

mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents.

For these reasons we developed the first generation prototype UML-Modelica-SysML integrated modeling environment [16] as a ModelicaML UML profile integrated in Eclipse as a plug-in. The profile reuses some artifacts from the System Modeling Language (SysML) profile, and combines the major UML diagrams with Modelica graphic connection diagrams. Requirement, equation, and simulation diagrams are also supported in an integrated way.

The second generation ModelicaML profile prototype [6][7] is more comprehensive and more precisely defined. It is based directly on UML, implemented using the Papyrus UML tool, generates executable Modelica code, and also supports activity diagrams and requirements diagrams. It is possible to model integrated software/hardware models, specify requirements, generate executable Modelica code, and simulate the model while checking if the requirements are fulfilled or violated. A recent further development in ModelicaML [8] supports generation of executable Modelica from UML state charts, e.g. for use in embedded systems modeling.

A related ongoing effort [9] by the OMG SysML-Modelica task force is to define an executable SysML with precise semantics by mapping it to Modelica

5 Parallel Execution on Multi-core Platforms

A common problem in simulation is too long computing time due to increasingly complex models. The new multi-core platforms provide more computing power, that however typically is not that easy to use since these platforms usually require significant restructuring and re-programming, making the programming task even more complex and increasing the risk of introducing errors. Automatic or semi-automatic parallelization is needed, but this a challenging problem to solve for realistic applications.

However, starting from a mathematical model at a high level of abstraction gives more freedom for the compiler to perform parallelization and avoids many low-level programming details that may hinder parallelization. This approach has been investigated by us, e.g. in [10], and involves extracting a sufficient amount of parallelism from equations represented in the form of a data dependence graph (task graph), requiring analysis of the code at a detailed level. Efficient clustering and merging of fine-grained tasks is essential to avoid communication overhead. Speedup results from the preliminary OpenModelica prototype on fast Linux clusters and Nvidia GPUs are encouraging.

A recent development [10] uses a combination of software pipelining and solver inlining to achieve good speedups for compiling a nonlinear mechanical Modelica model to an Intel multi-core platform. We are currently re-designing and re-implementing this prototype in order to increase scalability and support parallelization of larger models.

Work is also ongoing to extend the automatic parallelization to support efficient event-handling in parallelized code in order to handle hybrid continuous-discrete models, aiming at several target real-time execution platforms available in the OPENPROD project. Coarse-grained manual hints using transmission line modeling connections [13] can partition the simulation model for simulation on different

processors. Thus, both coarse-grained manual partitioning and automatic fine-grained parallelization can be combined.

6 Tool Interoperability through Functional Mockup Interface

A new open standard for model interchange and tool interoperability which simplifies whole product modeling has recently been released [14]¹, developed in the ITEA2 MODELISAR project [15]. This open standard is called FMI – Functional Mockup Interface. The current release is FMI for model exchange 1.0.

The intention of FMI is that a modeling environment can generate C-Code from a dynamic system model that can be exported to other modeling and simulation environments either in source or binary form.

Models are described by differential, algebraic and discrete equations with time-, state- and step-events. In particular, all Modelica 3.2 models are supported and all Modelica variable attributes (like units and display units) as well as description texts can be exchanged. The models to be treated by this interface can be large for usage in offline or online simulation or can be used in embedded control systems on micro-processors. It is possible to utilize several instances of a model and to connect models hierarchically.

The MODELISAR consortium plans to publish additional parts of the FMI, especially for coupling of different simulation tools via co-simulation and for coupling of simulators with applications like testing, optimization, and product lifecycle management. An open-source implementation of FMI has been started in the OPENPROD project.

7 Conclusions

This paper has given a short overview of the Modelica language, its background and the most important features of the language, as well as the OpenModelica open source environment. This environment includes many valuable features for engineers and researchers, and it is the only Modelica environment so far with good support for debugging Modelica algorithmic code as well as support for meta-programming integrated in the language. Prototype parallel code generation was also mentioned. We believe that this open source platform can be part of forming the foundation of the next generation of the Modelica language and environment development efforts, both from a research perspective and a system engineering usage point of view.

We have also presented ongoing work in integrating UML and Modelica in ModelicaML for model-based product development in the OPENPROD project, as well as the FMI standard on tool and model interoperability developed in the MODELISAR project. By basing an executable subset of UML on Modelica as an action language in ModelicaML, with translation of state charts or activity diagrams into Modelica, executable modeling of whole products with both software and hardware components is possible.

¹ Most of the text in this section is cited from [14].

Acknowledgements

Vinnova, the Swedish Governmental Agency for Innovation Systems, supports Linköping University in the ITEA2 OPENPROD project. Support is also received from VR and SSF.

References

- [1] Fritzson, P.: Principles of Object-Oriented Modeling and Simulation with Modelica 2.1., p. 940. Wiley-IEEE Press, Chichester (2004)
- [2] The Modelica Association. The Modelica Language Specification Version 3.2 (May 2010), <http://www.modelica.org>
- [3] OPENPROD project web page, <http://www.openprod.org> and http://www.itea2.org/public/project_leaflets/OPENPROD_profile_oct-09.pdf (accessed May 2010)
- [4] OSMC. The Open Source Modelica Consortium, <http://www.openmodelica.org> (accessed September 2010)
- [5] Fritzson, P., Aronsson, P., Lundvall, H., Nyström, K., Pop, A., Saldamli, L., Broman, D.: The OpenModelica Modeling, Simulation, and Software Development Environment. In: Simulation News Europe (SNE), vol. 44 (January 2006), <http://www.openmodelica.org>
- [6] Schamai, W.: Modelica Modeling Language (ModelicaML) A UML Profile for Modelica, Technical report 2009:5, Linkoping University, Sweden (2009), <http://www.ep.liu.se>
- [7] Schamai, W., Fritzson, P., Paredis, C., Pop, A.: Towards Unified System Modeling and Simulation with ModelicaML Modeling of Executable Behavior Using Graphical Notations. In: Proceedings of the 7th International Modelica Conference (Modelica 2009), Como, Italy (September 20-22, 2009), <http://www.modelica.org>
- [8] Schamai, W., Pohlmann, U., Fritzson, P., Paredis, C., Helle, P., Strobel, C.: Execution of UML State Machines Using Modelica. Accepted to EOOLT 2010 (June 2010), <http://www.eoolt.org>
- [9] Bernard, Y., Burkhardt, R.M., de Koning, H.-P., Friedenthal, S., Fritzson, P., Paredis, C., Rouquette, N.F., Schamai, W.: An Overview of the SysML-Modelica Transformation Specification. In: Proc. of INCOSE 2010, 20th Annual INCOSE Int. Symposium, Chicago, USA, July 11-15 (2010), <http://www.incose.org>
- [10] Aronsson, P.: Automatic Parallelization of Equation-Based Simulation Programs. PhD thesis, Dissertation No. 1022, Dept, Computer and Information Science, Linköping University, Linköping, Sweden
- [11] Lundvall, H., Fritzson, P.: Automatic Parallelization using Pipelining for Equation-Based Simulation Languages. In: Proc. of the 14th Workshop on Compilers for Parallel Computing (CPC 2009), Zurich, Switzerland, (January 7-9, 2009)
- [12] Fritzson, P., Aronsson, P., Lundvall, H., Nyström, K., Pop, A., Saldamli, L., Broman, D.: The OpenModelica Modeling, Simulation, and Software Development Environment. In: Simulation News Europe, vol. 44(45) (December 2005), <http://www.openmodelica.org>
- [13] Nyström, K., Fritzson, P.: Parallel Simulation with Transmission Lines in Modelica. In: Proceedings of the 5th International Modelica Conference (Modelica 2006), Vienna, Austria, (September 4-5, 2006)

- [14] Otter, M.: Functional Mockup Interface (FMI) for Model Exchange. In: Modelica Newsletter 2010-1 (April 2010), <http://www.modelica.org>, <http://www.functional-mockup-interface.org>
- [15] MODELISAR itea2 project, http://www.itea2.org/public/project_leaflets/MODELISAR_project_profile_oct-08.pdf (accessed May 2010)
- [16] Pop, A., Akhvelidiani, D., Fritzson, P.: Integrated UML and Modelica System Modeling with ModelicaML in Eclipse. In: Proceedings of the 11th IASTED International Conference on Software Engineering and Applications (SEA 2007), Cambridge, MA, USA (November 19-21, 2007)
- [17] Papyrus UML, <http://www.papyrusuml.org>
- [18] OMG. OMG Unified Modeling Language™ (OMG UML). Superstructure Version 2.2 (February 2009)
- [19] Barton, P., Pantelides, C.: The Modelling of Combined Discrete/Continuous Processes. AIChE Journal 40, 966–979 (1994)
- [20] Equa, A.B.: The IDA simulation tool, <http://www.equa.se> (accessed 2006)
- [21] Christen, E., Bakalar, K.: VHDL-AMS—A Hardware Description Language for Analog and Mixed-Signal Applications. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 46(10), 1263–1272 (1999)
- [22] Dynasim AB. Dymola—Dynamic Modeling Laboratory with Modelica, Users Manual, Version 6.0. Dynasim AB, Research Park Ideon, SE-223 70, Lund, Sweden (2006)
- [23] Elmquist, H.: A Structured Model Language for Large Continuous Systems. Ph.D. thesis, TFRT-1015, Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden (1978)
- [24] Elmquist, H., Bruck, D., Otter, M.: Dymola—User’s Manual. Dynasim AB, Research Park Ideon, SE-223 70, Lund, Sweden (1996)
- [25] Fritzson, P., Wiklund, L., Fritzson, D., Herber, J.: High-Level Mathematical Modelling and Programming. IEEE Software 12(4), 77–87 (1995), <http://www.ida.liu.se/labs/pelab/omath>
- [26] Fritzson, P., Pop, A., Aronsson, P.: Towards Comprehensive Meta-Modeling and Meta-Programming Capabilities in Modelica. In: Proceedings of the 4th International Modelica Conference, Hamburg, Germany (March 7-8, 2005)
- [27] MathCore Engineering AB. MathModelica User’s Guide (2010), <http://www.mathcore.com>
- [28] MathWorks. The Mathworks - Simulink - Simulation and Model-Based Design, <http://www.mathworks.com/products/simulink/> (last accessed: May 15, 2006)
- [29] Mattsson, S.-E., Andersson, M.: The Ideas Behind Omola. In: Proceedings of the 1992 IEEE Symposium on Computer-Aided Control System Design (CADC 1992), Napa, California (March 1992)
- [30] Pop, A., Fritzson, P.: A Portable Debugger for Algorithmic Modelica Code. In: Proceedings of the 4th International Modelica Conference, Hamburg, Germany (March 7-8, 2005)
- [31] Pop, A., Fritzson, P., Remar, A., Jagudin, E., Akhvelidiani, D.: Open-Modelica Development Environment with Eclipse Integration for Browsing, Modeling, and Debugging. In: Proc. of Modelica 2006, the 5th Int. Modelica Conf., Vienna, September 4-5 (2006)
- [32] Sahlin, P.: Modelling and Simulation Methods for Modular Continuous Systems in Buildings. Ph.D. thesis, Dept. of Building Science, Royal Inst. of Technology Stockholm, Sweden (May 1996)

- [33] Tiller, M.: Introduction to Physical Modeling with Modelica, p. 366. Kluwer Academic Publishers, Dordrecht (2001)
- [34] Pettersson, M.: Compiling Natural Semantics. LNCS, vol. 1549. Springer, Heidelberg (1999)
- [35] Hirzinger, G., Bals, J., Otter, M., Stelter, J.: The DLR-KUKA Success Story: Robotics Research Improves Industrial Robots. IEEE Robotics & Automation Magazine 12(3), 16–23 (2005)
- [36] Ferretti, G., Gritti, M., Magnani, G., Rocco, P.: A Remote User Interface to Modelica Robot Models. In: Fritzson, P. (ed.) Proceedings of the 3rd International Modelica Conference, Linköping (November 3-4, 2003),
http://www.modelica.org/events/Conference2003/papers/h23_Gritti.pdf
- [37] Frenkel, J., Schubert, C., Kunze, G., Jankov, K.: Using Modelica for Interactive Simulations of Technical Systems in a Virtual Reality Environment. In: Proceedings of the 7th International Modelica Conference, Como, Italy. Linköping Univ. Electronic Press (September 20-22, 2009),
<http://www.ep.liu.se/ecp/043/061/ecp09430080.pdf>
- [38] Reiner, M., Heckmann, A., Otter, M.: Inversion based control of flexible body systems. In: Proceedings of the 9th Conference on Motion and Vibration Control, MOVIC 2008, Munich (September 15, 2008)
- [39] Sjölund, M., Braun, R., Fritzson, P., Krus, P.: Towards Efficient Distributed Simulation in Modelica using Transmission Line Modeling. In: Proceedings of the 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, EOOLT 2010, In Conjunction with MODELS 2010, Oslo, Norway. Linköping University Electronic Press (October 3, 2010), <http://www.ep.liu.se>

Blender for Robotics: Integration into the Leuven Paradigm for Robot Task Specification and Human Motion Estimation

Koen Buys, Tinne De Laet, Ruben Smits, and Herman Bruyninckx

Department of Mechanical Engineering
Katholieke Universiteit Leuven, Belgium
koen.buys@mech.kuleuven.be
<http://www.mech.kuleuven.be/robotics>

Abstract. A growing number of robotics labs cooperate in the “Blender for Robotics” project [4], to integrate the *open source* computer animation program Blender into their research. In the first place as a visualisation tool, but gradually more and more as a full-fledged component in a robot simulation and programming environment. The lab in Leuven focuses on improving Blender support for the task specification and control of complex, sensor-based robot tasks, in which also the motion of the human(s) interacting with the task must be tracked.

1 Introduction

A lot of tools already exist in the robotics community to simulate robots in virtual environments and to visualise actual data in such a virtual world. Most of them were created for a specific research project or to interface a specific robotics framework. One of the most popular free and open-source software (FOSS) projects is *Player*, with the 2D Stage [13] simulator, and the 3D Gazebo [14] simulator; both have been accepted as core modules in the “Robot Operating System” ROS [27], which is rapidly becoming the number one software framework in robotics. So, why spend efforts to build *yet another* visualisation and simulation tool, with Blender [19] at its basis? The major reasons are the following significant differences with most robotics software frameworks: Blender is a well-established project (with a development history of over 15 years) with larger user and (professional!) developer communities than any robotics project; Blender runs on all major operating systems; its code base is, by design, extremely modular, so that it can accomodate different solutions for the same functionality, such as rendering, physics engines (Bullet [6], ODE [21], and others), or kinematics; its Python language bindings are extensive, and allow for flexible interfacing with external programs; the currently developed version 2.5 provides extensive support for hardware-accelerated computations (graphics in the first place), for the creation of application-specific interface panels, and for state of the art inverse kinematics of redundant tree-structured robots.

Section 2 gives an overview of K.U.Leuven’s contributions. They are illustrated by an example in Section 3. Section 4 concludes the paper with a discussion and a look at future work.

2 Contributions

The first tangible result of K.U.Leuven’s contributions has been the work of Benoit Bolsée, who implemented the iTaSC (instantaneous Task Specification using Constraints) framework [8][24][23] in Blender 2.5. This contribution lays the foundations of a complete physics *simulator* (and not just a *visualisation!*) of *kinematic chains* and their interactions with objects in the environment and has already shown to outperform the inverse kinematics coming with Blender (<http://vimeo.com/groups/blenderandrobotics/videos/5857397>).

In this paper we show how to use Blender not only as a computer animation package but also as a robotics toolbox. Both as a *visualisation* tool (Section 2.1), that is, making real world *experimental* data visible in a graphical way, but also as a *simulator* (Section 2.2) that can *create* virtual sensor data. In addition, we also show the *middleware* interaction possibilities (Section 2.3) of Blender with various robotic frameworks as Open Robot Control Software (Orocos) [5], Robot Operating System (ROS) [18], GenoM [11][15], YARP [10], ...

2.1 Visualisation Tool

Visualising in a graphically appealing way the large amount of data that is generated during typical robotics experiments is a major requirement of most robotics projects. To this end, the visualiser must provide the same interface as the robot controller. To this end, the interface should ideally provide a similar interface as the robot controller to allow for off-line visualisation of algorithms.

The visualisation feature of Blender can be used to get a graphical impression of the motion of robots. Figure 1 shows a snapshot of a rendering of one of K.U.Leuven’s industrial robot arms. To visualise the motion execution by the robot, the kinematic chain (or *armature*) underlying the industrial robot is modeled using *bones*, and the *mesh* defining the robot’s appearance is attached to this kinematic chain (Figure 2). In this example we use an external control program that calculates the robot’s joints using inverse kinematics (via the Kinematics and Dynamics Library [22]). These robot joint values are sent to a Python using a specific middleware. The Python script is attached to a Blender-native *sensor-controller-actuator* system, and to each “sample time” of the Blender Game Engine (BGE) rendering loop. This Python script moves the bones (and hence the attached mesh) of the Blender-native kinematic chain.

The visualisation feature of Blender is also illustrated in the markerless human motion capture example of Section 3, where the estimated joint angles coming from a *particle filter* are passed on to the Blender visualisation for visual verification. A video illustrating this can be found here: <http://people.mech.kuleuven.be/~u0062536/files/movies/simpar/visu/>



Fig. 1. Visualisation of an industrial robot

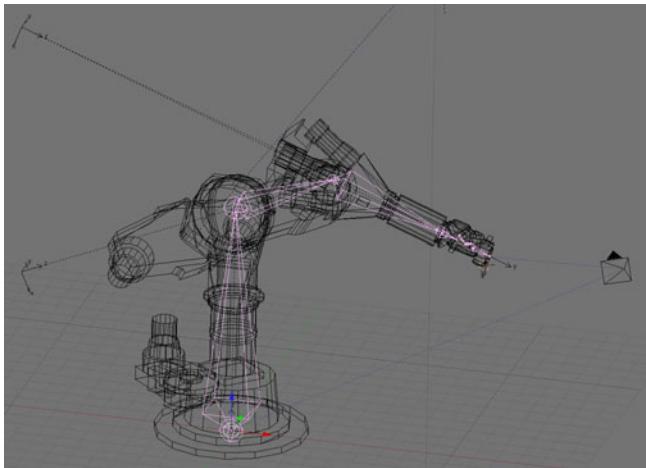


Fig. 2. The internal kinematic chain (of the Blender *armature* of the robot in Fig. 1)

2.2 Simulator

In addition to being a mere visualisation tool, the BGE can also be used as a *simulator* environment. This requires a full-duplex communication over the middleware as shown in Figure 3: the software algorithm running outside of Blender gets simulated sensor data from the simulator and sends its results back in order to let the BGE update the virtual environment within Blender.

At the moment the following sensors can be simulated by the BGE (thanks to the hard work at the LAAS and ONERA research labs, for videos see [9]): a



Fig. 3. Simulator blockdiagram. External programs must be given a bi-directional communication with Blender.

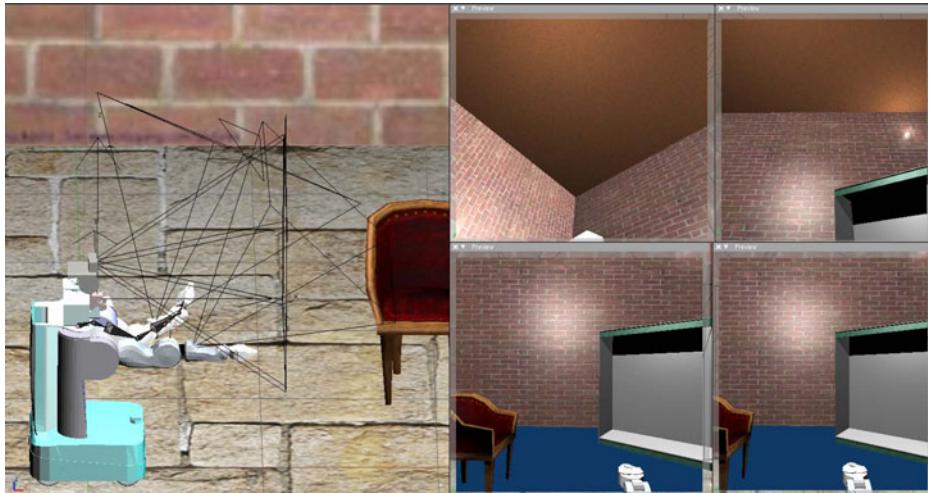


Fig. 4. On the left PR2 with its internal left arm kinematic chain and camera frames shown and on the right showing its camera output (Arm cameras and stereo head) to the ROS middleware

camera with configurable intrinsic and extrinsic parameters (Figure 4), a laser range finder, an inertial measurement unit (IMU), and a gyroscope; other sensors are in development.

The user has different possibilities to control a kinematic chain. First he has to decide whether to send end-effector positions or joint angles. In case end-effector positions are send, the user has the choice between the Blenders internal inverse kinematics (IK) solver functions or his own IK-solver and control the armature based on joint angles and translation vectors. For the first option we added a new IK-solver to Blender 2.5 which is able to solve based on the iTaSC framework [8][24][23]. For the second option we illustrated two different ways in our code: the first based on armature control, the second based on mesh control.

The second choice the user has to make concerning the control of a kinematic chain is on how to obtain the control signals. First he can make use of the BGE control structure, which allows him to control a robot with a keyboard, mouse, ... For this he needs to use the logic bricks that Blender provides (figure 5). Second, the user can also control the kinematic chain from a python module that connects over the middleware to an external program. This possibility is handled in section 2.3.

These accomplishments are illustrated in the markerless human motion capture example in which Blender is used as a 3D render engine to render the simulated expected measurements. In this example, the sensor feedback consists of a virtual camera image. A second example shows the Willow Garage PR2 robot simulated in Blender and gathering sensor data. A video illustrating this can be found here: <http://people.mech.kuleuven.be/~u0062536/files/movies/simpar/simu/>

2.3 Middleware

Because of the good integration of Python with the BGE, it is rather easy to connect the BGE to external programs via different types of middleware. This variety of middleware choices allows users to stay within the frameworks they are used to (as long as there are Python bindings for their preferred middleware). Python scripts are attached to the *logic block* of a BGE *controller* in the BGE logic panel which is shown in Figure 5. This panel is executed once in each BGE process loop, and allows programmers to trigger reactions based on some sensor input. These inputs are basically joystick, keyboard, distance, collisions,... within the original Blender context; but in the “Blender for Robotics” context, all of the above-mentioned robotics sensors can now also be used to trigger reactions of *logic blocks*. The latter contain logical relationships (like AND, OR, XOR,...), and even full Python scripts, who contain the core robotics functionality. The activated reactions can be movements of robots, or game/task functionality (like deciding to switch to another part of the task execution).

The current implementation of Blender runs the BGE in a single thread, hence the logic panel can provide only limited middleware connectivity. This

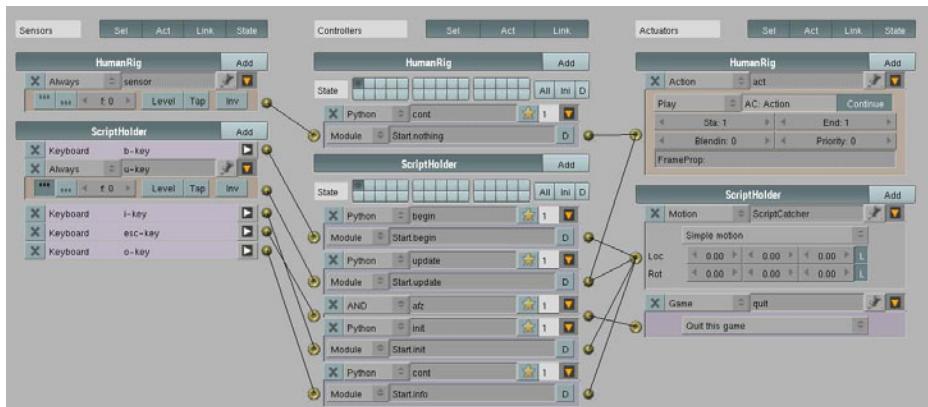


Fig. 5. Blender *logic bricks*, allowing to “reason” and make decisions from within Blender Game Engine’s native *sensor*, *controller* and/or *actuator* blocks

hinders applications that require continuous synchronisation. Currently, the following middleware interfaces (in addition to the universal TCP/UDP stream and datagram connection) have been tested, at K.U.Leuven and other labs:

- YARP [10] (Yet Another Robot Platform) allows the user to implement a large variety of transport protocols: TCP, UDP, Mcast, ShMem, Local,... Every connection has a specific type of carrier associated with it. By default YARP uses a TCP based connection. To allow various transport protocols to work with YARP, the specification has a carefully chosen data representation format, the NetType format. This NetType format chooses data types that can be represented easily in both binary and text format. YARP uses a name server to maintain a list of all YARP ports and how to connect to them. YARP also allows the programmer to implement his own carrier protocol. This means that if there is a python implementation available of the protocol, the user can implement it in the BGE by implementing the `yarp::os::impl::Carrier` interface.

Examples that use Blender visualisation via YARP can be found at [9].

- CORBA (Common Object Request Broker Architecture) [20] uses an interface definition language (IDL) that specifies how the interface to the network looks like. On top of this it has a mapping from a IDL file to the specific programming languages. An application has to initialise a Object Request Broker (ORB) that manages the network requests. The language mapping of the IDL format is highly dependent on the *object-oriented* architecture of the programming language. For the BGE Python interface this is not a problem, when the user program at the other side is using the C programming language however, this can be less straight forward. For the moment, we only use the *remote procedure call* (RPC) functionality of CORBA in the BGE, other services that CORBA provides (like a time service) could be a future extension of the middleware interface.

An example that uses Blender visualisation via CORBA is shown in [7], which illustrates our work on multiple target tracking and localisation/detection algorithms. In this example the BGE is connected over CORBA to an Orocros Real-Time Toolkit (RTT) [25] component containing the multiple target tracking and localisation algorithm. A video illustrating this can be found on: <http://people.mech.kuleuven.be/~u0062536/files/movies/simpar/mttl/>

- ROS [27] (Robot Operating System) uses a remote procedure call (XML-RPC [1]) approach, using HTTP as the transport and XML [26] as the encoding. ROS's master service provides the API for nodes to register themselves as a data publisher, a subscriber to a topic, and/or providing a service. The parameter server is part of the master and stores configuration parameters also based on XML-RPC.

By using a Python script it is possible to implement a ROS slave node in the BGE. Instead of using the normal callback handlers the BGE scheduling forces the programmer to use an alternative method and lets the programmer use a `wait_for_message` type of routine. This will get the latest message out

of the queue or waits for a message within a certain time-out range. This way the programmer can keep the Blender loop deterministic without interfering the communication with the ROS master.

An example that uses Blender via ROS is described in Section 3)

3 Example: Markerless Human Motion Capture

This section explains the current developments at K.U.Leuven to use the Blender Game Engine as, both, a visualisation tool and as a virtual lab (simulator), for the purpose of *markerless human motion capturing* as shown in Figure 6. That is, to estimate a human pose (i.e. joint angles in a skeleton model of the human) on the basis of video input from a real or simulated camera in a Blender environment. The estimated joint angles of the human skeleton model can then be used by a post-processing algorithm (e.g. for intent estimation, or gait classification, or gesture recognition, etc.).

The elaborate human model of Figure 7 is imported from the *MakeHuman* project [3] into the simulation part of the algorithm (external to Blender). A virtual lab is set up in Blender, in which a similar human model is imported and moved around. A particle filter estimator (using the *Bayesian Filter Library* (BFL) [12]) tracks the person's movements, by estimating it's pose and sends the results to the Blender visualisation.

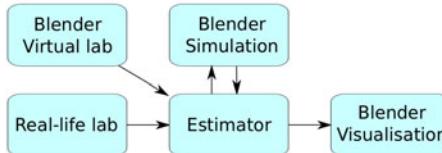


Fig. 6. Block diagram of the markerless human motion capture setup

All middleware communication is done using ROS. The measurements consist of camera images of the human. They can come from the real-world lab or the virtual lab. The images are converted into an OpenCV [17] structure, and then sent to the ROS estimator node over a time-stamped image ROS topic. For its operation the estimator has to calculate the expected measurements, i.e. the expected camera images, for each of its particles. To this end the estimator sends the particle state, which is a combination of position of the human centroid and the joint angles of the whole kinematic chain, using a ROS message to the BGE simulation. The BGE provides the 3D rendering, and returns either a black&white or a full color image. The binary image allows to evaluate poses based on a background-subtracted image to do simple pixel match counting. The full color image allows more complex color and feature matching algorithms.

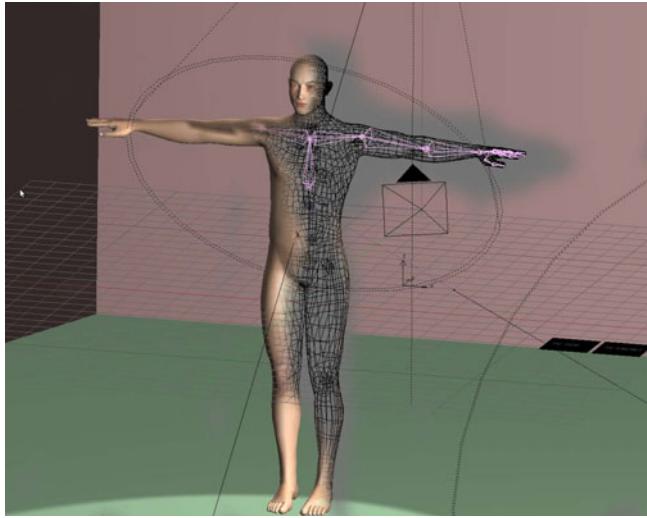


Fig. 7. Human model with internal kinematic chain

In our example, in order to evaluate each particle, the particle filter uses a pixel-based image correspondence calculation:

$$p(y|\tilde{y}) = \prod_{k=1}^K p(y_k|\tilde{y}_k), \quad (1)$$

with K the number of pixels, y_k and \tilde{y}_k the expected and the measured pixel values. In this calculation all pixels are assumed to be conditionally independent given the human pose. For each of the pixels, a 2D Gaussian pattern is used to calculate the local probability:

$$\begin{aligned} p(y_k|\tilde{y}) = & \Pi_1 p(y_k|\tilde{y}_k) + \Pi_2 p(y_k|\tilde{y}_{k-1}) + \\ & \Pi_3 p(y_k|\tilde{y}_{k+1}) + \Pi_4 p(y_k|\tilde{y}_{k-w}) + \Pi_5 p(y_k|\tilde{y}_{k+w}), \end{aligned} \quad (2)$$

with Π the Gaussian probability pattern and w the image width (see Figure 8).

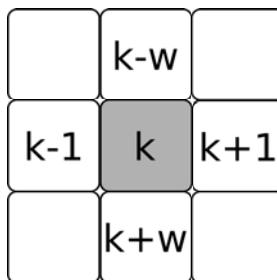


Fig. 8. Pixels used in Gaussian pattern

The code is available from: http://svn.mech.kuleuven.be/repos/orocos/trunk/kul-ros-pkg/human_motion_capt/; videos demonstrating the code are available on: <http://people.mech.kuleuven.be/~u0062536/files/movies/simpar/mhmc/>

4 Discussion and Future Work

A common need in all robotics labs is the use of Blender as a flexible simulation environment, capable of connecting to different robotics frameworks by a multitude of middleware solutions.

The use of Blender has some advantages: the first is that it is a Open Source project, providing the user with lot of freedom. Furthermore it has the physics engine Bullet integrated, it allows for photo-realistic rendering, has a structured code layout and has a large community supporting it.

However, there are currently some disadvantages of using Blender, some of which will be handled in future work: (i) Blender has a steep learning curve for beginners and a complicated graphical user interface (GUI) expecting some effort from the beginner to learn it; (ii) Blender has no external time synchronisation support; (iii) the Collada interface doesn't support the full 1.5 standard yet; and (iv) the Blender API documentation is often incomplete.

This paper showed how Blender can be used as a visualisation tool, how it can be extended to a simulation environment, and by what middlewares robotic frameworks can connect to it.

To help overcome the disadvantages ongoing shared work at the major "Blender for Robotics" labs is focused on the following: first writing a wrapper around the different types of middlewares, providing a generic interface to robotic frameworks on the one hand, and to Blender and its game engine on the other hand.

A second major missing feature when doing realistic simulation in a system built from various "legacy" components, is the need to let the activity of all components be triggered by one single global reference clock. In principle, Blender allows to run its internal graphical rendering loop in an event-triggered way, which is more than what most current robotics frameworks offer to their own components.

Another shared development effort is the implementation of the Collada 1.5 standard [2][16] to import/export scenes, robots, kinematic chains, etc. The Blender developers have already realised a partial integration of Collada 1.5 into Blender 2.5, but this implementation does not support armatures, which is an important features in robotics applications.

The last disadvantage of Blender we are trying to change is the GUI, we are adding a 'Robotics Panel' to Blender allowing users to place, configure and use robots, sensors and environments without having to know the extensive interface that Blender offers.

Acknowledgements

The Blender for Robotics project is mainly driven by the research labs of ONERA and LAAS in Toulouse, France. The authors acknowledge the Concerted Research Action project on “Global real-time optimal control of autonomous robots and mechatronic systems” funded by K.U.Leuven [GOA/2010/011].

References

1. XML-RPC, <http://www.xml-rpc.com/>
2. Barnes, M., Finch, E.L.: COLLADA—Digital Asset Schema Release 1.5.0 (2008), <http://www.collada.org> (last visited March 2010)
3. Bastioni, M., Flerackers, M.: Make human: Open source tool for making 3d characters (2007), <http://www.makehuman.org/> (last visited 2010)
4. Bruyninckx, H.: Blender for robotics, <http://wiki.blender.org/index.php/Robotics:Index> (accessed online June 20, 2010)
5. Bruyninckx, H.: Open Robot COnrol Software (2001), <http://www.orocos.org/> (last visited 2010)
6. Coumans, E.: Bullet physics library (2005), <http://bulletphysics.org/> (last visited March 2010)
7. De Laet, T.: Rigorously Bayesian Multitarget Tracking and Localization. PhD thesis, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium (May 2010)
8. De Schutter, J., De Laet, T., Rutgeerts, J., Decré, W., Smits, R., Aertbeliën, E., Claes, K., Bruyninckx, H.: Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty. *The International Journal of Robotics Research* 26(5), 433–455 (2007)
9. Echeverria, G., Lemaignan, S.: Blender for robotics video channel, <http://vimeo.com/groups/blenderandrobotics/videos> (last visited June 2010)
10. Fitzpatrick, P., Metta, G., Natale, L.: YARP: Yet Another Robot Platform (2002), <http://eris.liralab.it/yarp/> (last visited 2010)
11. Fleury, S., Herrb, M., Chatila, R.: GenoM: a tool for the specification and the implementation of operating modules in a distributed robot architecture. In: Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 1997, Grenoble, France, pp. 842–848 (1997)
12. Gadeyne, K., De Laet, T.: BFL: Bayesian Filtering Library (2001), <http://www.orocos.org/bfl> (last visited 2010)
13. Gerkey, B., Vaughan, R., Howard, A., Koenig, N.: The Player/Stage project (2007), <http://playerstage.sourceforge.net/>
14. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2004, Sendai, Japan, vol. 3, pp. 2149–2154 (2004)
15. Mallet, A., Fleury, S., Bruyninckx, H.: A specification of generic robotics software components and future evolutions of GenoM in the Orocōs context. In: Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2002, Lausanne, Switzerland, pp. 2292–2297 (2002)
16. NetAllied Systems GmbH. OpenCOLLADA (2009), <http://opencollada.org> (last visited March 2010)

17. OpenCV. Open computer vision, opencv (2001),
<http://www.intel.com/research/mrl/research/opencv/>
18. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T.B., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA Workshop on Open Source Software (2009)
19. Roosendaal, T.: Blender (2002), <http://www.blender.org> (last visited March 2010)
20. Schmidt, D.C., Kuhns, F.: An overview of the real-time CORBA specification. IEEE Computer special issue on Object-Oriented Real-time Distributed Computing 33(6), 56–63 (2000)
21. Smith, R.: ODE: Open Dynamics Library (2000), <http://www.ode.org> (last visited March 2010)
22. Smits, R.: KDL: Kinematics and Dynamics Library (2001),
<http://www.orocos.org/kdl> (last visited 2010)
23. Smits, R., Bruyninckx, H., De Schutter, J.: Software support for high-level specification, execution and estimation of event-driven, constraint-based multi-sensor robot tasks. In: Proceedings of the 2009 International Conference on Advanced Robotics, Munich, Germany (2009)
24. Smits, R., De Laet, T., Claes, K., Bruyninckx, H., De Schutter, J.: iTaSC: a tool for multi-sensor integration in robot manipulation. In: IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, MFI 2008, Seoul, South-Korea, pp. 426–433 (2008)
25. Soetens, P.: A Software Framework for Real-Time and Distributed Robot and Machine Control. PhD thesis (May 2006), <http://www.mech.kuleuven.be/dept/resources/docs/soetens.pdf>
26. W3C. XML: Extensible Markup Language (2008), <http://www.w3.org/XML/>
27. Garage, W.: Robot Operating System, ROS (2009), <http://www.ros.org> (accessed online Januari 26, 2010)

Simulating the C2SM ‘Fast’ Robot

R. Codd-Downey¹, M. Jenkin¹, M. Ansell², H.-K. Ng², and P. Jasiobedzki²

¹ Computer Science and Engineering, York University, Canada

² MDA Corporation, Brampton, Canada

Abstract. A critical problem in the deployment of commercial teleoperated robots is the development of effective training tools and methodologies. This paper describes our approach to the problem of providing such training to robot operators tasked to contaminated environments. The Vanguard MK2 and C2SMFast Sensor Simulator – a virtual reality based training system – provides an accurate simulation of the C2SM Crime Scene Modeler, an autonomous robotic system developed for the investigation of contaminated crime scenes. The training system provides a simulation of the underlying robotic platform and the C2SMFast sensor suite, and allows training on the system without physically deploying the robot or CBRNE contaminants. Here we describe the basic structure of the simulator and the software components that were used to construct it.

1 Introduction

Investigating crime scenes where narcotics or explosives may be found or responding to man-made or natural disasters puts police and first responders at significant safety risk as the nature of the event may introduce hazardous contaminants into the scene. The extreme version of this problem occurs when environments have been contaminated by Chemical, Biological, Radiological-Nuclear or Explosive (CBRNE) substances. Initial response to such events involves using unmanned robotic platforms to identify the hazardous substances and their distribution, and defining threat levels in order to plan the remediation operations before manned teams are deployed. Such robotic platforms are equipped with cameras and detectors selected for specific threats and are teleoperated in the scene providing live images and data to the operator station. Remote control of agile security robots using only images from onboard cameras requires practice and skill. It is even more challenging in confined spaces, contaminated or partially destroyed buildings, or when dealing with explosives using manipulators, tools or disruptors. The required skill can be gained only through practice.

The CBRN Crime Scene Modeler [6] is a mobile sensor system that has been developed to assist first responders in dealing with contaminated scenes. As a teleoperated device, the CBRNE Crime Scene Modeler provides an operator situated at a safe location the ability to build a 3D map of the contaminated environment and to locate within that environment concentrations of various events, chemical, radiological, etc., to aid in scene remediation and investigation. Models created by the CBRN can be integrated with standard database tools

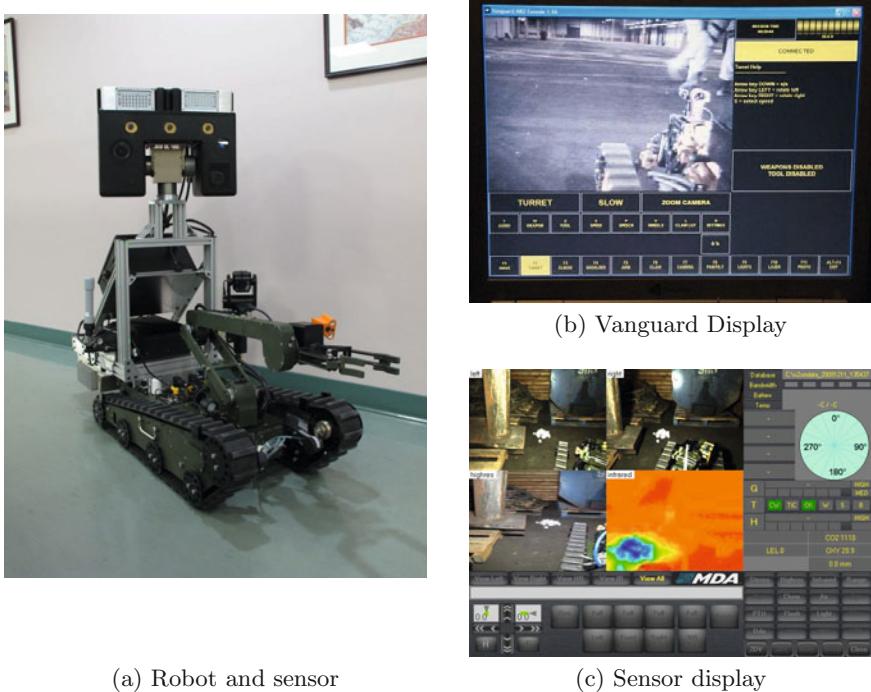


Fig. 1. C2SM mounted on a Vanguard robot platform. (a) Shows the C2SM mounted on top of the Vanguard robot platform. (b) and (c) show the two operator interface screens. (b) is the Vanguard robot platform interface display which provides mobility control of the vehicle. (c) shows the interface to the C2SM sensor subsystem. The operator uses two separate computers and keyboard to control the system.

representing other data obtained by investigators to build complex geo-located models of crime scenes [10][13].

Although field trials of the device have demonstrated its effectiveness, the technology places a significant cognitive load on the operator. Figure 1 illustrates some of the complexities faced by the operator of the system. The operator is presented with a range of different controls, onboard cameras and sensors. Some controls are tasked with controlling the movements of the robot and operate its onboard manipulator and cameras, while other onboard sensors are used to construct a 3D model of the scene and to obtain imagery outside of the visible spectrum. Other sensors measure radiation, air quality and chemical concentrations. Operating the robot and its sensors effectively becomes a technical art that involves moving the robot around a scene visually aided by onboard camera systems and using onboard sensors systematically to detect and localize threats.

Sensors available onboard the C2SM robot coupled with the robot’s ability to obtain accurate egomotion estimation through a “visual egomotion” sensor allow for the construction of large scale 3D environmental representations onto which local sensor measurements can be overlain. This is illustrated in Figure 2 which

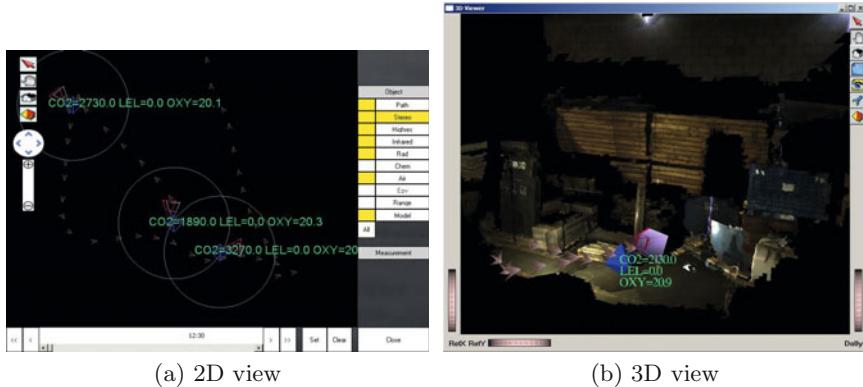


Fig. 2. Data obtained by the C2SM system. Sensors on board the robot obtain textured 3d models of the local environment which can be integrated into a large scale environmental model given the known egomotion of the device. Sensor data can be overlaid on these models providing a rich dataset upon which scene remediation and investigation can take place.

shows recovered sensor data overlaid on the ground plane (a) and the recovered 3D model with sensor measurement data (b). In order to perform tasks such as locating the source of a particular chemical contaminant, operators must be able to deal with these complex representations, current sensor data, and use this information to develop plans for further motion of the vehicle. This involves the operator developing a robust internal representation of the remote environment.

Given the complexities associated with conducting “real” field trials, there is considerable interest in the development of simulation systems that can aid in training. Such a system must provide an effective simulation of the multimodal nature of true environments. Visual and non-visual hazards in the environment must be simulated to an extent that an operator can feel as if they are looking at “real” sensor data. The physical interactions between the robot and the environment do not call for a high level of realism (such as explosions and chemical damage), but must nevertheless simulate complex indoor and outdoor environments. The system must also provide a realistic simulation of the control inputs and displays associated with the real hardware, providing a replica of the user interface of the simulated robot and sensor, and (critically) must be cost effective to deploy.

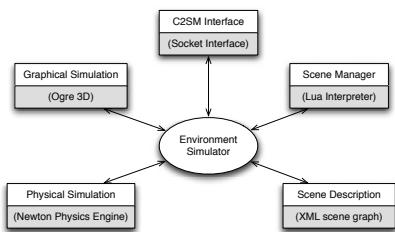
2 Background

There is a long history of the use of virtual reality and similar technologies in the development of training systems. Complex training system simulators can be traced back as far as the *Link Trainers* of the 1930’s (see [7] for a

description of early aviation trainers). As the technology has evolved, training systems, and especially virtual reality-based training systems, have evolved for a range of different tasks and environments including power substation operation [14], dentistry [2], welding [15], and of course aircraft operation [11]. The popularity of this approach has lead to the creation of a large number of such systems, and to the development of *Serious Games* which utilize commodity hardware and software systems to provide virtual training opportunities. Serious Games have evolved for a large number of different tasks including training hospitality professionals [1], dealing with feeding disorders in infants [12], and the training of health care providers [9]. The rationale for the development of these and similar training systems is that effective simulations can be constructed that enable trainees to participate in complex and controlled scenarios while reducing cost, time, and risk to participants. These requirements are well met when considering the problem of training operators of robotic devices that must deal with chemical, biological, radiological, nuclear and explosive materials. Such environments occur in many countries, albeit relatively infrequently, and there is a pressing need to train first responders for such environments.

3 System Design

The simulation system must provide an accurate physical simulation of the robot and its environment as well as its sensing capabilities. Of particular importance is the accurate simulation of the vehicle's mobility and its interaction with complex terrain. This necessitates the development of a virtual environment in which the



(a) Basic system structure



(b) Operator view

Fig. 3. Basic System Structure. The environment simulator integrates a physical and visual environment simulation along with simulations of the various non-visual sensors available onboard the robot. The operator is presented with an operation console that is identical to the “real” robot interface shown in Figure II(b). Simulated sensor data is provided via a standard TCP/IP interface to the sensor user interface shown in Figure II(c).

robot and sensors can be properly simulated. This requires realistic simulation of, at least;

- visual features in the environment,
- physical interaction between the robot and objects in the environment,
- chemical sources within the environment,
- air quality conditions within the environment,
- radiological sources within the environment, and,
- infrared sources within the environment.

The basic structure of the system is shown in Figure 3. A central application employs a graphical rendering and physics engine to manage the simulation. The sensors are controlled and monitored via a TCP/IP socket interface. The environment is described by an XML file and managed by a Lua [4] script that can manipulate the state of objects and needed sensor information within the environment.

The C2SM simulator leverages a number of open source or easily licensed software packages:

- The Ogre3D [8] rendering system is used to provide a basic scene graph structure and rendering.
- The Newton Physics Engine [5] is used to provide basic physics simulation.
- The Blender [3] modelling system for the development of visual models.
- A Lua [4] interpreter is used as a scenario manager and to model non-visual events.

These software packages were chosen because of their ability to run on a number of different hardware/software platforms, their cost effectiveness and their ability to provide an effective simulation of the robot and the environment.

3.1 Physical and Visual Scene Simulation

The C2SM simulator leverages a number of publicly available tools to simplify the construction and rendering of simulated environments. The simulator uses Ogre's scene graph format to structure objects within the environment. Scenes are represented in an extension of Ogre's `dotScene` format. This extension provides specification of the object's state in terms of the physics engine. Figure 5 illustrates that portion of the `dotScene` file that provides the link between the robot, the scene and the scenario manager. The robot is placed at the specified `position` and `orientation` within the simulated environment. Once the Ogre3D and Newton environments have been initialized the script file defined by `scenario` is loaded and executed. This establishes the initial state of the simulation and also defines any callbacks needed for non-visual sensors.

The visual simulation requires simulation of geometry, lighting and surface models. Figure 6 shows how lights are added to the scene to provide dynamic light to the environment. Ogre provides a number of different kinds of lights including spot, direction and point light sources. This enables a large number of different lighting effects.

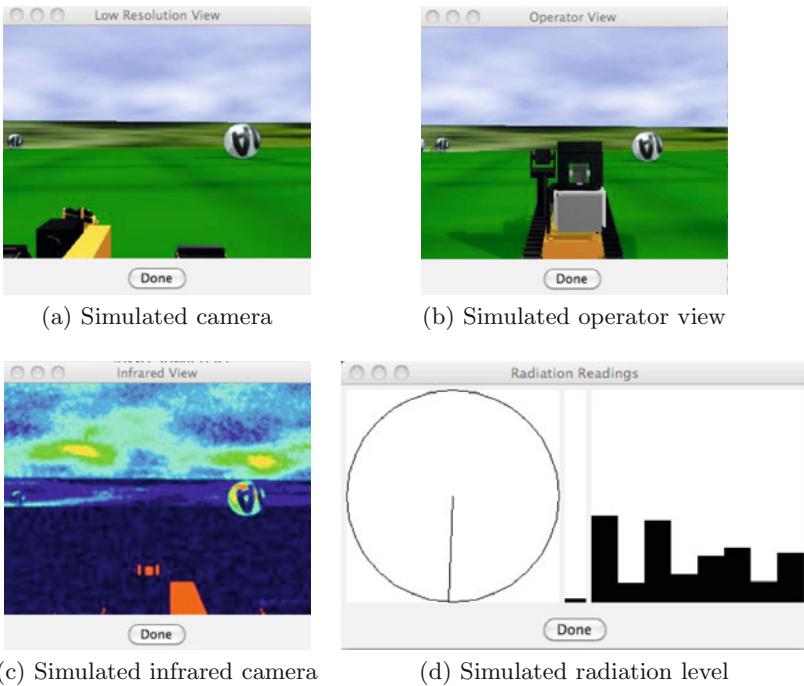


Fig. 4. Simulated sensor data. Each of the simulated sensors on the C2SM sensor provide data streams that are accessible by the C2SM user interface. These data streams are also available to other external tools. Shown here are the sensor streams from four of the sensors associated with the robot. (a) provides a view from one of the cameras mounted on the C2SM. (b) provides the view obtained by an operator standing behind the robot. (c) provides the view from the infrared camera mounted on the robot. (d) shows data from the radiation sensor integrated within the C2SM sensor.

```
<environment>
  <scenario name="scenario1.lua"/>
  <robot position="5 0.1 5" orientation="0 0 0 1"/>
</environment>
```

Fig. 5. A snippet of a scene environment description with robot position, orientation and Lua scene manager script

Figure 7 illustrates how a single object (here a sphere) is coded in the simulator. Each object has a name (here `ellipsoid`), that provides a unique label for the object for later manipulation by scene scripts. The visual appearance of an object is defined in terms of a Blender `mesh` object which may be modified by a set of transformations. In addition to its visible appearance, objects may also have a physical existence in terms of the physics engine. This includes mass and physical extent. Here the object has a mass of 1 and a physical extent that matches its visual appearance. This is a movable object within the scene. Static physical objects are signalled by omitting the `mass` property.

```

<node name="Light1">
    <light type="spot">
        <colourDiffuse r="0.3" g="0.3" b="0.3"/>
        <colourSpecular r="0.0" g="1.0" b="0.0"/>
        <position x="50.0" y="50.0" z="50.0"/>
        <direction x="-1.0" y="-1.0" z="-1.0"/>
        <range inner="5" outer="15" falloff="1.0"/>
    </light>
</node>

```

Fig. 6. Definition of a spot light from a scene file. Extremely complex lighting models are possible.

```

<node name="ellipsoid">
    <position x="-10.0" y="25.0" z="0.0"/>
    <scale x="0.1" y="0.1" z="0.1"/>
    <entity meshFile="ellipsoid.mesh"/>
    <userData>
        <property name="Primitive" data="ellipsoid"/>
        <property name="Mass" data="1.0"/>
        <property name="Size" data="0.1 0.1 0.1"/>
    </userData>
</node>

```

Fig. 7. A snippet of a scene description. Ogre's *dotScene* XML scene description is augmented with state information for the physics engine.

3.2 Simulating Other Visual Sensors

Simulating the infrared camera on board the robot involves two key features of the rendering system. The first of which is Ogre3D's shader framework. This permits fine grained control over the rendering pipeline, which we exploit to transforms light from the scene into "heat" and the absence of light being "cold". Ogre's Material schemes allow for top-level switching of rendering techniques which is used to switch from the traditional rendering pipeline to our homegrown shaders that produce the desired effect. Since our shaders convert a grayscale light spectrum to colours that represent the infrared spectrum of light, we wish to give some objects a heat value irrespective of their "true" colours in the visual environment. This is accomplished by manipulating the material attributes of the object. A material scheme named thermal is used to give the object a high light value when rendering the thermal scene. This value is then converted to a heat signature by the shader. Figure 8 illustrates how this thermal information is integrated into the rendering pipeline for an object. Different levels of ambient noise can be introduced within this process. Figure 4(c) shows the simulated infrared camera in action. As the robot's arm is perpendicular to the lighting direction it shows as "hot". The infrared camera is further illustrated in the sample scenario below.

The range camera is implemented in a similar fashion to the infrared camera. The range camera uses a set of shaders that calculates each pixel's value in the

```

material spherehot {
    technique {
        pass {
            diffuse 1.000000 1.000000 1.000000
            specular 0.500000 0.500000 0.500000 12.500000
            texture_unit {
                texture mda.jpg
            }
        }
    }
    technique {
        scheme thermal
        pass {
            ambient 1.0 1.0 1.0 1.0
        }
    }
}

```

Fig. 8. The infrared camera pipeline. An object coloured with the `spherehot` material will appear as a normally lit sphere with the texture defined in `mda.jpg` applied to it. In the infrared camera, however, the `thermal` rendering scheme will be applied which identifies it as hot. It will then appear coloured red in the thermal camera.

z-buffer and converts it to a grayscale colour value. Unlike the infrared camera no object needs to be marked as special, i.e., there are no heat sources. Rather objects obtain their “colour values” based on how far they are from the camera. Instead of assigning special material schemes to objects, at render time all objects are assigned a common material that will colour them based on their depth.

The customizable pipeline approach for scene rendering can be used to simulate a variety of different sensors. One can imagine creating a night vision camera that could be simulated by coupling a shader with a material and applying that to all objects at render time. Another possible sensor could be a camera that only picks up ultraviolet light. This could be implemented by adding a UV material scheme and light values to each object that emits light in the ultraviolet spectrum and coupling it with a shader much like the infrared shader described above.

3.3 Simulating Non-visual Sensors

In addition to providing a realistic visual and physical environment, the C2SM simulator must also simulate a number of non-visual events including radiation and chemical distributions. Whereas the visual and physical simulations can leverage existing software systems that were designed to simulate the visual and physical world, this is not the case for other classes of events. In order to provide the developers of training scenarios the greatest flexibility in terms of defining the interaction between the non-visual event being simulated and the environment, the C2SM simulator uses a procedural definition of these events written as a script in Lua.

```

radiationData = function()
  t = {getObjectState("robotvanguard_body")}
  best = 1;
  x = {getObjectState(targets[1])}
  dbest = dist(x[1], x[2], x[3], t[1], t[2], t[3])
  for i, v in ipairs(targets) do
    x = {getObjectState(v)}
    d = dist(x[1], x[2], x[3], t[1], t[2], t[3])
    if d < dbest then
      dbest = d
      best = i
    end
  end
  x = {getObjectState(targets[best])}
  dir = norml({x[1] - t[1], x[2] - t[2], x[3] - t[3]})
  head = quatRotate(t[4], t[5], t[6], t[7], 0, 0, -1)
  dotp = head[1] * dir[1] + head[2] * dir[2] + head[3] * dir[3]
  angle = math.acos(dotp)
  cp = crossProduct(dir, head)
  if cp[2] > 0 then
    angle = -angle
  end
  spectrum = {0,0,0,0,0,0,0,0}

  magn = 1000 / (dbest * dbest + 1)
  return magn, angle, #spectrum, spectrum
end

```

Fig. 9. Script snippet illustrating how non-visual sensors are defined procedurally. This snippet defines the radiation landscape. The location of a number of sources are known and the closest source is identified. Given this source a simulated magnitude, direction and signal spectrum are generated. The sensor on board the robot adhere to this type of radiological measuring.

Figure 9 shows a snippet of how a simple radiation source model can be scripted within the simulator. (Of course, more sophisticated radiation landscapes can be defined using the same interface.) A Lua script file is associated with a simulation that defines the function `radiationData` that must return the radiation signal in a specific format (here including magnitude and direction to the source and along with information about the signal spectrum). The state of the simulated world is exposed to the scripting language through the functions `getObjectState` which returns a vector that describes the current state of that object in the simulation (its position, orientation and its visibility status). A similar function `setObjectState` allows a script to manipulate the state of an object.

4 Sample Scenario

Figure 10 illustrates a sample scenario with the C2SM Fast simulator. The operator's goal is to identify the 'hot' infrared source in the environment. The

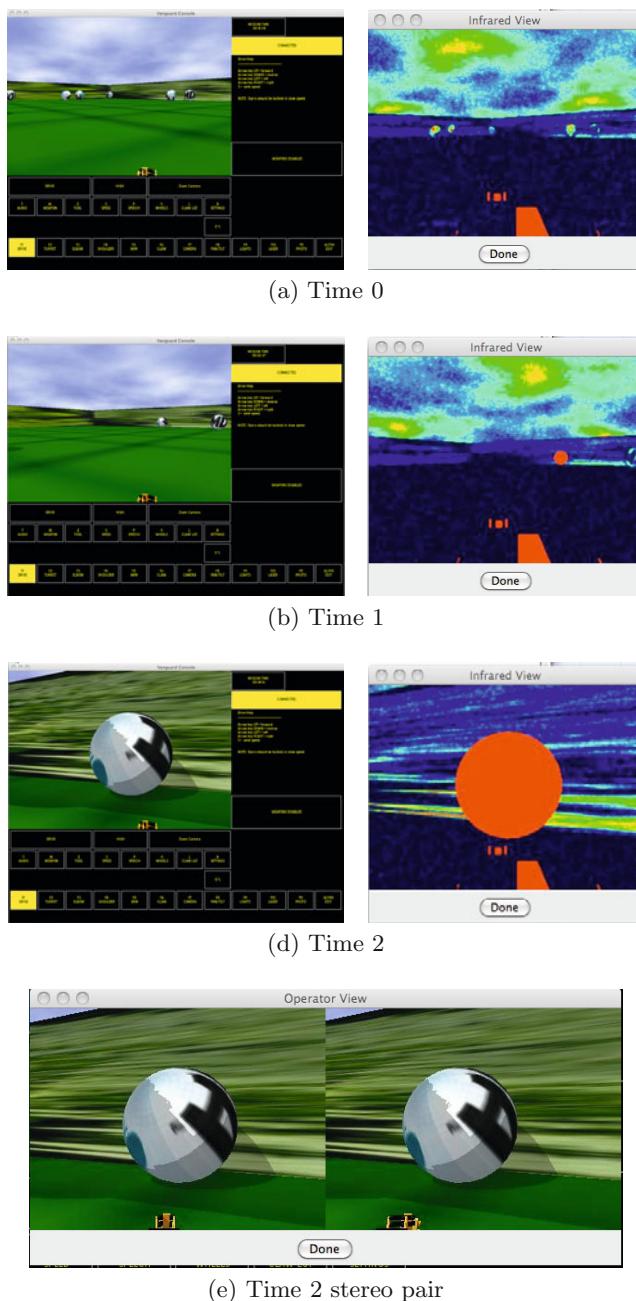


Fig. 10. Sample scenario seeking a hot source (here a large textured sphere in the environment). The operator’s goal is to find and localize the ‘hot’ sphere in an environment of similar ‘cold’ spheres.

environment is populated with a large number of textured spheres. These spheres all have an identical visual appearance, as seen in the operator's console. Finding the 'hot' sphere involves using the C2SM infrared camera to distinguish between the various spheres in the environment. After scanning the environment using the Vanguard mobile base, the operator identifies the 'hot' sphere and then localizes it with the stereo cameras mounted on the C2SM sensor. From this stereo pair the operator can build a depth map of the hot object.

5 Discussion and Future Work

For sophisticated robotic systems – even teleoperated ones – to find widespread deployment it is essential that effective training technologies be developed. With advances in video game engines and physical modelling systems it is now relatively straightforward to build realistic simulation systems using open source or easily licensed software components. When coupled with appropriate embedded scripting languages and tools for content generation this provides a powerful set of inexpensive tools for the development of effective robotic simulation systems.

Ongoing work involves the development of realistic training scenarios and integration of live data from the robot within 3D models that can be loaded into the simulator. This will allow the simulation system to be used for task rehearsal when planning missions involving contaminated crime scenes.

Acknowledgments

This project is supported by the Defence Research and Development, Centre of Security Science, CBRNE Research and Technology Initiative and by the National Science and Engineering Council of Canada.

References

1. Cantoni, L., Kalbaska, N.: The waiter game: structure and development of an hospitality training game. In: 2nd Int. Conf. on Games and Virtual Worlds for Serious Applications (VS-GAMES), pp. 83–86 (2010)
2. Hashimotot, N., Kato, H., Matsui, K.: Evaluation of training effect of tooth scaling simulator by measurement of operation force. In: Proc. IEEE Int. Conf. on Virtual Reality, pp. 273–274 (2010)
3. Hess, R.: The Essential Blender: Guide to 3D Creation with the Open Source Suite Blender. No Starch Press (2007)
4. Ierusalimschy, R.: Programming in Lua, 2nd edn. (2006), Lua.org
5. Jarez, J., Suero, A.: Newton game dynamics, <http://newtondynamics.com>
6. Jasiobedzki, P., Ng, N.K., Bondy, M., McDiarmid, C.H.: C2SM: a mobile system for detecting and 3d mapping of chemical, radiological and nuclear contamination. In: Carapezza, E.M. (ed.) Proc. SPIE Conf. on Sensors, and Command, Control Communications and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense VIII, vol. 7305, pp. 730509–730509–10 (2009)

7. Jaspers, H.: Resotring and operating historical aviation trainers. In: Flight simulation 1929–2029: a centennial perspective, London, UK (2004)
8. Junker, G.: Pro OGRE 3D Programming. Apress (2006)
9. Khattak, S., Sabri, H., Hogan, M., Kapralos, B., Finney, K., Dabrowski, A.: A serious game for community health nursing education and training. *Journal of Health Professions Education* 1 (2009)
10. Kwietniewski, M., Wilson, S., Topol, A., Gill, S., Gryz, J., Jenkin, M., Jasiobedzki, P., Ng, H.K.: A multimedia event database for 3d crime scene representation and analysis. In: Proc. 24th Int. Conf. on Data Engineering, Cancun, Mexico (2008)
11. Martynowicz, Z.E.: Digital avionics simulation for trainers. In: Proc. 1992 IIE/AIAA Digital Avionics Systems Conference, pp. 434–439 (1992)
12. Petrasova, A., Dzanner, S., Chalmers, A., Farrer, J.V., Wolke, D.: The playability evaluation of virtual baby feeding application. In: 2nd Int. Conf. on Games and Virtual Worlds for Serious Applications (VS-GAMES), pp. 95–100 (2010)
13. Topol, A., Jenkin, M., Gryz, J., Wilson, S., Kwietniewski, M., Jasiobedzki, P., Ng, H.K., Bondy, M.: Generating semantic information from 3D scans of crime scenes. In: Proc. Computer and Robot Vision (CRV), Windsor, ON (2009)
14. Wang, W., Li, G.: Virtual reality in the substation training simulator. In: 14th Int. Conf. on Computer Supported Work in Design (CISWD), pp. 438–443 (2010)
15. White, S., Prachyabrued, M., Baghi, D., Aglawe, A., Reiners, D., Borst, C., Chambers, T.: Virtual welder trainer. In: Proc. IEEE Int. Conf. on Virtual Reality, p. 303 (2009)

Extending Open Dynamics Engine for Robotics Simulation

Evan Drumwright¹, John Hsu², Nathan Koenig², and Dylan Shell⁴

¹ George Washington University, Washington DC, USA
`drum@gwu.edu`

² Willow Garage, Menlo Park, CA, USA
`johnhsu@willowgarage.com, nkoenig@willowgarage.com`

³ Texas A&M University, College Station, TX, USA
`dshell@cs.tamu.edu`

Abstract. Open Dynamics Engine (ODE) is the most popular rigid-body dynamics implementation for robotics simulation applications. While using it to simulate common robotic scenarios like mobile robot locomotion and simple grasping, we have identified the following shortcomings each of which adversely affect robot simulation: lack of computational efficiency, poor support for practical joint-dampening, inadequate solver robustness, and friction approximation via linearization. In this paper we describe extensions to ODE that address each of these problems. Because some of these objectives lie in opposition to others—*e.g.*, speed versus verisimilitude—we have carried out experiments in order to identify the trade-offs involved in selecting from our extensions. Results from both elementary physics and robotic task-based scenarios show that speed improvements can be gained along with useful joint-dampening. If one is willing to accept an execution time cost, we are able to represent the full-friction cone, while simultaneously guaranteeing a solution from our numerical solver.

Keywords: Physical Simulation, Dynamics, Locomotion and Grasping.

1 Introduction

Simulation serves as an important tool for the development, programming, and testing of robots and their control software. We anticipate that in the coming years the significance of simulation will grow as it becomes an essential part of robotic design, debugging and evaluation processes. In retrospect, the inadequacy of current simulation technologies will likely be identified as one of early impediments to the field’s progress and an initial failure to reach high standards of repeatability and scientific rigor. Indeed, current algorithms for physical simulation have several shortcomings which limit their use for problems of practical interest to the roboticist. In this paper we will describe and go some way toward addressing four such shortcomings in a popular rigid-body dynamics implementation for robotics simulation applications.

A roboticist needs frustratingly little imagination in order to encounter serious flaws in current physical simulation. A scenario as simple as driving a mobile robot on a plane can be challenging if the user requires that frictional effects between tire and ground be realistically modeled. Several approximations are made in such a scenario, each of which affects the quality of the resultant output. Computational intractability, inefficient implementation, or both, mean that a user desiring real-time (or faster than real-time) response must sacrifice the level of detail modeled: but coarse polyhedral approximations and large integrator step-sizes influence the fidelity of the output. Perhaps worst of all, these influences operate in ways that remain poorly understood.

Although specialized algorithms, libraries and solvers exist to address these and other particular problems, in practice few have been incorporated in widely used robot simulators. Since most robot simulators employ external libraries and third-party code to handle simulation of the physical dynamics and contact mechanics, improvements usually occur when those code-bases are improved. Thus far, few observations of problems arising specifically in simulation of robots have resulted in extensions and modifications of these general purpose (and sometimes game-oriented) physical dynamics libraries. Modification of one such library is the approach we have employed in this paper.

Using the current version of the Open Dynamics Engine (ODE) [27] via Gazebo [18] to simulate mobile robot locomotion and simple grasping by a manipulator, we encountered and identified the following four distinct issues:

1. Currently, significant computational resources are required by ODE. Simulated execution is often significantly slower than real-time—and can be slower by orders of magnitude. The resultant long running times ultimately undermine the simulator’s utility: lack of interaction makes for an inconvenient environment for iterative processes, running time limits the number of experiments that can be carried out, and physical hardware instantiations are required sooner in the design process than would be ideal.
2. The joint-dampening approach currently employed does not adequately model joints found in real robots and their physical mechanisms. Realistic simulation would require the parameter settings to be near the upper limits of the viscous friction joint dampening constant, which is not easily determined as a constant before run-time. This limits both the types of robots that can be simulated, and the sorts of environmental interactions (and hence tasks) that can be modeled.
3. The true friction cone is currently approximated by a linearized version which introduces orientation specific artifacts. These artifacts limit the set of physical phenomena that can be reproduced at a level adequate for robot applications. The default box-friction model results in highly non-isotropic frictional effects that can drastically alter the motions of mobile robots, grasped objects, vibrating items, *etc.*
4. Currently ODE employs linear complementarity problem (LCP) [10] solver to ensure forces satisfy non-interpenetration and joint-constraints. The solver does not always find a solution, which causes the constraints to be violated

within the simulation run. This produces in nonphysical behavior and a state from which further recovery is ill-defined.

We have implemented additions to ODE in order address these issues: extensions enabling multi-threaded execution of *quickstep*, iterative updates for joint damping, and a convex-optimization solver. We also report on experiments that we have conducted in order to validate the correctness, assess the effectiveness of the methods, and determine the trade-offs involved.

2 Related Work

While no single robot simulator has yet emerged as a standard, many existing simulators make use of ODE as a library. By addressing shortcomings in this underlying piece of software, we believe one is most effective at improving the state of robotic simulation.

2.1 ODE for Simulating Robots

Open Dynamics Engine (ODE) is the most popular rigid-body dynamics implementation for robotics simulation. Two of the most widely used simulators are the open source Gazebo [18] and the commercial Webots [23], together they are used in hundreds of institutions and laboratories. Both make direct use of ODE for their dynamics simulation. Both are general robotics simulators: they model a range of robots. OpenSimulator [17] is similar in this regard, as is the book by Demur [12] devoted to the particularities of using ODE for simulation of a range of robots.

Several simulators intended specifically for simulating RoboCup soccer make use of ODE, including SPARK [26], SimRobot [19], UchilSim [33], and ÜberSim [7]. Virtual-RE [8] is an ODE-based simulator focused on soccer playing humanoid robots. Humanoid and bipedal robot simulators that employ ODE across a range of other tasks include the iCub Simulator [30], that of Lee and Oh [20], Sugiura and Takahashi [29], Moores and MacDonald [24], and Lima et al. [21]. ODE has also been used to model particular robots for synthesis applications *e.g.*, Wolff and Wahde [31] model physics via ODE to enable genetic programming to evolve controllers for biped locomotion. Finally, several groups have developed pluggable, or configurable simulations that cite ODE as a suitable module, most notably, MuRoSimF [15] and VISUM [14].

2.2 Other Methods for Robotic Simulation

Also worth mentioning, Bullet [11] has an implementation of the same PGS algorithm in ODE. Preliminary test runs for simple constrained dynamics through Gazebo showed similar constraint solving performances.

Comparisons for various simulators including popular physics engines such as Bullet [11] and PhysX [9] can be found in previous benchmarking efforts [5]. Unfortunately for robotics research and engineering, many physics engines have been developed and benchmarked with animation and gaming applications in mind [32], imperceptible error tolerances may not always be the goal for many robotic simulations.

3 Approach

3.1 Current ODE Details

Smith [27] provides detail on the current version of ODE; this is an overview.

Governing Equations. In general, rigid body dynamic simulators attempt to solve the constrained Newton-Euler equation

$$\mathbf{M} \mathbf{a} = \mathbf{f}_{\text{ext}} + \mathbf{f}_{\text{constraint}} + \mathbf{f}_{\text{damp}} \quad (1)$$

for systems of dynamic rigid bodies, where \mathbf{f}_{ext} denotes the external applied forces, $\mathbf{f}_{\text{constraint}}$ denotes the constraint forces and \mathbf{f}_{damp} denotes the viscous joint damping forces. ODE takes the approach of posing the governing equation (1) as an LCP problem in the maximal coordinate system, solving for constraint satisfying impulses for the rigid bodies connected by joints in simulation.

3.2 Speeding Up ODE: Quickstep Implementation

ODE Quickstep uses the projected Gauss-Seidel (PGS) method with Successive Overrelaxation (SOR) [3, 2].

Given that rigid body constraints are described by a constraint Jacobian \mathbf{J} such that

$$\mathbf{J} \mathbf{v} = \mathbf{c} \quad (2)$$

The following mixed complementarity formulation [4] results:

$$[\mathbf{JM}^{-1}\mathbf{J}^T] \lambda = \frac{\mathbf{c}}{\Delta t} - \mathbf{J} \left[\frac{\mathbf{v}^n}{\Delta t} + \mathbf{M}^{-1} (\mathbf{f}_{\text{ext}} + \mathbf{f}_{\text{damp}}) \right], \quad (3)$$

where $\lambda \geq 0$ and $\mathbf{c} \geq 0$ for unilateral contacts.

During an update step of equation (3), the Jacobians \mathbf{J} are treated explicitly (held constant).

Equation (3) can be written as a simple system of linear equations

$$\mathbf{A} \lambda = \mathbf{b} \quad (4)$$

where

$$\mathbf{A} = [\mathbf{JM}^{-1}\mathbf{J}^T], \quad \mathbf{b} = \frac{\mathbf{c}}{\Delta t} - \mathbf{J} \left[\frac{\mathbf{v}^n}{\Delta t} + \mathbf{M}^{-1} (\mathbf{f}_{\text{ext}} + \mathbf{f}_{\text{damp}}) \right]. \quad (5)$$

Solve for the unknowns λ in (3) using Gauss-Seidel written in delta form,

$$\delta_i = \frac{b_i}{A_{ii}} - \sum_{j=1}^{N_{\text{constraints}}} \frac{A_{ij}}{A_{ii}} \lambda_j, \text{ where } \lambda_i^{n+1} = \delta_i^n + \lambda_i^n \text{ for } i = 1, \dots, N_{\text{constraints}}. \quad (6)$$

With SOR, a relaxation constant ω is introduced. The solution update becomes

$$\hat{\lambda}^{n+1} = (1 - \omega) \lambda^n + \omega \lambda^{n+1}, \quad (7)$$

or

$$\hat{\lambda}^{n+1} = \lambda^n + \omega(\delta^n). \quad (8)$$

Thus, equation (6) is scaled by ω

$$\delta_i = \omega \left(\frac{b_i}{A_{ii}} - \sum_{j=1}^{N_{\text{constraints}}} \frac{A_{ij}}{A_{ii}} \lambda_j \right). \quad (9)$$

Additional implementation notes. For efficiency, formulate the desired solution in the form of acceleration, denoted by

$$\mathbf{a}_c = \mathbf{M}^{-1} \mathbf{f}_c = \mathbf{M}^{-1} \mathbf{J}^T \lambda \quad (10)$$

then λ updates in equation (9) becomes

$$\delta_i = \omega \left(\frac{b_i}{A_{ii}} - \frac{\sum_{k=1}^{N_{\text{DOF}}} J_{ik} a_{ck}}{A_{ii}} \right) \text{ for } i = 1, \dots, N_{\text{constraints}}. \quad (11)$$

At every iteration, for each i update above, constraint accelerations a_{ck} are updated in the following manner:

$$a_{ck}^{n+1} = a_{ck}^n + G_{ki} \delta_i^n \text{ for } k = 1, \dots, N_{\text{DOF}} \quad (12)$$

and where \mathbf{G} is defined as

$$\mathbf{G} \equiv \mathbf{M}^{-1} \mathbf{J}^T. \quad (13)$$

Solution Projection. Lastly, to enforce inequality constraints, each intermediate solution λ_i^{n+1} is projected into its corresponding solution space depending on the type of constraint specified. For contact constraints, the constraint force corresponding to the normal direction of the contact,

$$f_{\text{constraint}_i} = \mathbf{J}_i^T \lambda_i, \quad (14)$$

is required to be push the bodies apart (no stiction), *i.e.*,

$$\lambda_i \geq 0, \quad (15)$$

where \mathbf{J}_i^T denotes the transpose of the i -th column of \mathbf{J} . And the tangential friction coefficients are truncated into its respective solution space depending on the friction coefficient,

$$f_{\text{fric}} \leq |\mu| f_{\text{normal}} \quad (16)$$

Parallel Jacobi Gauss-Seidel Iterations. Many past attempts at parallelizing iterative LCP solvers exist. Some examples are [22] [13] [16]. The approach taken in this paper simply performs update of individual row of the Gauss-Seidel iterations in parallel using latest available information. By updating multiple rows simultaneously, we have effectively turned subsets of the Gauss-Seidel algorithm into Jacobi iterations. Given that Jacobi's method has a tighter stability bound

than Gauss-Seidel iterations, runtime parameters must be tuned to prevent numerical instabilities. If the constraint matrix is well conditioned and diagonally dominant, the solution will converge. Otherwise, the user has to decrease the relaxation variable ω or increase ODE's *CFM* parameter to stabilize the system.

We observe that non-iterative approaches exist in the literature and that these can have speed advantages. Methods like those of Baraff [4] can solve acyclic arrangements in linear time. We note, however, that cyclic constraints occur frequently in robotics research, and thus we believe iterative methods are required in the general case.

4 Viscous Joint Damping

In the current *quickstep* implementation, ODE does not support simulation of viscous joint damping. The user is left with the option to apply joint damping explicitly outside of ODE, where the joint velocity from previous time step is used to compute the viscous joint damping force within the current time step,

$$\mathbf{f}_{\text{damp}}^{n+1} = k \mathbf{v}_{\text{joint}}^n. \quad (17)$$

Physical joints on real robots frequently have high mechanical reduction ratio leading to high viscous damping coefficients and low efficiency. Simulating large viscous damping forces leads to numerical stiffness for the physical system, requiring very small time step sizes and poor simulation performance. On the other hand, solving for damping forces implicitly is expensive. As a compromise, the forces can be updated within each PGS iteration with minimal loss in computational efficiency and higher tolerated viscous damping coefficient for a given fixed time step size.

4.1 Convex Optimization

ODE emphasizes speed and stability over physical accuracy Smith [27]. We describe here a method we term the convex optimization model, which models the complete friction cone rather than a linearized version of it.

The convex optimization model is formed and solved in two phases. Strictly speaking, this decomposition is unnecessary, though fast solution methods are admitted in this manner. The first phase consists of solving a *linear complementarity problem* (LCP). Much of the notation and problem setup will be similar to that presented in Anitescu and Potra [3]. Given the contact normal twist matrix \mathbf{N} , bilateral constraint Jacobian \mathbf{J} , lower and upper joint limit Jacobians \mathbf{J}_l and \mathbf{J}_h , vector of external forces \mathbf{k} and step size h , we solve the LCP below:

$$\begin{bmatrix} \mathbf{M} & \mathbf{J}^\top & -\mathbf{N}^\top & -\mathbf{J}_l^\top & -\mathbf{J}_h^\top \\ \mathbf{J} & 0 & 0 & 0 & 0 \\ \mathbf{N} & 0 & 0 & 0 & 0 \\ \mathbf{J}_l & 0 & 0 & 0 & 0 \\ \mathbf{J}_h & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}^{n+1} \\ \mathbf{c}_v \\ \mathbf{c}_n \\ \mathbf{c}_l \\ \mathbf{c}_h \end{bmatrix} + \begin{bmatrix} -\mathbf{M}\mathbf{v}^{n+1} - h\mathbf{k} \\ -\mathbf{j}_c \\ -\mathbf{n}_c \\ -\mathbf{j}_l \\ -\mathbf{j}_h \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{w} \\ \mathbf{x} \\ \mathbf{y} \end{bmatrix} \quad (18)$$

for the variables \mathbf{v}^{n+1} (the new vector of generalized velocities after all impulses are applied), \mathbf{c}_v (the bilateral constraint impulses), \mathbf{c}_n (the magnitudes of impulses in the normal directions), \mathbf{c}_l (the magnitudes of impulses applied at the lower joint limits), \mathbf{c}_h (the magnitudes of impulses applied at the upper joint limits) and the (unused) slack variables \mathbf{w} , \mathbf{y} , and \mathbf{z} . Note that \mathbf{c}_n and \mathbf{w} are complementary, as are \mathbf{c}_l and \mathbf{x} and \mathbf{c}_h and \mathbf{y} ; in notation, $\mathbf{w}^\top \mathbf{c}_n = \mathbf{x}^\top \mathbf{c}_l = \mathbf{y}^\top \mathbf{c}_h = 0$.

The vectors \mathbf{j}_c , \mathbf{n}_c , \mathbf{j}_l , and \mathbf{j}_h represent the amount of joint constraint violation, contact interpenetration, and joint limit violation already present in the simulation. In an ideal simulation environment, the vectors \mathbf{j}_c , \mathbf{n}_c , \mathbf{j}_l and \mathbf{j}_h would be zero vectors. In reality—due to ODE’s fixed step sizes—joint constraints will be violated, contact interpenetration will occur, and joint limits will be violated. These vectors are provided automatically by ODE’s joint constraint functions.

We now describe how to efficiently solve the LCP above. We can represent the matrix in (18) by blocks to yield $\begin{bmatrix} \mathbf{G} & -\mathbf{A} \\ \mathbf{A}^\top & \mathbf{0} \end{bmatrix}$, where $\mathbf{G} = \begin{bmatrix} \mathbf{M} \mathbf{J}^\top \\ \mathbf{J} \mathbf{0} \end{bmatrix}$ and $\mathbf{A} = \begin{bmatrix} -\mathbf{N}^\top & -\mathbf{J}_l^\top & -\mathbf{J}_h^\top \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$.

We can then solve the LCP (\mathbf{q}, \mathbf{S}) (using the notation of Cottle et al. [10]), where $\mathbf{S} = \mathbf{A}^\top \mathbf{G}^{-1} \mathbf{A}$ and $\mathbf{q} = \begin{bmatrix} \mathbf{0} \\ -\mathbf{j}_l \\ -\mathbf{j}_h \end{bmatrix} + \mathbf{A}^\top \mathbf{G}^{-1} \begin{bmatrix} -\mathbf{M}\mathbf{v}^n - h\mathbf{k} \\ -\mathbf{j}_c \end{bmatrix}$, assuming that \mathbf{G} is invertible.¹ To compute the inverse of \mathbf{G} , we use the Schur-complement (see Nocedal and Wright [25]):

$$\begin{bmatrix} \mathbf{M} \mathbf{J}^\top \\ \mathbf{J} \mathbf{0} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{C} & \mathbf{E} \\ \mathbf{E}^\top & \mathbf{F} \end{bmatrix} \quad \text{where} \quad \begin{aligned} \mathbf{F} &= -\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^\top \\ \mathbf{E} &= -\mathbf{M}^{-1}\mathbf{J}^\top \mathbf{F} \\ \mathbf{C} &= \mathbf{M}^{-1} - \mathbf{E}\mathbf{J}\mathbf{M}^{-1}. \end{aligned} \quad (19)$$

Solving the LCP (\mathbf{q}, \mathbf{S}) yields the solution variables \mathbf{c}_n , \mathbf{c}_l , and \mathbf{c}_h . Plugging these variables back into (18) yields the equality constrained variables. The advantage of solving this LCP, compared to the LCP used in ODE’s original solver, is that the former problem is convex and much more easily solvable. We use a gradient projection method (*c.f.* Nocedal and Wright [25]) which is always able to provide a solution to the LCP.

Once the LCP has been solved, a frictionless, energy conserving solution to the system has been obtained. Friction can be added to the system using a nonlinear convex optimization model; the initial feasible point for the model is the solution to the MLCP determined above. The advantage of adding friction with such a model is that the convex optimization solver can be terminated early with only a loss of solution accuracy: energy will only be removed from the system.

¹ \mathbf{G} is invertible as long as \mathbf{J} is of full row rank, due to positive-definiteness of \mathbf{M} ; we use a SVD-based pseudo-inverse which avoids this issue.

The nonlinear model consists of optimizing the following problem:

Optimize:	$\frac{1}{2} \mathbf{v}^{n+1} \mathbf{M} \mathbf{v}^{n+1}$	(Minimize kinetic energy)
Subject to:	$\mathbf{N} \mathbf{v}^{n+1} \geq \mathbf{n}_c$	(Noninterpenetration constraint)
	$\mathbf{J} \mathbf{v}^{n+1} = \mathbf{j}_c$	(Bilateral constraint)
	$\mathbf{J}_l \mathbf{v}^{n+1} \geq \mathbf{j}_l$	(Lower joint limit)
	$\mathbf{J}_h \mathbf{v}^{n+1} \geq \mathbf{j}_h$	(Upper joint limit)
	$\mathbf{c}_n \geq \mathbf{0}$	(Compressive constraint)
	$\mathbf{1}^\top \mathbf{c}_n \leq \kappa$	(Compressive limit constraint)
	$\mu_i^2 c_{n_i}^2 \geq c_{s_i}^2 + c_t^2$	(Friction cone constraint)

where $\mathbf{v}^{n+1} = \mathbf{M}^{-1}(\mathbf{N}^\top \mathbf{c}_n + \mathbf{J}^\top \mathbf{c}_j + \mathbf{J}_l^\top \mathbf{c}_l + \mathbf{J}_h^\top \mathbf{c}_h + \mathbf{S}^\top \mathbf{c}_s + \mathbf{T}^\top \mathbf{c}_t + h\mathbf{k}) + \mathbf{v}^n$.

The matrices S and T are the contact tangential twist matrices.

This nonlinear convex optimization problem exhibits the same ($O(N^3)$) complexity as the first phase of the model, though a significantly slower running time. Nevertheless, a full friction cone is obtained through the conic constraint above, and a ϵ -suboptimal solution to the model can be obtained with superlinear convergence using a primal dual interior-point solver [6].

5 Experiments

A variety of different scenarios were considered, ranging from elementary physics based experiments to complex, application oriented robotics scenarios.

5.1 Ball Drop

An initially stationary 1kg sphere with unit radius is dropped from a height of 5m onto a plane. Figure 1 plots the vertical position of the ball. Both *quick*

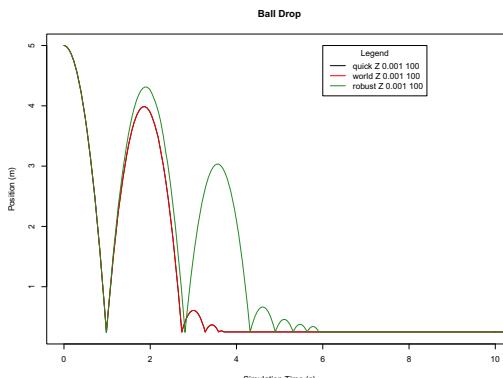
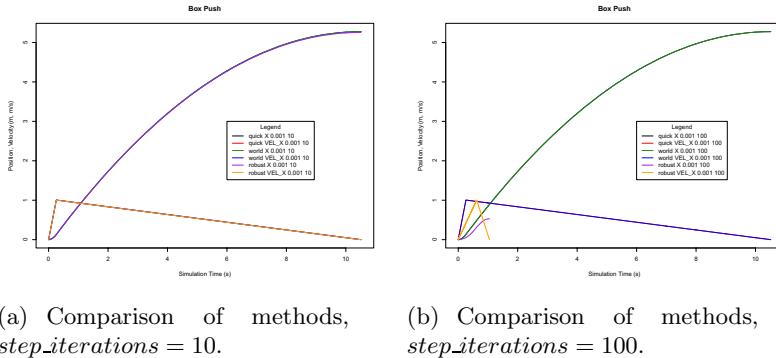


Fig. 1. Vertical (Z) position falling ball over time; comparison of methods, $step_iterations = 100$

and *world* (which is the standard ODE function) show the same behavior. The *robust* method exhibits qualitatively similar behavior; the difference could be explained by different interpretations of the coefficient of restitution parameter, and the result suggests that a straightforward function may relate the parameter for the first two cases, to a different value which produces identical behavior in the *robust* method.

5.2 Box Push

A 0.1kg unit cube initially rests on a plane; a 0.2N force along \mathbf{x} is applied until the box reaches a speed of 1ms^{-1} . The force is withdrawn and the box slides to a stop. Figure 2 shows a particularly interesting result. The *robust* method brings the object to rest sooner than either of the other two methods. Analysis of the particular impulses generated by the *robust* method are consistent and correct under the principle of maximum dissipation [28]. Comparing the left and right figures, shows that increasing the convex optimization solver's number of iterations does indeed remove greater energy, as shown in the different slopes. Comparison of this example to an implementation outside of ODE highlighted something else: the normal impulses produced by ODE in order to compensate for penetration (due its treatment which may permit some interpenetration) may be larger than absolutely necessary. This increases the apex height of friction cone, permitting larger tangential impulses, which in turn, affects the time to bring the box to a halt.



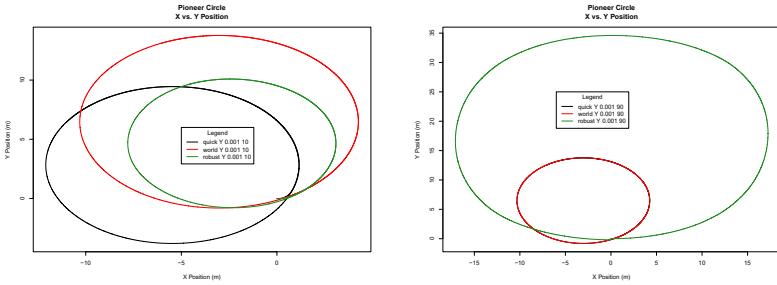
(a) Comparison of methods,
 $step_iterations = 10$.

(b) Comparison of methods,
 $step_iterations = 100$.

Fig. 2. Box Push Scenario

5.3 Pioneer Circle

A MobileRobots pioneer robot starts initially from rest. Torques of $2.2\text{N}\cdot\text{m}$ and $2.1\text{N}\cdot\text{m}$ are applied to the left wheel and right wheels respectively. The simulation terminates once 1.5 revolutions of the pioneer have been completed. Figure 3 illustrates the difference in circular performance. The difference in trajectory reflects friction cone approximation, and in the limit of large iterations, the robust method is expected to reproduce the true Coloumb friction behavior.



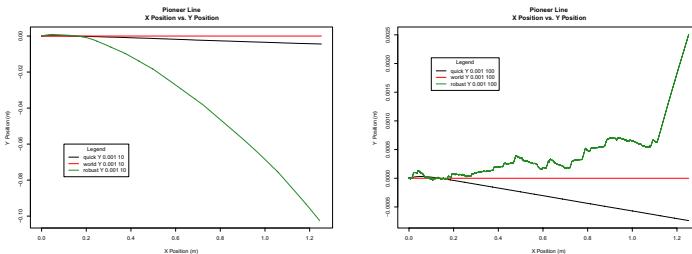
(a) Comparison of methods,
 $step_iterations = 10$.

(b) Comparison of methods,
 $step_iterations = 90$.

Fig. 3. Pioneer Circle Scenario

5.4 Pioneer Line

A MobileRobots pioneer robot starts initially from rest. Torques of 0.1N·m are applied to each wheel. The simulation terminates after 20s. Figure 4 shows the results from this scenario. Although unflattering, particularly to the *robust* method attention must be drawn to the comparatively small scale of the vertical axes in the plots, and hence the error.



(a) Comparison of methods,
 $step_iterations = 10$.

(b) Comparison of methods,
 $step_iterations = 100$.

Fig. 4. Pioneer Line Scenario

5.5 Pioneer Gripper

A MobileRobots pioneer robot is simulated as it applies forces to a box (resting on the ground plane) with its paddle-style gripper. The box, placed between the paddles, is held in place through application of 500N of force. Figures 5(a), 5(b), and 5(c) show the results of experiments run with $\Delta t = 0.01$ s and $step_iterations = 10$. All three methods result in instability in which the grasped box ends up moving away from the correct location. We observe a surprising qualitative similarity in the results of all methods.

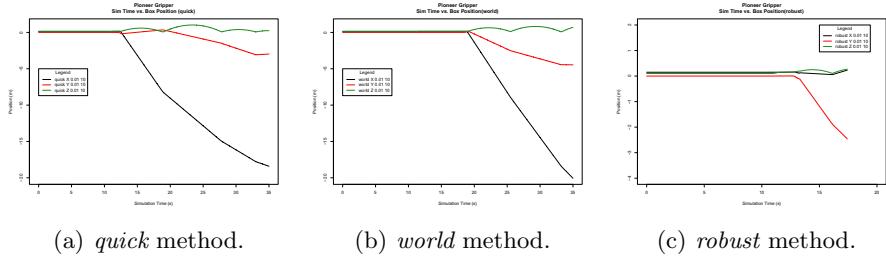


Fig. 5. Position of grasped box over time

5.6 Parallelization: Box Stacks

We simulated four stacks of six boxes 1,2,4,8 processors, each parallelizing the box stacks row-wise. Each process drove residuals to 10×10^{-3} at every time step. Figure 6 shows the result.

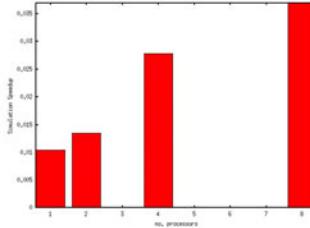


Fig. 6. Speedup of box stacks

6 Conclusion

We have focused on identifying problems that affect the Open Dynamics Engine (ODE), the rigid-body dynamics implementation used by the most popular robotics simulators, when attempting to simulate simple mobile robot locomotion and grasping. We have contributed the following: a parallelized implementation of ODE's quick step, viscous joint dampening, and a convex-optimization contact method. Our implementation has permitted us explore the behavior and identify some trade-offs and approximations involved (*e.g.*, robust method yields a true friction cone, but ODE may unphysically increase normal impulses) involved.

Much remains as future work: implementations of robust step method and parallel quick-step remain imperfect, and optimization remains possible. Nevertheless, we offer these tools as a mechanism for researchers needing their benefits.

References

- [1] Anitescu, M., Potra, F.A.: Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics* 14, 231–247 (1997)
- [2] Anitescu, M., Tasora, A.: An iterative approach for cone complementarity problems for nonsmooth dynamics. In: Computational Optimization and Applications (2008)
- [3] Arechavaleta, G., López-Damian, E., Morales, J.L.: On the Use of Iterative LCP Solvers for Dry Frictional Contacts in Grasping. In: International Conference on Advanced Robotics, Munich, Germany, pp. 1–6 (June 2009)
- [4] Baraff, D.: Linear-time dynamics using Lagrange multipliers. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH 1996, pp. 137–146 (1996)
- [5] Boeing, A., Bräunl, T.: Evaluation of real-time physics simulation systems. In: Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia - GRAPHITE 2007, vol. 1(212), p. 281 (2007)
- [6] Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press, Cambridge (2004)
- [7] Browning, B., Tryzelaar, E.: ÜberSim: A Realistic Simulation Engine for Robot Soccer. In: Proceedings of Autonomous Agents and Multi-Agent Systems (AA-MAS 2003), Melbourne, Australia (July 2003)
- [8] Acosta Calderon, C.A., Mohan, R.E., Zhou, C.: Virtual-RE: A Humanoid Robotic Soccer Simulator. In: Proceedings of International Conference on Cyberworlds, Hangzhou, China, pp. 561–566 (September 2008)
- [9] NVIDIA Corporation. PhysX SDK (2008),
http://www.nvidia.com/object/nvidia_physx.html
- [10] Cottle, R.W., Pang, J.-S., Stone, R.E.: The Linear Complementarity Problem. Academic Press, Boston (1992)
- [11] Coumans, E.: Bullet Physics Engine For Rigid Body Dynamics,
<http://bulletphysics.org/>
- [12] Demur, K.: Robot Simulation — Robot programming with Open Dynamics Engine. Morikita Publishing Co. Ltd., Tokyo (2007)
- [13] Fijany, A.: New Factorization Techniques and Parallel O (Log N) Algorithms for Forward Dynamics Solution of Single Closed-Chain Robot Manipulators. Jet Propulsion Laboratory, California Institute of Technology
- [14] Finkenzeller, D., Baas, M., Thüring, S., Yigit, S., Schmitt, A.: VISUM: A VR system for the interactive and dynamics Simulation of mechatronic systems. In: Fischer, X., Coutellier, D. (eds.) Research in Interactive Design: Proceedings of Virtual Concept 2003, Biarritz, France. Springer, Heidelberg (2003)
- [15] Friedmann, M., Petersena, K., von Stryk, O.: Adequate motion simulation and collision detection for soccer playing humanoid robots. *Robotics and Autonomous Systems* 57(8), 786–795 (2009)
- [16] Garstenauer, H., Kurka, D.I.D.G.: A unified framework for rigid body dynamics. Degree Paper (2006)
- [17] Jung, D.: Opensim (2006), <http://opensimulator.sourceforge.net>
- [18] Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: Proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), Sendai, Japan, pp. 2149–2154 (September 2004)

- [19] Laue, T., Spiess, K., Röfer, T.: SimRobot — A General Physical Robot Simulator and Its Application in RoboCup. In: Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y. (eds.) RoboCup 2005. LNCS (LNAI), vol. 4020, pp. 173–183. Springer, Heidelberg (2006)
- [20] Lee, J., Oh, J.H.: Biped Walking Pattern Generation Using Reinforcement Learning. *International Journal of Humanoid Robotics* 6(1), 1–21 (2009)
- [21] Lima, J.L., Goncalves, J.C., Costa, P.G., Moreira, A.P.: Realistic Behaviour Simulation of a Humanoid Robot. In: Proceedings of 8th Conference on Autonomous Robot Systems and Competitions, Aveiro, Portugal (April 2008)
- [22] Mangasarian, O.L., Leone, R.: Parallel gradient projection successive overrelaxation for symmetric linear complementarity problems and linear programs. *Annals of Operations Research* 14(1), 41–59 (1988)
- [23] Michel, O.: Webots: Professional Mobile Robot Simulation. *International Journal of Advanced Robotic Systems* 1(1), 39–42 (2004)
- [24] Moores, B.T., MacDonald, B.A.: A dynamics simulation architecture for robotic systems. In: Proc. of IEEE Intl. Conf. on Robotics and Automation (ICRA), Barcelona, Spain (April 2005)
- [25] Nocedal, J., Wright, S.J.: Numerical Optimization, 2nd edn. Springer, Heidelberg (2006)
- [26] Obst, O., Rollmann, M.: SPARK – A Generic Simulator for Physical Multiagent Simulations. In: Lindemann, G., Denzinger, J., Timm, I.J., Unland, R. (eds.) MATES 2004. LNCS (LNAI), vol. 3187, pp. 243–257. Springer, Heidelberg (2004)
- [27] Smith, R.: ODE: Open Dynamics Engine
- [28] Stewart, D.E.: Rigid-body dynamics with friction and impact. *SIAM Review* 42(1), 3–39 (2000)
- [29] Sugiura, N., Takahashi, M.: Development of a Humanoid Robot Simulator and Walking Motion Analysis. In: Workshop Proceedings of SIMPAR, International Conference on Simulation, Modeling and Programming for Autonomous Robots, Venice, Italy, pp. 151–158 (November 2008)
- [30] Tikhanoff, V., Fitzpatrick, P., Metta, G., Natale, L., Nori, F., Cangelosi, A.: An Open-Source Simulator for Cognitive Robotics Research: The Prototype of the iCub Humanoid Robot Simulator. In: Workshop on Performance Metrics for Intelligent Systems, National Institute of Standards and Technology, Washington DC, USA (August 2008)
- [31] Wolff, K., Wahde, M.: Evolution of Biped Locomotion Using Linear Genetic Programming, ch. 16, pp. 335–356. Itech Education and Publishing, Vienna (October 2007)
- [32] Yeh, T., Reinman, G., Patel, S.J., Faloutsos, P., Ageia Technologies: Fool Me Twice: Exploring and Exploiting Error Tolerance in Physics-Based Animation. In: Simulation (2006)
- [33] Zagal, J.C., Del Solar, J.R.: UchilSim: A Dynamically and Visually Realistic Simulator for the RoboCup Four Legged League. In: Nardi, D., Riedmiller, M., Sammut, C., Santos-Victor, J. (eds.) RoboCup 2004. LNCS (LNAI), vol. 3276, pp. 34–45. Springer, Heidelberg (2005)

Virtual Robot Experimentation Platform

V-REP: A Versatile 3D Robot Simulator

Marc Freese*, Surya Singh, Fumio Ozaki, and Nobuto Matsuhira

*K-Team Corporation, Y-Parc - Rue Galilée 9,
1400 Yverdon-les-Bains, Switzerland

mfreese@gmx.ch, spns@acfr.usyd.edu.au,

{fumio.ozaki,nobuto.matsuhira}@toshiba.co.jp

www.v-rep.eu, www.k-team.com

Abstract. From exploring planets to cleaning homes, the reach and versatility of robotics is vast. The integration of actuation, sensing and control makes robotics systems powerful, but complicates their simulation. This paper introduces a modular and decentralized architecture for robotics simulation. In contrast to centralized approaches, this balances functionality, provides more diversity, and simplifies connectivity between (independent) calculation modules. As the Virtual Robot Experimentation Platform (V-REP) demonstrates, this gives a small-footprint 3D robot simulator that concurrently simulates control, actuation, sensing and monitoring. Its distributed and modular approach are ideal for complex scenarios in which a diversity of sensors and actuators operate asynchronously with various rates and characteristics. This allows for versatile prototyping applications including systems verification, safety/remote monitoring, rapid algorithm development, and factory automation simulation.

Keywords: Robot Simulator, V-REP, Distributed Control.

1 Introduction

The overall architecture and control methodology are crucial elements in robot simulators. A robust systems approach advocates for a versatile and fine-grained simulation strategy. By emphasizing modularity, scalability and expandability, simulations remain robust particularly when abstracting underlying robotic systems since system specificities cannot be foreseen.

The increased processing power of computers, the advent of several dynamics (physics) libraries, as well as faster 3D graphics hardware have drastically changed the landscape in the field of (3D) robotics simulation. While it is possible to cobble together these various elements almost trivially, good simulation requires careful architecture to yield both performance and accurate calculations. For example, rather than using dynamics blindly for every aspect of a simulation, it is preferable to use it only when other methods (e.g. kinematics) fail. A modular architecture allows for the combination of various functionality to obtain the best possible synergy.

Compared to robots a few years ago, robots now employ highly complex control methods. Practically those are implemented in a distributed fashion in order to simplify development and the overall control task complexity. Just as distributed communication networks and protocols allow a *plug-and-play* behavior [3], a distributed control approach is needed for robotic simulation. In addition to robustness and parallelism (similar to the distributed hardware), it allows a robotic simulator to offer a *copy-and-paste* functionality not only for objects or models, but also for their associated control methods: robots or robot elements can be placed and combined in a scene without having to adjust any control code.

The paper is organized in three parts. The first part of this paper focuses on V-REP's architecture; based on various functionalities wrapped in *scene objects* or calculation modules, it allows the user to pick from them as required, or as best for a given simulation. Moreover a special care has been given so as to develop those functionalities in a balanced manner, giving each one the same amount of attention as one would do in a real robotic system. The second part of this paper focuses on V-REP's control methodology; in V-REP, control is distributed and based on an unlimited number of scripts that are directly associated or attached to *scene objects*. Finally, in order to further clarify elements introduced in the first and second part of this paper, the third part describes a typical V-REP simulation set-up.

2 V-REP's Architecture

V-REP is designed around a versatile architecture. There is no *main* or *central* functionality in V-REP. Rather, V-REP possesses various relatively independent functionalities, that can be enabled or disabled as required.

Imagine a simulation scenario where an industrial robot has to pick-up boxes and move them to another location; V-REP computes the dynamics for grasping and holding the boxes and performs a kinematic simulation for the other parts of the cycle when dynamic effects are negligible. This approach makes it possible to calculate the industrial robot's movement quickly and precisely, which would not be the case had it been simulated entirely using complex dynamics libraries. This type of hybrid simulation is justified in this situation, if the robot is stiff and fixed and not otherwise influenced by its environment.

In addition to adaptively enabling various of its functionalities in a selective manner, V-REP can also use them in a symbiotic manner, having one cooperate with another. In the case of a humanoid robot, for example, V-REP can handle leg movements by (a) first calculating inverse kinematics for each leg (i.e., from a desired foot position and orientation, all leg joint positions are calculated); and then (b) assigning the calculated joint positions to be used as target joint positions by the dynamics module. This allows specifying the humanoid motion in a very versatile way, since each foot would simply have to be assigned to follow a 6-dimensional path: the rest of calculations are automatically taken care of.

2.1 Scene Objects

A V-REP simulation scene contains several *scene objects* or elemental objects that are assembled in a tree-like hierarchy. The following *scene objects* are supported in V-REP:

- **Joints:** *joints* are elements that link two or more *scene objects* together with one to three degrees of freedom (e.g., prismatic, revolute, screw-like, etc.).
- **Paths:** *paths* allow complex movement definitions in space (succession of freely combinable translations, rotations and/or pauses), and are used for guiding a welding robot’s torch along a predefined trajectory, or for allowing conveyor belt movements for example. Children of a path object can be constrained to move along the path trajectory at a given velocity.
- **Shapes:** *shapes* are triangular meshes, used for rigid body visualization. Other *scene objects* or calculation modules rely on *shapes* for their calculations (collision detection, minimum distance calculation, etc.).
- **Cameras and lights:** *cameras* and *lights* are used for scene visualization purposes mainly, but can also have effects on other *scene objects* (e.g. *lights* directly influence *rendering sensors* (see hereafter)).
- **Dummies:** *dummies* are “points with orientation,” or reference frames, that can be used for various tasks, and are mainly used in conjunction with other *scene objects*, and as such can be seen as “helpers.”
- **Proximity sensors:** the *proximity sensors* *scene objects* perform an exact minimum distance calculation within a given detection volume  (see Fig. II) as opposed to simply performing collision detection between some selected *sensing rays* and the environment. They allow specifying a limit angle for surface normals, and through their operation mode smoothen the sensor reading to a more continuous operation.
- **Rendering sensors:** *rendering sensors* in V-REP are camera-like sensors, allowing to extract complex image information from a simulation scene (colors, object sizes, depth maps, etc.) (see Fig. II). The built-in filtering and image processing enable the composition of blocks of filter elements (with additional filter elements via plugins). *Rendering sensors* make use of hardware acceleration for the raw image acquisition (OpenGL).
- **Force sensors:** *force sensors* are rigid links between *shapes*, that can record applied forces and torques, and that can conditionally break apart when a given threshold is overshot.
- **Mills:** *mills* are customizable convex volumes that can be used to simulate surface cutting operations on *shapes* (e.g., milling, laser cutting, etc.).
- **Graphs:** *graphs* are *scene objects* that can record a large variety of one dimensional data streams. Data streams can be displayed directly (time graph of a given data type), or combined with each other to display X/Y graphs, or 3D curves.

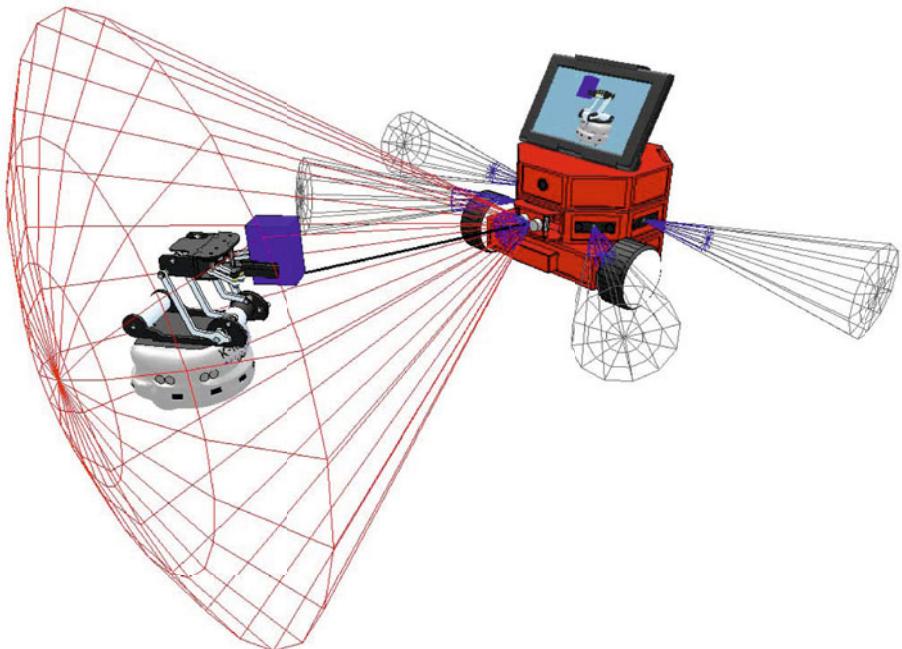


Fig. 1. Mobile robot (right) equipped with 5 *proximity sensors* and one *rendering sensor*. The *rendering sensor* is not used for detection in this case, but for texture generation for the LCD panel. (The left robot model is courtesy of K-Team Corp., the right robot model is courtesy of Cubictek Corp. and NT Research Corp.).

2.2 Calculation Modules

Scene objects are rarely used on their own, they rather operate on (or in conjunction with) other *scene objects* (e.g. a *proximity sensor* will detect *shapes* or *dummies* that intersect with its detection volume). In addition, V-REP has several calculation modules that can directly operate on one or several *scene objects*. Following are V-REP’s main calculation modules:

- **Forward and inverse kinematics module:** allows kinematics calculations for any type of mechanism (branched, closed, redundant, containing nested loops, etc.). The module is based on calculation of the damped least squares pseudoinverse [7]. It supports conditional solving, damped and weighted resolution, and obstacle avoidance based constraints.
- **Dynamics or physics module:** allows handling rigid body dynamics calculation and interaction (collision response, grasping, etc.) via the Bullet Physics Library [1].
- **Path planning module:** allows holonomic path planning tasks and non-holonomic path planning tasks (for car-like vehicles) via an approach derived from the *Rapidly-exploring Random Tree (RRT)* algorithm [6].
- **Collision detection module:** allows fast interference checking between any *shape* or collection of *shapes*. Optionally, the collision contour can also

be calculated. The module uses data structures based on a binary tree of Oriented Bounding Boxes [5] for accelerations. Additional optimization is achieved with a temporal coherency caching technique.

- **Minimum distance calculation module:** allows fast minimum distance calculations between any *shape* (convex, concave, open, closed, etc.) or collection of *shapes*. The module uses the same data structures as the collision detection module. Additional optimization is also achieved with a temporal coherency caching technique.

Except for the dynamics or physics module that directly operates on all dynamically enabled *scene objects*, other calculation modules require the definition of a calculation task or calculation object, that specifies on which *scene objects* the module should operate and how. If for example the user wishes to have the minimum distance between **shape A** and **shape B** automatically calculated and maybe also recorded, then a minimum distance object has to be defined, having as parameters **shape A** and **shape B**. Fig. 2 shows V-REP's typical simulation loop, including main *scene objects* and calculation modules.

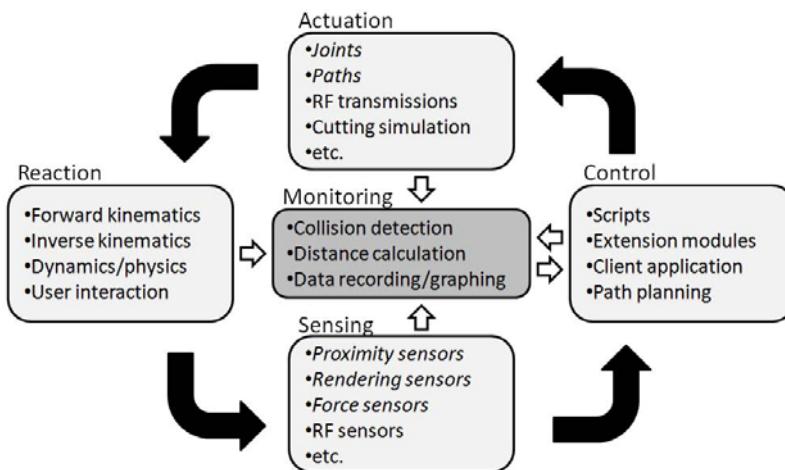


Fig. 2. Simulation loop in V-REP

2.3 Scalability

Destruction of one or several *scene objects* can involve automatic destruction of an associated calculation object. In a similar way, duplication of one or several *scene objects* can involve automatic duplication of associated calculation objects. This also includes automatic duplication of associated control scripts (see next section). The result of this is that duplicated *scene objects* will automatically be fully functional, allowing a flexible *plug-and-play* like behaviour.

3 V-REP’s Control Methodology

V-REP offers various means for controlling simulations or even to customizing the simulator itself (see Fig. B). V-REP is wrapped in a function library, and requires a client application to run. The V-REP default client application is quite simple and takes care of loading extension modules, registers event callbacks (or message callbacks), relays them to the loaded extension modules, initializes the simulator, and handles the application and simulation loop. In addition, custom simulation functions can be added via:

- **Scripts in the *Lua* language.** *Lua* [2] is a lightweight extension programming language designed to support procedural programming. The *Lua* script interpreter is embedded in V-REP, and extended with several hundreds of V-REP specific commands. Scripts in V-REP are the main control mechanism for a simulation.
- **Extension modules to V-REP (plugins).** Extension modules allow for registering and handling custom commands or functions. A high-level script command (e.g., *robotMoveAndAvoidObstacles(duration)*) can then be handled by an extension module which will execute the corresponding logic and low-level API function calls in a fast and hidden fashion.

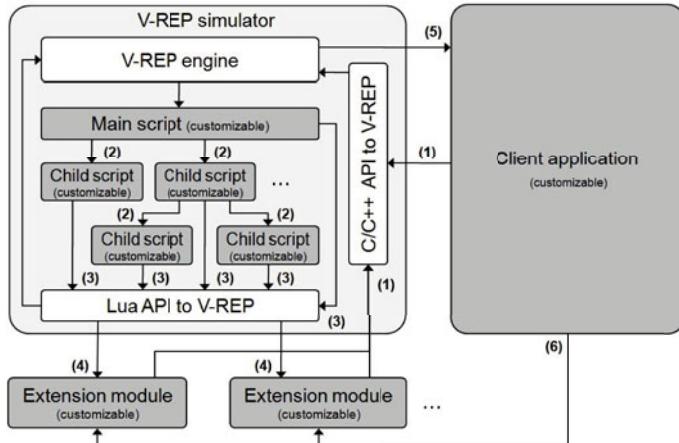


Fig. 3. Control architecture in V-REP. Greyed areas can be customized by the user. (1) C/C++ API calls to V-REP from the client application, or from extension modules. (2) Script handling calls. Typically *simHandleChildScript(sim_handle_all)*. Executes all first encountered child scripts in the current hierarchy. (3) Lua API calls to V-REP from scripts. (4) Callback calls to extension modules. Originate when a script calls a custom function, previously registered by an extension module. (5) Event callback calls to the client application. (6) Relayed event calls to extension modules.

3.1 Script Calling Methodology

A simulation is handled when the client application calls a *main script*, which in turn can call *child scripts*.

Each simulation scene has exactly one *main script* that handles all default behaviour of a simulation, allowing simple simulations to run without even writing a single line of code. The *main script* is called at every simulation pass and is non-threaded.

Child scripts on the other hand are not limited in number, and are associated with (or attached to) *scene objects*. As such, they are automatically duplicated if the associated *scene object* is duplicated. In addition to that, duplicated scripts do not need any code adjustment and will automatically fetch correct object handles when accessing them. *Child scripts* can be non-threaded or threaded (i.e. launch a new thread).

The default *child script* calling methodology is hierachial; each script is in charge of calling all first encountered *child scripts* in the current hierarchy (since *scene objects* are built in a tree-like hierarchy, scripts automatically inherit the same hierarchy). This is achieved with a single function call: *simHandleChildScript(sim_handle_all)*.

Taking the example of Fig. 4, when the *main script* calls *simHandleChildScript(sim_handle_all)*, then *child scripts* associated with objects 3, 4 and 7 will be executed. Only when the *child script* associated with object 3 in its turn calls *simHandleChildScript(sim_handle_all)*, will *child script* associated with object 6 also be executed.

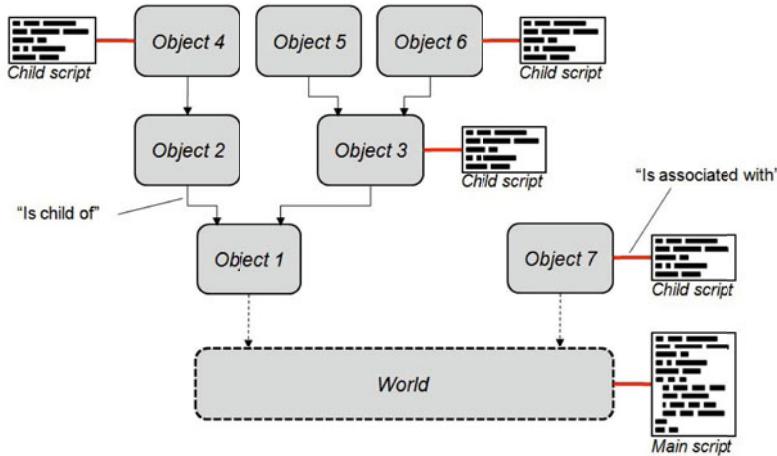


Fig. 4. Main script and child scripts in a scene

The default *main script* will always call *simHandleChildScript(sim_handle_all)*, and *child scripts* should do the same.

3.2 Non-threaded *Child Script* Example

Following code illustrates an empty non-threaded *child script* in V-REP:

```
if (simGetScriptExecutionCount()==0) then
    -- Initialization code comes here
end
simHandleChildScript(sim_handle_all) -- Handles child scripts
-- Main code comes here
if (simGetSimulationState()==sim_simulation_advancing_lastbeforestop) then
    -- Restoration code comes here
end
```

Non-threaded *child scripts* are "pass-through", which means that at each simulation pass they will execute, and directly return control to the caller. The caller can provide input parameters (input arguments). When a *child script* is called explicitly (i.e. *simHandleChildScript("childScriptID")*) instead of *simHandleChildScript(sim_handle_all)*, then it can also return output parameters (return values).

3.3 Threaded *Child Script* Example

Threaded *child scripts* require a slightly differentiated handling, since other *child scripts* built on top of them should also be guaranteed to be executed at each simulation pass. Following code illustrates an empty, threaded *child script* in V-REP:

```
simDelegateChildScriptExecution() -- Delegates child script execution
simSetThreadSwitchTiming(100) -- optional
-- Initialization code comes here
while (simGetSimulationState()~=sim_simulation_advancing_abouttostop) do
    -- Main code comes here
    simSwitchThread() -- optional
end
-- Restoration code comes here
```

As can be seen from above code, threaded *child scripts* should delegate their *child script* execution to the *main script* to make sure they will be called at each simulation pass. The code also shows a particularity of V-REP's threads: V-REP doesn't use *regular* threads, but rather coroutines. The advantage of this is a greater flexibility in thread execution timing, with possibility of synchronization with the *main script*. Indeed, *simSetThreadSwitchTiming* sets the time after which the thread should automatically switch to another thread. The switching can also be explicitly performed with *simSwitchThread*. Control is given back to threads each time the *main script* is about to execute.

3.4 Scalability

This distributive, hierachial script execution mechanism makes the handling of newly added (e.g. copy/pasted) *scene objects* or models very easy, since associated *child scripts* will automatically be executed, without having to adjust or

modify any code. Additionally, added *child scripts* will not be executed in a random order, but according to their position within the scene hierarchy.

Extension modules to V-REP seamlessly integrate into this distributive control approach: extending V-REP with a specific robot language becomes as easy as wrapping the robot language interpreter into a V-REP extension module. A similar approach can be taken to embed emulators (e.g. microcontroller emulators) into V-REP, in order to control a simulation natively for instance.

Finally, V-REP offers sofisticated messaging mechanisms. In particular for inter-script communications; messages can be global (i.e. can be received by all *child scripts*), local (i.e. can be received only by *child scripts* in the current hierarchy), or direct (i.e. can only be received by a single specific *child script*).

4 Example Simulation Set-Up

Following example scene in V-REP (see Fig. 5) clarifies several previously mentioned aspects of the simulator.

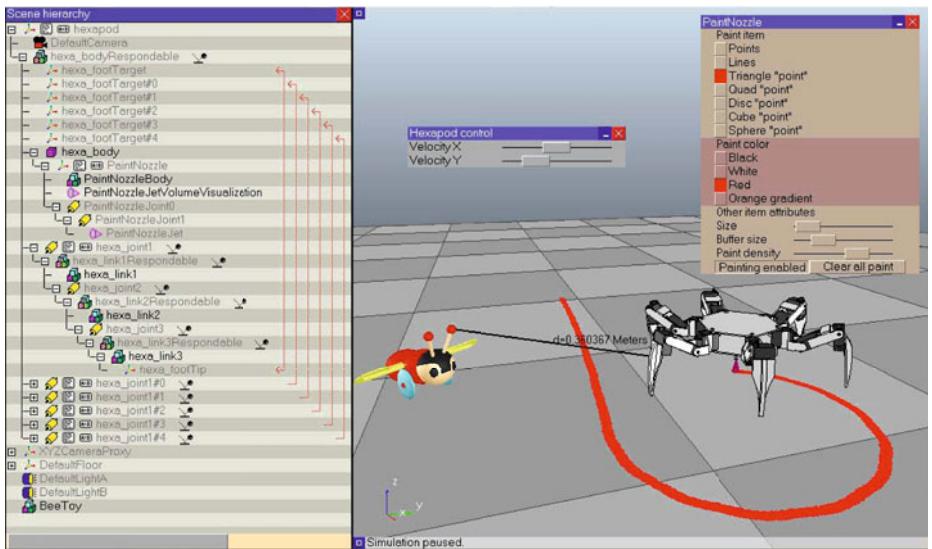


Fig. 5. V-REP example scene. The left part shows the scene hierarchy, the right part shows the scene 3D content.

The scene contains a hexapod walking robot (defined by the hierarchy tree starting at the *scene object* "hexapod"), and a little toy *scene object* ("Bee-Toy"). The hexapod itself is set-up with 6 identical legs rotated by 60 degrees relative to each others. For each leg, an inverse kinematics object was defined that resolves the correct leg joint positions for a given desired foot position; inverse kinematics constraints are indicated in the scene hierarchy with red arrows (e.g. "hexa_footTarget" is the desired foot position for "hexa_footTip").

The joint positions, calculated by the forward and inverse kinematics module, are then applied as target positions by the dynamics or physics module. While a *child script* associated with "hexapod" is in charge of generating a foot motion sequence, each leg has an own *child script* that will apply the generated foot motion sequence with a different delay. Except for that delay, the 6 leg *child scripts* are identical and the first one is reproduced here:

```

baseHandle=... -- Handle of the base object ("hexapod")
if (simGetScriptExecutionCount()==0) then
    modulePos=0 -- Movement delay for this leg
    -- Retrieve various handles and prepare initial values:
    tip=simGetObjectHandle('hexa_footTip')
    target=simGetObjectHandle('hexa_footTarget')
    j1=simGetObjectHandle('hexa_joint1')
    j2=simGetObjectHandle('hexa_joint2')
    j3=simGetObjectHandle('hexa_joint3')
    simSetJointPosition(j1,0)
    simSetJointPosition(j2,-30*math.pi/180)
    simSetJointPosition(j3,120*math.pi/180)
    footOriginalPos=simGetObjectPosition(tip,baseHandle)
    isf=simGetObjectSizeFactor(baseHandle)
end
-- Read the movement data:
data=simReceiveData(0,'HEXA_x')
xMovementTable=simUnpackFloats(data)
data=simReceiveData(0,'HEXA_y')
yMovementTable=simUnpackFloats(data)
data=simReceiveData(0,'HEXA_z')
zMovementTable=simUnpackFloats(data)
data=simReceiveData(0,'HEXA_imd')
interModuleDelay=simUnpackInts(data)[1]
-- Make sure that scaling during simulation will work flawlessly:
sf=simGetObjectSizeFactor(baseHandle)
af=sf/isf
-- Apply the movement data (with the appropriate delay):
targetNewPos={
    footOriginalPos[1]*af+xMovementTable[1+modulePos*interModuleDelay]*sf,
    footOriginalPos[2]*af+yMovementTable[1+modulePos*interModuleDelay]*sf,
    footOriginalPos[3]*af+zMovementTable[1+modulePos*interModuleDelay]*sf}
-- The IK module will automatically have the foot tip follow the "target",
-- so we just need to set the position of the "target" object:
simSetObjectPosition(target,baseHandle,targetNewPos)
-- Make sure that any attached child script will also be executed:
simHandleChildScript(sim_handle_all_except_explicit)

```

A minimum distance object between all *scene objects* composing the hexapod and all other *scene objects* was defined so that the hexapod clearance can be tracked. A paint nozzle model was attached to the hexapod and allows marking the floor with color. The paint nozzle model operates independently from

the hexapod robot and is handled by its own *child script*. The *scene objects* "hexapod" and "PaintNozzle" are associated with 2 custom dialogs that allow users to interact with a script's behaviour. This represents another powerful feature in V-REP: an unlimited number of custom dialogs can be defined and associated with *scene objects*. They are destroyed or duplicated in a similar way as calculation objects (refer to section 2.3).

5 Conclusion

V-REP demonstrates a modular simulation architecture combined with a distributed control mechanism. This results in a versatile and scalable framework that fits the simulation needs of complex robotic systems.

V-REP provides a balanced functionality through a multitude of additional calculation modules, offering a real advantage in terms of simulation fidelity and simulation completeness (see Fig. 6).

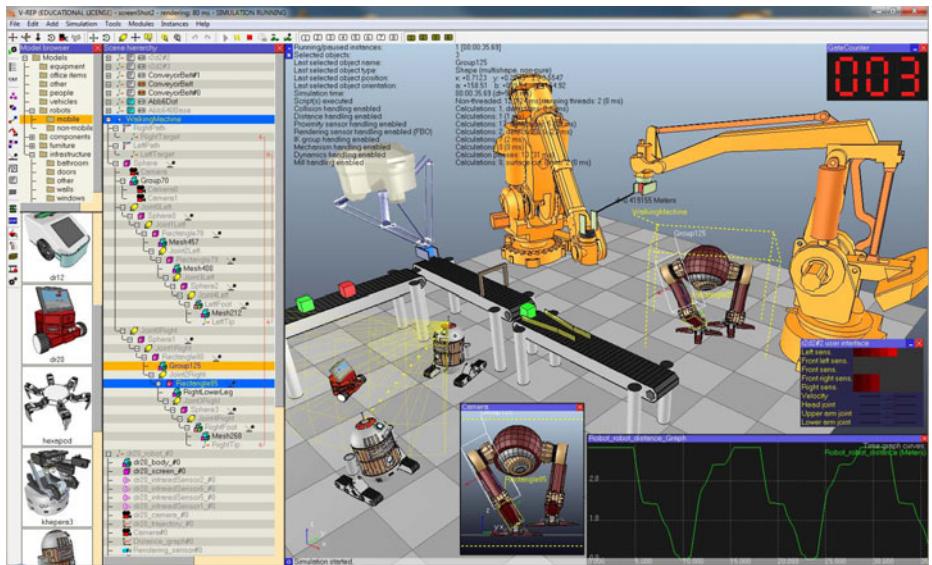


Fig. 6. Screen shot of V-REP's application main window (robot models are courtesy of Lyall Randell, Cubictek Corp., NT Research Corp., and ABB Corp. there is no link of any kind between V-REP and ABB Corp.)

One central aspect further enabling V-REP's versatility and scalability is its distributed control approach; it is possible to test configurations with 2, 3, or up to several hundreds of (identical or different) robots (e.g. in a swarm configuration), by simple drag-and-drop or copy-and-paste actions. There is no need for code adjustment, not even a supervisor program is required. V-REP's versatility and scalability allows for this *plug-and-play* like behavior.

Finally, V-REP's expandability through extension modules (plugins), gives the user an easy and fast way to customize a simulation, or the simulator itself.

References

1. Bullet physics library, <http://www.bulletphysics.org>
2. Lua, <http://www.lua.org>
3. Ando, N., Suehiro, T., Kitagaki, K., Kotoku, T., Yoon, W.: RT-component object model in RT-middleware - distributed component middleware for RT (robot technology). In: 2005 IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA 2005), Espoo, Finland, pp. 457–462 (June 2005)
4. Freese, M., Ozaki, F., Matsuhira, N.: Collision detection, distance calculation and proximity sensor simulation using oriented bounding box trees. In: 4th International Conference on Advanced Mechatronics, Asahikawa, Japan, pp. 13–18 (October 2004)
5. Gottschalk, S., Lin, M.C., Manocha, D.: OBB-tree: a hierarchical structure for rapid interference detection. In: ACM SIGGRAPH, New Orleans, USA, pp. 171–180 (October 1996)
6. Kuffner Jr., J.J.: RRT-connect: an efficient approach to single-query path planning. In: IEEE International Conference on Robotics and Automation, San Fransisco, USA, pp. 995–1001 (April 2000)
7. Wampler, C.W.: Manipulator inverse kinematic solutions based on vector formulations and damped least squares methods. *IEEE Trans. Syst., Man, Cybern.* 16(1), 93–101 (1986)

Evaluation and Enhancement of Common Simulation Methods for Robotic Range Sensors

Martin Friedmann, Karen Petersen, and Oskar von Stryk

Technische Universität Darmstadt, Department of Computer Science
Hochschulstr. 10, D-64289 Darmstadt, Germany
{friedmann,petersen,stryk}@sim.tu-darmstadt.de
<http://www.sim.tu-darmstadt.de>

Abstract. Distance sensors are an important class of external sensors used in many autonomous robots. Thus it is of importance to provide proper simulation for these sensors to enable software-in-the-loop testing of a robot's control software. Two different methods for distance calculation are commonly applied for the simulation of such sensors, namely reading back the depth buffer from 3D renderings and the calculation of ray-object-intersections. Many simulators impose restrictions on either method, none of the widely used robot simulators reviewed in this paper currently considers material dependent simulation of the distances measured.

In this paper extended versions of both methods are presented which provide additional information on the object perceived by distance sensors. Several methods for incorporating distance- and object-information into a complete distance-sensor simulation-module are discussed. Implementations of either method are compared for their performance depending on the sensor resolution on different computer systems. These measurements show, that the break even of the methods strongly depends on the hardware, thus stressing the importance of providing either method in a robot simulation in a transparent way in order to obtain optimal performance of the simulation.

1 Introduction

Laser range finders and other distance sensors like time-of-flight (ToF) cameras or ultrasound sensors are important for many fundamental capabilities of mobile, autonomous robots like mapping or obstacle avoidance. To support development of software for mobile autonomous robots, many existing simulators provide simulation of distance sensors.

Two major approaches are usually applied to simulate the distances measured by the sensor: calculating ray-object intersections using ray-casting or rendering the scene from the sensor's point of view and subsequently reading back the depth buffer. As shown in [1, 2] either method has specific advantages and drawbacks, depending on the geometry and resolution of the simulated device. Nevertheless many existing robot simulators are limited to only one of these methods, or

impose limitations on the devices simulated with either method. Even though the material of the objects perceived by a distance sensor has significant impact on the error produced by the sensor (e. g. [3]) none of the investigated simulators provides material dependent error models for the sensors.

In this paper both commonly used methods are compared according their performance. Further on it is discussed how either method can be extended to provide information on the object perceived by the sensor.

The remainder of this paper is structured as follows: In Section 2 an overview of the simulation capabilities for distance sensors found in current simulators is given. In Section 3 the authors' implementations of both general methods including the aforementioned enhancements are presented. After this, simulators using these implementations and extensive measurements of the performance of the implementations are presented and discussed in section 4. The paper closes with concluding remarks and an outlook in Section 5.

2 Simulation of Distance Sensors in Robot Simulators

Many simulators used throughout the autonomous robots community provide simulation for distance sensors. In this section the respective simulation capabilities are discussed and summarized in Table 1.

Gazebo [4], the 3D dynamics simulation developed as part of the Player-Stage-project, is based on the Open-Dynamics-Engine (ODE) for physics simulation. The collision detection mechanisms provided by ODE are used to calculate ray-object intersections which are used to simulate distance sensors.

Microsoft Robotics Studio [5] provides a simulation for mobile robots, including laser range finders. It is not known, though, how this simulation of distance sensors is realized in detail.

Simbad [6,7] is a 3D robots simulation implemented in Java. Simulation of distance sensors is based on the picking functions provided by the Java3D API.

SimRobot [8,9] is a general 3D robot simulation based on ODE. Two methods for the simulation of distance sensors are provided: readings from the depth-buffer and calculation of ray-object intersections based on the ODE collision detection. The latter, though, is limited to single rays.

Stage [10] is a 2D simulation for mobile robots which has been developed as part of the Player-Stage-Project. Simulation of distance sensors is only available for measurements in the 2D plane. It is based on calculating intersections between rays and the 2D scene representation. Version 3 of the simulation introduced a 3D representation for the scene [11]. In this version, distance sensors can be simulated by ray-intersection in 3D space or by evaluating the depth-buffer.

USARSim [12,13] is a multi-robot-simulation based on the game engine of Unreal Tournament. The simulation is realized as a set of scripts executed by the game-engine as well as game levels representing the simulated scenarios. As the scripts only are allowed limited access to the internals of the game engine, simulation of distance-sensors can only be done by intersecting individual rays with the scene, but not by evaluating the depth-buffer.

Webots [14][15] is a commercially available simulation for mobile robots. Simulation of range finders is provided by evaluating the depth-buffer of OpenGL based 3D renderings. Besides the standard pinhole projection a method for the simulation of spherical projections based on multiple rendering passes is provided. Other distance sensors can be simulated by calculation of ray-object intersections. This method is limited to single rays and cones of equally distributed rays (defined by aperture angle and number of rays). The readings of the simulated sensors can be post-processed to simulate different output characteristics including noise.

Table 1. Overview of simulation methods for distance sensors in existing simulators

Simulator	Simulation method Raycasting	Depthbuffer
Gazebo	yes	no
MSRS	yes (method unknown)	
Simbad	yes	no
SimRobot	yes (limited to single rays)	yes
Stage 2.1.1	yes (2D simulation, limited to plane)	no
Stage 3.2	yes (limited 3D support due to model of environment)	yes
USARSim	yes (only single ray, fixed cones and 2D-sweeps)	yes
Webots	yes (only single ray and predefined distributions)	yes

Discussion. Many of the discussed simulators impose restrictions on the directions of rays in the ray-intersection based simulated method. Not all simulators provide simulation of distance sensors based on the depth buffer, only few of those provide simulation beyond the pinhole model. These circumstances complicate a transparent exchange of the simulation method, if it is possible at all.

Some of the investigated simulators allow to modify the measurements of the simulated sensors to reproduce sensor specific output characteristics. Nevertheless none of the simulators considers the impact of different materials on the measurements. This limitation is most likely caused by the fact, that the methods used to determine the distances only provide information on the distance of objects, but not on their type.

3 Enhanced Simulation of Distance Sensors

Simulation of a distance sensor is a two step process. The first mandatory step consists of the calculation of the first intersection of each ray emitted by the sensor. Besides information on the length of the ray, this step may produce additional information on the kind of object hit by the ray. The data produced by this step can be considered as readings from a perfect sensor.

In a second, optional, step, the ray length is post-processed. Independent of the object hit by a ray, effects like noise or other sensor specific output characteristics can be simulated as a function of the length calculated before. If additional

information on the object is present, further object or material specific processing of the sensor output can be calculated.

This leads to three distinct levels of accuracy for the sensor simulation:

1. Simulation of a *perfect* sensor,
2. Simulation of a sensor with specific output characteristics,
3. Simulation of a sensor considering material dependent behavior.

The highest level of accuracy can only be simulated, if information on the objects hit by the ray are provided by the first step.

In the following subsections two methods used in many simulators for the calculation of the ray length are discussed. Further on it is investigated, how these methods can be extended to provide further information on the objects hit by the rays. Both methods have been implemented as part of the Multi-Robot-Simulation-Framework (**MuRoSimF**) by the authors of this paper. Later on the methods will be compared for their respective performance concerning the different levels of accuracy.

MuRoSimF provides methods for the simulation of wheeled (e.g. [1]) and legged (e.g. [16]) locomotion as well as for different sensors of autonomous mobile robots. It allows an unrestricted combination of these simulation methods to create simulations with different levels of accuracy and abstraction.

3.1 Depth Buffer Based Method for Ray Simulation

This method is based on rendering the simulated scene from the sensor's point of view using OpenGL¹. After rendering the scene, the depth-buffer used for hidden surface removal by OpenGL is read back to process memory for calculation of the length of each view-ray. Inverting the steps of the perspective projection manipulations of the depth coordinate described in [17] leads to

$$z = -\frac{f \cdot n}{\tilde{z} \cdot (f - n) - f} \quad (1)$$

with z being the orthogonal distance of a point with depth coordinate \tilde{z} from the viewing plane. The parameters f and n denote the near and far clipping plane defining the viewing volume used for the rendering process (see Figure II for details). To calculate the length l of a ray, the direction α of this ray must be taken into account leading to

$$l = \frac{z}{\cos \alpha}. \quad (2)$$

If additional information on the object hit by a ray is desired, this information can be encoded into the image during rendering. To do this, an id describing the surface of each object is transformed into an RGB-triplet which is used as color during rendering. Rendering itself is done without lighting, shading or blending

¹ Note that the same method can be used using other 3D rendering systems, as long as they provide access to the depth buffer for the CPU.

effects, so that the color's RGB-values are preserved during the rasterization process. To obtain object information for the rays, the frame-buffer is read back to process memory and evaluated for each pixel leading to surface information for the object hit by each of the rays.

As discussed in [12] the view-rays have an equal distribution in the image plane. This contradicts the equal angular distribution usually found in laser range finders. Different strategies to cope with this problem are discussed in Section 3.3.

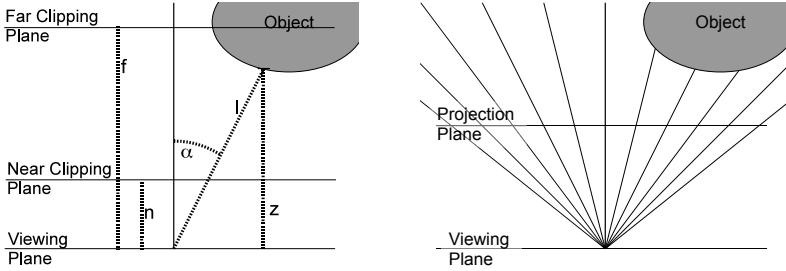


Fig. 1. Left: Length of a ray. Right: distribution of rays (adapted from [1]).

3.2 Ray Intersection Based Method for Ray Simulation

The second method for calculation of the rays is based on the explicit calculation of ray-object intersections. A sensor can be defined by an arbitrary number of rays, each with an individual direction. No limits are imposed on the aperture angle or the distribution of the rays.

To avoid intersecting each ray of the sensor with each object of the scene, the scene is decomposed into compounds containing the objects of the scene. For each compound and each individual object an axis-aligned bounding-box is calculated (and if the object is mobile, re-calculated for each time-step of the simulation). Ray-intersection is based on this hierarchy of bounding boxes, thus allowing to discard many objects before calculating time consuming tests on object-geometry-level. The whole hierarchy is provided by the collision-detection module which is provided as part of the dynamics-simulation of MuRoSimF [16]. Simulation of all distance sensors uses the same hierarchy, thus avoiding multiple re-calculations of the hierarchy.

As the bounding-box-hierarchy is aligned to the world-coordinate systems, the rays have to be transformed to the same coordinate system before intersection tests can be performed. To further speed up the simulation, rays are aggregated to sets of rays, which fit into a bounding box, so that further objects of the scene can be discarded for some of the sets.

3.3 Simulation of Different Distance Sensors

After calculation of the ray length (and optionally the material hit by the individual rays) additional post-processing of the data may be applied to

simulate different kinds of sensors. In this section several of the possible steps are discussed.

Error modeling. Several possibilities for modeling the sensor's errors exist and can be applied after the ray-calculation on single-ray whole-scan level.

- On the level of single rays, an error can be calculated as a function of the length of the ray. This function may contain a randomized component to simulate noise. This function can be chosen individually for each simulated sensor.
- If additional information on the material hit by the individual ray is present, the function for error modeling can be chosen according to the material for each ray.
- On the level of the whole scan, effects like crosstalk between neighboring rays can be considered.

Handling errors of ray-direction. When using the depth-buffer based methods, the directions of the calculated rays are limited to those directions defined by the viewing geometry. These directions are not necessarily the same as the directions of the rays of simulated sensor. Especially laser scanners often have an even-angular distribution of the rays. Several options are available to cope with this problem:

- In case the error is small (enough for the desired application), the problem may be ignored.
- Interpolation can be applied to approximate the length of the rays of the simulated sensor.
- To reduce the angular error, a higher resolution for rendering can be chosen.
- If the aperture angle of the sensor is high, it may be necessary to use more than one rendering pass using different viewing directions to reduce the error.

A different kind of error is caused by the motion of the sensor during measurement. If the method used for ray simulation can be calculated efficiently for parts of a scan (optimally for single rays), it can be interleaved with the motion simulation. By this, distortions of the scan caused by the motion of the robot can be simulated.

Aggregation of rays. For distance sensors which produce one single value, e. g. ultrasound or infrared distance sensors, it nevertheless may be of interest to calculate intersections of several rays with the scene. This is especially interesting if the sensor has a significant cone characteristic, as it is the case with many ultrasound-sensors. The easiest way to simulate a sensor producing a single value is to calculate the minimum distance of all calculated rays. If further information on the sensor's characteristic is known, the directions of the rays may be used as additional argument for the calculation of the sensor output value.

4 Results

4.1 Applications

Both simulation methods have been implemented as part of the Multi-Robot-Simulation-Framework (**MuRoSimF**). With a robot simulation based on **MuRoSimF** it is possible to provide different simulation methods for the same purpose (like simulation of a robot's motion or a distance sensor). These methods can be exchanged transparently and independently of the other elements of the simulation and be adapted well to very different purposes of robot simulation [12].

Several simulations for robots equipped with laser-scanners have been created using **MuRoSimF**. These include a simulation of simplified virtual robots for educational purposes and a simulation of a search-and-rescue vehicle (see Figure 2).

The simulation of the vehicle is based on a real robot used for urban search and rescue [18] which is equipped with two laser-scanners, among other sensors. To allow for a transparent integration of the sensor simulation with the robot-control-software, a simulation specific module was added which emulates the SCIP (see [19]) protocol used by the sensors of the real robot. For each laser-scanner of each simulated robot it is possible to choose the simulation method, resolution and rate independently in the configuration of the simulation. Besides selecting a resolution appropriate for a specific simulated experiment, it is possible to select the method providing the best performance on the respective computer. This selection can be made by running the simulation twice for a short time with either method and comparing the performance, before starting the main experiment. The simulation has been used for the development of sensing and mapping algorithms as well as for high-level behaviors of the vehicle which could be transferred successfully to the real vehicle.

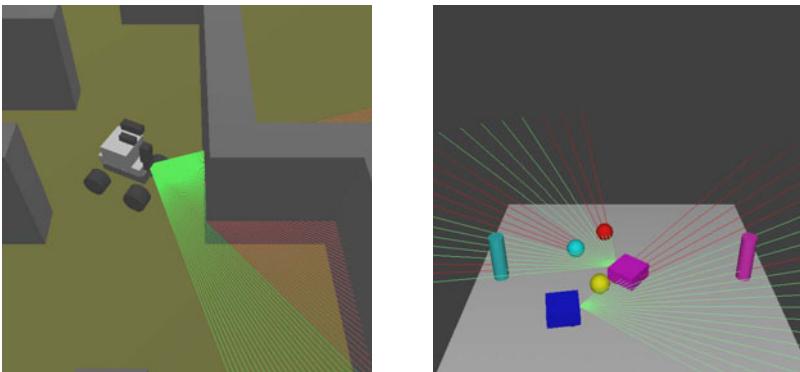


Fig. 2. Left: Simulation of a search-and-rescue robot. Right: Educational simulation of two virtual robots. Note that rays are rendered green as long as they did not touch an obstacle and red afterwards.

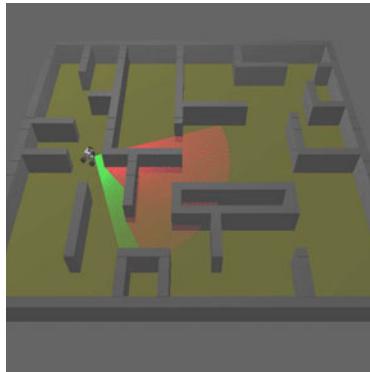


Fig. 3. The scenario used for performance evaluation

Table 2. Computers used in experiments

	A	B	Computer
	A	B	C
Operating System	MacOS X 10.6.3 (32 Bit)	Ubuntu Lucid Lynx (64 Bit)	Windows Vista Business SP1 (32 Bit)
CPU	Intel Core 2 Duo T8300	Intel Core 2 Duo P9500	Intel Core 2 Duo P8600
Clock	2.4 GHz	2.53 GHz	2.4 GHz
RAM	2 GByte	4 GByte	4 GByte
Graphics Hardware	Intel GMA X3100 (chipset)	NVIDIA Quadro NVS 160M	NVIDIA Quadro NVS 160M

4.2 Performance

The performance of the simulation methods has been measured with the search-and-rescue vehicle simulation situating the robot in a typical maze-like environment (see Figure 3). For the measurements only one laser-scanner of the robot was activated. In this scenario the time required to calculate one sweep of the laser scanner has been measured for several resolutions using both methods with and without determination of the material (see Table 3). For each of these setups a new run of the simulation was started and 300 the computation time for the sensor simulation was measured for 300 times. During all measurements the aperture angle of the sensor was 90 degrees (with the obvious exception of the 1x1-resolution). Measurements were taken on three different laptop-computers (see Table 2).

The main results from these measurements are:

- The ray-intersection-methods have smaller computation times at small resolutions, while the depth-buffer-methods have a better performance at high resolutions.
- Additional determination of the material hit by the respective rays has nearly no impact on the performance of the ray-intersection based methods. On the

Table 3. Average computation time in ms for one scan at different resolutions for different simulation modes (R = ray-intersection, ZB = z-buffer, ...wM = ...with material determination)

Resolution	Computer A				Computer B				Computer C			
	R	RwM	ZB	ZBwM	R	RwM	ZB	ZBwM	R	RwM	ZB	ZBwM
1x1	0.021	0.023	6.3	6.33	0.048	0.05	3.87	23.2	0.03	0.033	1.57	1.64
1x3	0.017	0.018	6.2	6.28	0.041	0.041	4.17	23.3	0.027	0.031	1.52	1.61
1x10	0.066	0.068	6.14	6.17	0.19	0.193	3.89	4.19	0.116	0.127	1.57	1.58
1x30	0.165	0.165	6.22	6.27	0.468	0.569	3.96	3.98	0.256	0.272	1.52	1.6
1x100	0.467	0.474	6.18	6.24	1.35	1.31	4.11	3.88	0.758	0.782	1.55	1.58
1x300	1.33	1.35	6.12	6.27	4.03	4.03	3.88	3.94	2.17	2.19	1.5	1.7
1x1000	4.31	4.37	6.12	6.35	12.0	11.2	4.15	4.13	8.15	7.06	1.53	1.64
35x35	5.52	5.59	6.3	7.11	13.7	12.6	3.98	4.1	8.07	8.64	1.62	1.78
50x50	12.1	12.1	6.78	7.37	19.7	19.0	4.19	4.3	20.8	21.2	1.6	1.81
70x70	23.6	24.0	6.9	7.54	31.6	31.6	4.42	4.62	47.8	47.6	1.74	1.99
100x100	47.6	48.2	7.05	7.67	52.8	52.2	4.98	5.51	68.3	67.7	2.11	2.47
140x140	93.8	94.4	7.4	8.33	91.7	92.2	5.86	6.67	126	129	2.5	2.91
200x200	192	195	7.91	9.53	188	190	7.98	9.65	259	263	3.16	4.24

other hand, there is a visible impact at higher resolutions when using the depth-buffer-methods. Parts of this impact can be explained by the additional effort required to read back the frame-buffer (see discussion below). In some cases a method with additional determination of the material took less time than the method without the addition. This is most likely caused by disturbances due to background activities of the respective computer.

- The computation time of the ray-intersection methods scales quite well with the number of rays. Compared to this, the depth-buffer-methods start at a higher basic computation time, which does not change that much with rising resolutions. This can be explained by the fact, that independently of the actual resolution always the whole scene has to be rendered from the sensor's point of view.
- The break even of the computation time of the different methods is highly dependent on the current system.
- Two measurements on system B led to extreme average computation times (ZBwM at 1x1 and 1x3). Further investigations showed, that this effect appeared randomly for several resolutions on system B but not on the other systems. It is not clear what caused this effect.

Another result which came clear during these experiments is the strong impact of reading-back the frame- and depth-buffer on the overall performance of the depth-buffer based methods. Measurements of the time required to read back these buffers are given in Tab. 4. Interestingly reading back the frame buffer takes less time than reading the depth-buffer. To further investigate this effect, the reading order was inverted in an experiment on system A, leading to an exchange of the reading times.

Table 4. Time in ms required for read back of depth and framebuffer

Resolution	Computer A		Computer B		Computer C	
	depth-buffer	frame-buffer	depth-buffer	frame-buffer	depth-buffer	frame-buffer
1x1	1.24	0.075	0.076	0.036	0.312	0.086
1x3	1.21	0.075	0.078	0.036	0.313	0.093
1x10	1.18	0.077	0.169	0.068	0.246	0.088
1x30	1.20	0.079	0.116	0.056	0.289	0.087
1x100	1.21	0.090	0.130	0.060	0.271	0.089
1x300	1.17	0.111	0.135	0.056	0.311	0.096
1x1000	1.08	0.182	0.162	0.085	0.296	0.104
35x35	1.30	0.785	0.166	0.080	0.353	0.186
50x50	1.71	0.621	0.226	0.100	0.288	0.200
70x70	1.76	0.640	0.295	0.126	0.308	0.247
100x100	1.78	0.625	0.491	0.172	0.504	0.298
140x140	1.90	0.802	0.775	0.272	0.608	0.372
200x200	2.02	1.220	1.470	0.540	0.887	0.767

4.3 Discussion

The performance results from the previous section suggest, that it is highly desirable to provide both methods for ray simulation. Depending on the computer the simulation is executed on, this will allow to choose the better performing method.

If the methods are to be truly exchangeable within a simulation, it is necessary, that equal interfaces to the provided data are available. Any post-processing for sensor error simulation can be done independently of the calculation of the rays.

Independent of the performance other considerations can be made to choose either one of the methods: The depth-buffer based method has a fixed distribution of the view rays. Thus it can perform best, if either this distribution matches the simulated sensor (e.g. ToF-cameras), or is of no interest (e.g. if the rays will be aggregated later on for ultrasound cones). For sensors with different distributions of the rays (e.g. 2D or 3D laser scanners), additional processing steps are required as discussed in Section 3.3. Due to the high basic computation time of the method, it cannot be used to simulate single rays, as it is necessary when considering the motion of the sensor during measurement.

The ray-intersection based method imposes no limitations on the distribution of the rays. Further on the method's computation time scales well with the number of rays. It is feasible to even simulate single rays at a high rate, thus allowing the simulation of effects caused by the motion of the robot while sensing.

Both methods can be used for the additional calculation of the material hit by the individual rays. Only at higher resolutions, there is a visible impact on the computation time of the depth-buffer based method. Nevertheless the method performs much better than the ray-based method at these resolutions.

5 Conclusions and Outlook

Two different methods for the calculation of the distances measured by distance sensors were discussed in this paper, namely reading back the depth buffer from

3D renderings and the calculation of ray-object-intersections. Either method can be extended to determine the material hit by individual rays of the sensor. This allows for the simulation of sensor errors on different levels of accuracy which is not possible in current general purpose robot simulators commonly used in the autonomous robotics community.

Both methods have been implemented as part of the Multi-Robot-Simulation-Framework (**MuRoSimF**) and can be exchanged transparently within simulations. Extensive measurements of the performance of either method on different computer systems were made. The results of these measurements suggest that it is very useful to provide either method fully exchangeable within the simulation to allow the adaption of the simulation to different computers to provide optimal performance. Further on different advantages and drawbacks of each method were discussed to give an aid in choosing a method depending on the requirements of the individual simulation.

With the existing implementations of both methods the complete infrastructure for the simulation of distance sensors is provided. As a next step the authors are planning to determine concrete functions to model errors of real world sensors to be evaluated with the robots of the authors' group, thus providing a validated simulation of these devices. Identification of parameterized error models can be based on comparing the outputs of real and simulated sensors in experiments and applying optimization methods like least square fitting.

The good scalability of the ray intersection based simulation method opens up another opportunity: By simulating few or even single rays of a whole sensor sweep for every time step of the simulation of the robot's motion, effects caused by the motion of the sensor can be simulated.

Acknowledgment. Parts of this research have been supported by the German Research Foundation (DFG) within the Research Training Group 1362 "Cooperative, adaptive and responsive monitoring in mixed mode environments".

References

1. Friedmann, M., Petersen, K., von Stryk, O.: Simulation of Multi-Robot Teams with Flexible Level of Detail. In: Carpin, S., Noda, I., Pagello, E., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2008. LNCS (LNAI), vol. 5325, pp. 29–40. Springer, Heidelberg (2008)
2. Friedmann, M.: Simulation of Autonomous Robot Teams With Adaptable Levels of Abstraction. PhD thesis, Technische Universität Darmstadt (November 30, 2009)
3. Kawata, H., Ohya, A., Yuta, S., Santosh, W., Mori, T.: Development of ultra-small lightweight optical range sensor system. In: Proc. of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS (2005)
4. Koenig, N., Howard, A.: Design and Use Paradigms of Gazebo, an Open-Source Multi-Robot Simulator. In: Proc. of the 2004 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, IROS (2004)
5. Jackson, J.: Microsoft Robotic Studio: A Technical Introduction. Robotics and Automation Magazine 14(4), 82–87 (2007)

6. Hugues, L., Bredeche, N.: Simbad: an Autonomous Robot Simulation Package for Education and Research. In: Proc. of the 2006 Intl. Conf. on the Simulation of Adaptive Behavior (SAB), Rome, Italy (2006)
7. Simbad website (2009), <http://simbad.sourceforge.net/>
8. Laue, T., Spiess, K., Röfer, T.: SimRobot - A General Physical Robot Simulator and Its Application in RoboCup. In: Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y. (eds.) RoboCup 2005. LNCS (LNAI), vol. 4020, pp. 173–183. Springer, Heidelberg (2006)
9. Röfer, T., Laue, T., Burchardt, A., Damrose, E., Gillmann, K., Graf, C., de Haas, T.J., Härtl, A., Rieskamp, A., Schreck, A., Worch, J.-H.: B-Human Team Report and Code Release 2008. Technical report (2008)
10. Gerkey, B.P., Vaughan, R.T., Howard, A.: The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In: Proc. of the 2003 Intl. Conf. on Advanced Robotics (ICAR), Coimbra, Portugal, pp. 317–323 (June 30-July 3 2003)
11. Vaughan, R.: Massively multi-robot simulation in stage. Swarm Intelligence 2, 189–208 (2008)
12. Carpin, S., Lewis, M., Wang, J., Balakirsky, S., Scrapper, C.: USARSim: a robot simulator for research and education. In: Proc. of the 2007 IEEE Intl. Conf. on Robotics and Automation, ICRA (2007)
13. Wang, J., Balakirsky, S.: UARSSim V3.1.3 - A Game-based Simulation of mobile robots. Technical report, NIST (2008)
14. Michel, O.: Cyberbotics Ltd. - Webots(TM): Professional Mobile Robot Simulation. Intl. Journal of Advanced Robotic Systems 1(1), 39–42 (2004)
15. Webots user guide 6.2.3. Technical report, Cyberbotics, Ltd. (2010)
16. Friedmann, M., Petersen, K., von Stryk, O.: Adequate Motion Simulation and Collision Detection for Soccer Playing Humanoid Robots. Robotics and Autonomous Systems 57, 786–795 (2009)
17. Segal, M., Akeley, K.: The OpenGL Graphics System: A Specification (Version 2.0 - October 22, 2004). Technical report, Silicon Graphics, Inc. (2004)
18. Andriluka, M., Kohlbrecher, S., Meyer, J., Petersen, K., Schnitzspan, P., von Stryk, O.: RoboCupRescue 2010 - Robot League Team Hector Darmstadt (Germany). Technical report, Technische Universität Darmstadt (2010)
19. URG Series Communication Protocol Specification SCIP-Version 2.0. Technical report, Hokuyo Automatic co., Ltd. (2006)

High Fidelity Sensor Simulations for the Virtual Autonomous Navigation Environment

Chris Goodin, Phillip J. Durst, Burhman Gates,
Chris Cummins, and Jody Priddy

U.S.Army Engineer Research and Development Center

Abstract. The Virtual Autonomous Navigation Environment (VANE) is a high-fidelity simulation environment for ground robotics. Physics-based realism is the primary goal of the VANE. The VANE simulation incorporates realistic lighting, vehicle-terrain interaction, environmental attributions, and sensors. The sensor models, including camera, laser ranging, and GPS, are the focus of this work. These sensor models were designed to incorporate both internal (electronic) and external (environment) noise in order to produce a sensor output that closely matches that produced in real-world environments. This sensor output will allow roboticists to use simulation further into the development and debugging process before exposing robots to field conditions.

1 Introduction

The Virtual Autonomous Navigation Environment (VANE) is a high-fidelity simulation environment that is being developed by the U.S. Army Engineer Research and Development Center to facilitate virtual testing and algorithm development for ground robotics. The VANE incorporates physics-based vehicle dynamics, radiation transfer, environmental attributions, and sensor models into a closed-loop simulation.

An overview of robotics simulations was recently completed by [1], who found that most commercial simulations were of medium to low fidelity. Lower fidelity, real-time simulation environments such as Real/Player, Microsoft Robotics Studio, or the National Institute of Standard's Unified System for Automation and Robot Simulation (USARSim) [2] have proven useful in aiding the development and debugging of robotic algorithms. However, these environments often present a sanitized, noiseless environment that may not expose certain algorithmic problems that are later discovered in field tests. The VANE is intended to provide an unsanitized simulation environment that incorporates the noise and unpredictability that is often encountered in the real world, both in the environment and the sensors.

The VANE simulation concept is depicted in Fig. 1, which shows the critical components of the VANE. The first component, shown as the large yellow block, is the VANE Run Time Interface (RTI). The VANE RTI is an event driven simulation environment that manages the passage of simulation time. Because

the VANE's focus is on realism and high-fidelity, it is not intended to be a "real time" simulation. Instead, care was taken to create an environment with flexible time management that allows each physics-based model to operate at its characteristic frequency.

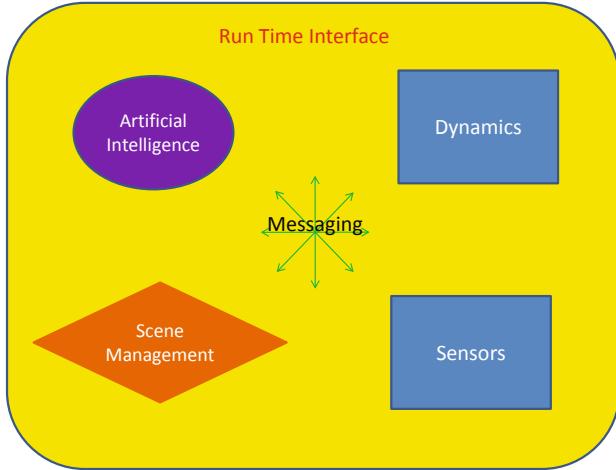


Fig. 1. A visualization of the VANE simulation architecture. The yellow rounded box containing the other boxes is the RTI that provides a general framework for setting up the simulation. The details of the environment are controlled by the Scene Manager (orange diamond), and the physical models are represented by the blue rectangles. The messaging, depicted by the green arrows, is a separate but integral part of the RTI that communicates between the components of the simulation.

The second part of the VANE simulation is the messaging, depicted by the green arrows in Fig. 1. The messaging defines protocols for sending messages of a particular type, including text, sensor data, or execution instructions to the other models. The messaging is independent of the RTI both for the sake of simplicity and flexibility. While the details of the physical models (the blue boxes in Fig. 1) may change from one simulation to the next, the message types will remain the same. Furthermore, the messaging system allows information to be passed within a processor, between processors, or over a network.

The orange diamond in Fig. 1 is the scene manager, which controls the virtual representation of the physical environment being simulated. Each of the physical models (blue boxes) may require a unique representation of the virtual environment, and for a dynamic simulation the physical environment may change in an unpredictable way as the simulation proceeds. Furthermore, the models may have different requirements for the level of fidelity used to represent the environment, and the required level of detail may itself be dynamic. For example, a typical Laser Detection and Ranging (LADAR) sensor may require the scene to be represented at the millimeter scale to a range of 40 meters but

a centimeter scale beyond. This varying level of detail would require different representations of the scene for each sensor (and implicitly each robot) in the simulation. The scene manager tracks the dynamic environment and makes the necessary updates to the scene during the simulation.

The purple oval in Fig. 11 represents the Artificial Intelligence (AI) in the simulation. In VANE simulations, “AI” primarily refers to the Autonomous Driver (ADr) on the robot being tested. However, “AI” may also refer to any other intelligence algorithms that control some aspect of the simulation. For example, for a simulation of riotous behavior in groups of people within the VANE, the algorithms controlling the mob’s behavior would be included in the “AI” box. As mentioned above, the simulation is not real time, meaning an AI interacting with these scenes must be time managed.

The final components of the VANE are the blue boxes in Fig. 11, which represent the physical models of the VANE. The models depicted are “Dynamics” and “Sensors.” The Dynamics models include the Vehicle-Terrain Interface (VTI), which models the forces on wheeled vehicles based on ground and vehicle properties, as well as the forces on any other dynamic objects in the scene. The Sensors section includes LADAR, cameras, Global Positioning System (GPS), accelerometers, and gyroscopes.

The three main types of sensor models currently used in the VANE are GPS, inertial sensors (gyroscope, accelerometer), and electromagnetic sensors like LADAR and cameras. The electromagnetic sensors were introduced in a previous work [3]. Both the camera and LADAR sensors use a raycaster, called the QuickCaster (QC), to input the appropriate radiance values into the sensor models. The QC therefore incorporates the attributions and fidelity of the environment with detailed ray casting as a first step in creating realistic sensor output.

2 The Quick Caster

The QC is a parallelized ray-casting code that is optimized to render images of very detailed scenes. Scene geometries are represented in the QC as meshes of triangular facets. A typical desktop computer will only be able to render scenes with a few million facets, perhaps up to ten million, but the QC is capable of rendering scenes with over 300 million facets. This allows the scene geometries to be very detailed and encompass a large physical extent in the simulation.

The QC is run on a Cray XT4, called Jade, which is available through the DoD Supercomputing Resource Center (DSRC). Jade has 2,146 2.1-GHz AMD Opteron 64-bit quad-core processors with 8 GBytes of dedicated memory each. Most parallel ray casters are parallelized on rays, pixels, or parallelized by object or spatial location. However, in order to take advantage of the large number of processors available through the DSRC while also ensuring load balancing, the QC was parallelized by facets. The facets comprising a scene are distributed round-robin style across all the available processors. Each processor then performs the ray-casting routines (triangle intersections) on its facets. The results are combined at the end

to determine which facets are in view, which are blocked, and which are closest. After that, the QC can be used in different ways to provide input radiance data for the camera and LADAR sensors and ranges for the GPS.

Input for the Charge Couple Device (CCD) camera model comes from the QC as an array of radiance values based on each facet's reflective properties and the solar loading. The corresponding distances for the rays are also passed to the camera model from the QC. These array values are used by the CCD model to create an ideal image. Similar data values are also passed for the LADAR model but rays are only traced from the sensor into the scene. The pinhole camera model is used in a novel way to account for the diverging beam in the LADAR model.

3 Generic CCD Model

The CCD model used in the VANE is based on ray optics. Conceptually, the model is quite similar to that proposed by Subbarao and Nikzad [4], which models the camera as a series of algebraic and convolution operations on an initial ideal image. These operations include filtering, vignetting, spectral response, exposure time, and lens optics. Subbarao and Nikzad propose a model using the Point Spread Function (PSF) to map the rays of the ideal image to the real image, while the CCD model in the VANE uses an explicit ray tracing based on the third-order Seidel aberrations as described in [5].

An example of a CCD model image is presented in the right part of Fig. 2. The corresponding ideal image is shown in the left part of the same figure. There are several important features in the model image which may have an effect on machine vision or image processing. These include pixel saturation, color shifts due to filter and CCD response functions, spherical aberration, and intensity vignetting. The CCD model used in the VANE accounts for all these effects, as shown in Fig. 2



Fig. 2. A comparison of an ideal (left) and CCD model (right) image. Effects reproduced in the CCD model include vignetting and aberrations.

At the present time, the CCD model is notional and has not been validated against a particular camera. However, the model is highly parameterized, allowing several different types of CCD cameras to be simulated with the same model. More specific mathematical details of the CCD sensor model can be found in [3].

4 Scanning LADAR

The model presented here is roughly representative of the Sick LMS-291, for which mathematical models have previously been developed by [6] and [7]. Our model differs from these in that we use ray-tracing through an ideal-image plane to define the laser pulse shape and divergence by using a virtual focal length and detector dimensions.

It was shown in our previous work [3] that accounting for beam divergence effects in LADAR simulations produces more realistic results than simple pencil-ray tracing or z-buffer LADAR. Beam divergence for the SICK LMS-291 line scanning LADAR has been modeled in VANE by tracing between 100 and 10,000 rays for each pulse of the laser. As we previously determined, the necessary number of rays (N) is given by

$$N = \frac{2(r + R_{max} \tan(\frac{\Theta}{2}))}{x} \quad (1)$$

where x is the scale of the spatial variation of the target, r is the aperture radius of the receiver, R_{max} is the maximum range of the LADAR, and Θ is the divergence of the beam.

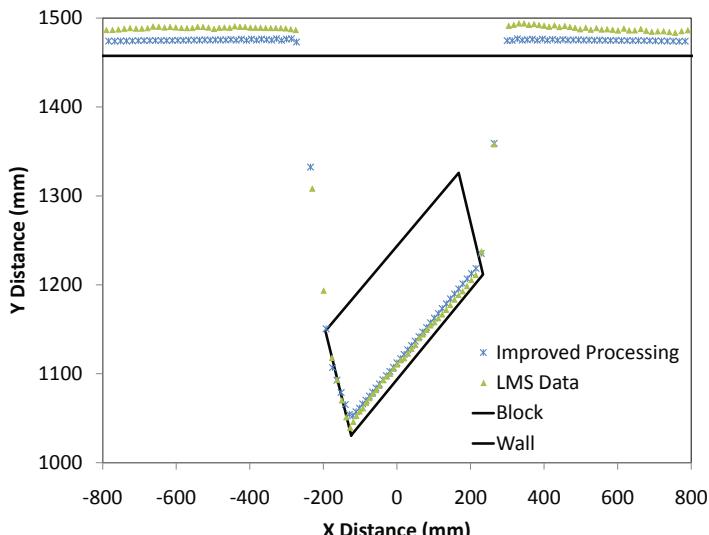


Fig. 3. A schematic of a simple experiment used to determine the signal processing algorithm for the SICK LMS-291. In this plot, the real data are represented by the green triangles, and the improved processing algorithm are the blue stars. The truth data for the wall and block are represented by the solid black lines.

The rays cover the spatial extent of the beam spot, which can be close to one meter diameter at the maximum operating distance of the LADAR, which is about 80 meters. The intensity distribution of the beam spot is modeled as a Gaussian beam profile. Each ray carries a radiant intensity corresponding to its position in the profile, and the reflected intensity is calculated based on the properties of the materials in the environment. All the return intensities are time stamped and summed to produce a return pulse.

Since our previous work, we found that the processing of this signal can have a significant impact on the distance returned. We have used a trial-and-error method with a real system in a set of controlled experiments to determine as closely as possible the correct signal processing to yield the most realistic results. A schematic of the experiment is shown in Fig. 3. A rectangular block was placed in front of a wall and scanned with the SICK LMS-291 LADAR. The key feature of the resulting data are the “floating” points or mixed pixels between the block and the wall. The figure shows how the data from the model with improved signal processing closely match the real data.

5 GPS Sensor Model

The GPS model finds the receiver’s position using a geo-specific, pseudorange method that calculates position using the distances between the receiver and the GPS space vehicles (SVs). The model operates in three steps. First, it uses the QC to find the true ranges, R , between the receiver and the SVs. The initial SV positions are specific to the site at which the receiver is being simulated, and they are taken from the Scripps Orbit and Permanent Array Center (SOPAC) archive [8]. At each time step during the simulation, the SV positions are updated using Kepler’s equations [9] and the ephemeris (orbit parameter) data contained in the SV broadcasted RINEX file [10]. Once the SV positions have been determined, the QC is used to check whether or not the SV is blocked by objects in the scene and to check for multipath effects.

After finding the unblocked SVs and their distances from the receiver, the model takes the true range (or multipath range) and alters it using several error functions. The effects of the Earth’s atmosphere are modeled using three different error functions. Ionosphere range errors (dR_{iono}) are modeled using a Monzenbruck mapping function [11] with the constants derived from the standard range error at zenith. Likewise, the effects of the upper Troposphere ($dR_{trop,dry}$) are modeled using a Neill mapping function [12], the detailed derivation of which can be found in [13]. The range errors caused by the lower atmosphere ($dR_{trop,wet}$), or ‘hydrostatic Troposphere’, are modeled using the equations derived by Park and Ha in [14].

In addition to atmospheric effects, the model also accounts for range errors due to SV onboard ephemeris errors and errors in the GPS system atomic clock. As the SVs orbit the Earth, errors in their internal calculations can degrade the SV position reported to the GPS receiver. Errors in the reported SV position

in turn create errors in the SV-receiver range. These range errors are modeled using an equation derived from the work of Hwang and Hwang and Parkinson [15] [16]

$$dR_{SVephemeris} = \sqrt{(1 + \frac{1}{2} \sin \theta)(10^{\frac{2R_{ref}}{R_0}}) + c^2(10^{-18}) + 0.918} \quad (2)$$

where θ is the SV's latitude, R_{ref} is a reference orbit radius of 26550000 meters, R_0 is the SV orbit radius, and c is the speed of light in a vacuum.

Once all of the error effects have been accounted for, a new distance between the receiver and SV is calculated

$$R' = R + dR_{iono} + dR_{trop,wet} + dR_{trop,dry} + dR_{SVephemeris} \quad (3)$$

The new ranges, R' , are then fed into a generic trilateration algorithm, which computes the receiver's position. A detailed discussion on how GPS receivers find position using SV-receiver ranges can be found in [16]. By using this method, the receiver's returned position will suffer from errors caused by poor SV geometry, an effect known as dilution of precision (DOP). Fig. 4 shows the effects of each error source on the GPS receiver output position during a test run around a circular track.

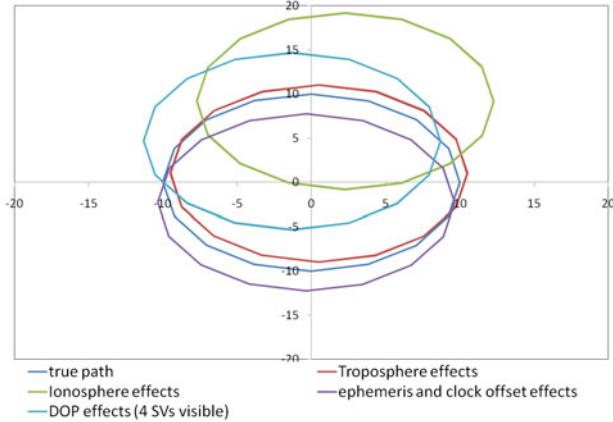


Fig. 4. Output GPS receiver position for each individual error source

6 Results

Several tests of VANE capabilities were conducted to demonstrate the capabilities of the VANE. The three key pieces of a simulation are

1. The environment. This includes the geometry of the environment, the attributions, and the trafficability and soil parameters affecting navigation.

Environments can either be geo-specific (exact representation of a particular latitude and longitude) or geo-typical (a non-specific representation of a type of place, like a forest). Environments of both types were constructed to test the VANE.

2. The sensor package. Physics-based sensors in the VANE include CCD, LADAR, GPS, gyroscope, and accelerometer. Many sensor packages on existing robotic platforms can be reproduced using only these sensors. Other sensors, such as vehicle sensors like wheel encoders, are currently being developed
3. The ADr. We tested the VANE with two simple software ADr for truck-sized robots. The first is a simple waypoint follower that issues driving commands with no regard for sensor information. The other is a slightly more advanced path planner that uses a two dimensional “go/no-go” map generated from LADAR data to navigate through a scene.

Figures 5 shows an example of a geo-typical scene that is roughly representative of a Middle-Eastern city. An initial test using this Middle-Eastern scene was conducted using the simple path planning ADr. Partial results from this test are shown in Fig. 6, which shows the map that was generated by a robot driving through the streets with two panning SICK LADAR mounted on the front and rear of the robot. The map was generated from the resulting (x,y,z) point cloud. The green line in this figure is drawn from the robot’s current position to the desired waypoint.

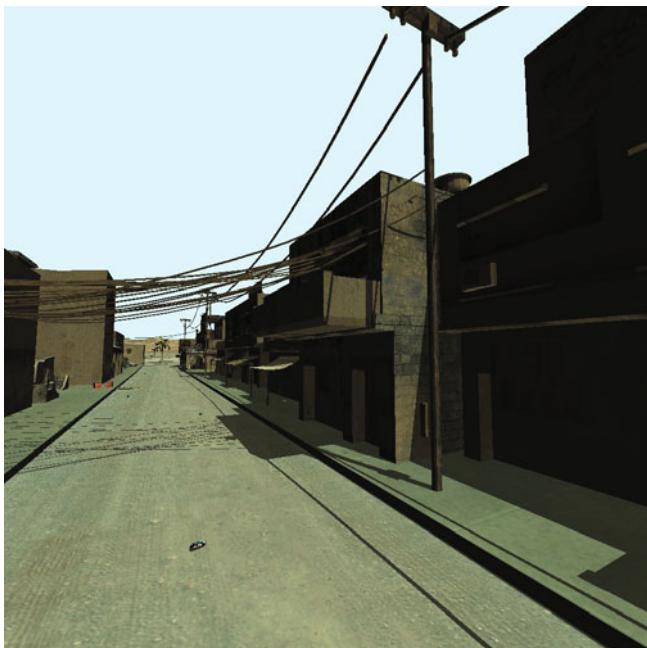


Fig. 5. A street-level view of the geo-typical Middle-Eastern city

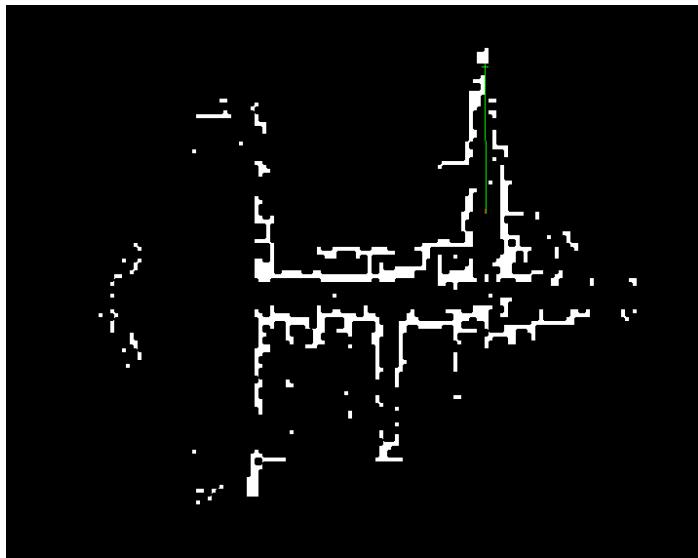


Fig. 6. A “go/no-go” map generated from a robot with a single panning SICK LADAR driving through the city

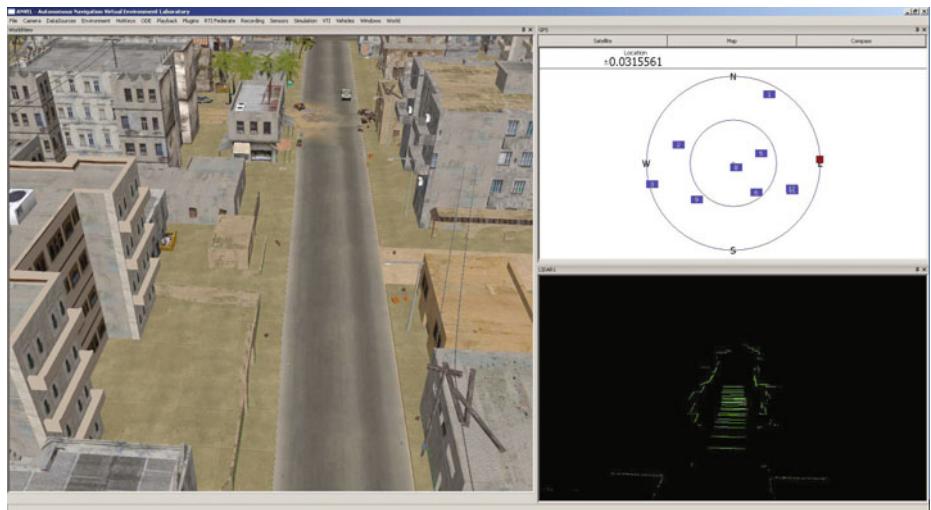


Fig. 7. VANE simulation replay in the ANVEL. The left part shows the simulation environment. Te upper right is the GPS model, and the lower right is the LADAR point cloud.

Post processing of VANE simulations, including replay, debugging, mission planning, and interactive data viewing, are accomplished with the Autonomous Navigation Virtual Environment Laboratory (ANVEL) [17], which can be used

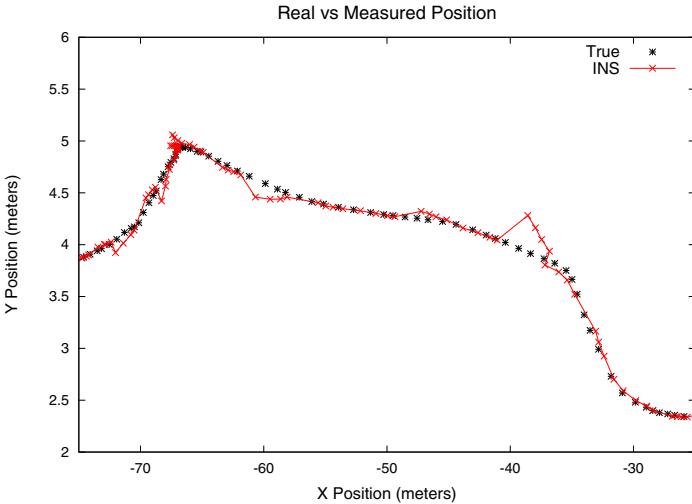


Fig. 8. Comparison of the true and perceived path as calculated by Inertial Navigation System (INS) using IMU and GPS data. The points are spaced at $\frac{1}{4}$ of a second in time. The GPS updates occurred at 1 Hz, and the IMU was used for localization between GPS updates.

to view LADAR, GPS, and image data and replay the path of the vehicle and other objects in the simulation. A screenshot of ANVEL in the Middle-Eastern city is shown in Fig. 7, which shows the world model on the left, the GPS model data on the top right, and the LADAR point cloud on the bottom right. Note that the LADAR point cloud includes intrinsic sensor noise, environmental noise, and registration error caused by GPS error.

Several initial tests of the VANE simulation process have been completed and demonstrate the results of incorporating sensor noise and realistic intrinsics in the simulation. One result of using realistic GPS and Inertial Measurement Unit (IMU) models is shown in Fig. 8, which shows the vehicles real path compared to its perceived path from IMU and GPS data. Further tests in this environment are ongoing.

7 Future Work and Conclusions

The VANE is still in the initial stages of development. Future work will focus on the simulation architecture, environmental attributions, developing new sensors, and verification and validation. In the area of attributions, we are specifically focused on adding directional spectral reflectance properties to the objects in the environment by incorporating a Bidirectional Reflectance Distribution Function to the material attributions of each object. Models for the Velodyne and Hokuyo LADAR are also currently under development, and our verification and validation efforts will focus initially on validation of these models.

In conclusion, we developed a high-fidelity physics-based simulation environment to aid in the development and testing of autonomous navigation algorithms for unmanned ground vehicles. Initial tests indicate that the environment provides a virtual proving ground for autonomous vehicles and that the sensor data used to feed the robot perception closely resembles real sensor data. Potential development partners for the VANE should contact the authors.

Disclaimer

Permission to publish was granted by Director, Geotechnical and Structures Laboratory.

References

1. Craighead, J., Murphy, R., Burke, J., Goldiez, B.: A survey of commercial & open source unmanned vehicle simulators. In: 2007 IEEE International Conference on Robotics and Automation, pp. 852–857 (2007)
2. Balakirsky, S., Scrapper, C., Carpin, S., Lewis, M.: UsarSim: providing a framework for multirobot performance evaluation. In: Proceedings of PerMIS, Citeseer, vol. 2006 (2006)
3. Goodin, C., Kala, R., Carrillo, A., Liu, L.Y.: Sensor modeling for the virtual autonomous navigation environment. *IEEE Sensors* (2009)
4. Subbarao, M., Nikzad, A.: Model for image sensing and digitization in machine vision. In: Proceedings of SPIE, vol. 1385, p. 70. SPIE, San Jose (1991)
5. Born, M., Wolf, E.: Principles of Optics, 6th edn. Pergamon Press, Elmsford (1980)
6. Tuley, J., Vandapel, N., Hebert, M.: Analysis and Removal of artifacts in 3-D LADAR Data. In: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2005, pp. 2203–2210 (2005)
7. Rong, L., Chen, W., Yu, S., Song, W.: Mathematical model of coordinate transformations for 3D Depth-of-field collection system. In: 6th IEEE International Conference on Industrial Informatics, INDIN 2008, pp. 80–85 (2008)
8. SOPAC: Scripps orbit and permanent array center (2010), <http://sopac.ucsd.edu/cgi-bin/dbShowArraySitesMap.cgi>
9. Mehrtash, M.: Gps navigation toolbox gnt08.1.2 GNT08.1.2 (2008), <http://www.mathworks.com/matlabcentral/fileexchange/20578i>
10. RINEX: Receiver independent exchange format, ver. 2.10 (2009), <http://gps.wva.net/html.common/rinex.html>
11. Montenbruck, O., Garcia-Fernandez, M.: Ionosphere Path Delay Models for Spaceborne GPS. Deutsches Zentrum für Luft- und Raumfahrt DLR-GSOC TN 05-07 (2005)
12. Niell, A.E.: Improved atmospheric mapping functions for VLBI and GPS. *Earth Planets Space* 52, 699–702 (2000)
13. Niell, A.: The IMF Mapping Functions. In: GPSMet Workshop (2003)
14. Kim, H.I., Ha, J., Park, K.D., Lee, S., Kim, J.: Comparison of Tropospheric Signal Delay Models for GNSS Error Simulation. *Journal of Astronomy and Space Sciences* 26, 21–220 (2000)

15. Hwang, C., Hwang, L.S.: Satellite orbit error due to geopotential model error using perturbation theory: applications to ROCSAT-2 and COSMIC missions. *Computers and Geosciences* 28, 357–367 (2000)
16. Parkinson, B., Spilker, J.: The global positioning system: theory and applications. In: Aiaa (1996)
17. Rohde, M., Crawford, J., Toschlog, M., Iagnemma, K., Kewlani, G., Cummins, C., Jones, R., Horner, D.: An interactive physics-based unmanned ground vehicle simulator leveraging open source gaming technology: progress in the development and application of the virtual autonomous navigation environment (VANE) desktop. In: Gerhart, G.R., Gage, D.W., Shoemaker, C.M. (eds.) *Unmanned Systems Technology XI*. SPIE, Orlando (2009)

GPS/Galileo Testbed Using a High Precision Optical Positioning System*

Robin Heß and Klaus Schilling

University of Würzburg, Informatik 7, Am Hubland D-97074 Würzburg, Germany

Abstract. This paper presents a new approach for verifying and testing devices and algorithms that use position information determined by the Global Position System (GPS), a GPS testbed. A high precision tracking system is used measuring the position of an object and error terms and an offset are added in order to simulate its GPS position. The distribution of the position error of a common GPS receiver is analyzed in this paper and applied to a random number generator used for simulating the errors. Finally experiments with a real mobile robot are made to verify the presented system.

1 Introduction

One of the key-problems in the field of mobile robot navigation is determining the position of a movable robot. Therefore in the past 20 years much effort was made developing new algorithms and sensors for absolute and relative determination of mobile robots position.

The most common system today, used for outdoor applications, is the satellite based Global Positioning System (GPS) respectively the future Galileo. Using commercial off-the-shelf hardware a maximum accuracy of $\pm 7.8m$ in 95% of the measurements can be achieved, due to systematic, nonsystematic errors and satellite position. Differential methods or military GPS receivers can minimize some of the systematic errors, which lead to an accuracy up to $\pm 0.3m$. However Differential Global Positioning systems are quite expansive. Before applying such systems the requirements for an application, using position information, have to be analyzed. In order to get a reliable proposition, hardware experiments are necessary. Due to limitations of standard GPS-receivers this is a challenging problem. The maximum update frequency and accuracy for determination of position are defined by the used hardware, hence it is only possible to simulate lower values out of the data from that receiver. The non geosynchronous orbit of GPS satellites causes another problem. Even at exactly the same location in relation to earth, the number of visible satellites and their distribution change over time, what has a huge influence to the accuracy [6]. For this reason repeatability of hardware experiments using GPS position information is not given, which makes reliable analysis very difficult.

* This work was partially supported by the Europäischen Fonds für regionale Entwicklung (EFRE) der EU und des Freistaates Bayern.

In this context the paper presents a GPS testbed which enables the user to analyze and verify algorithms and applications that use GPS position information as input. Therefore the ground truth position of a mobile robot is converted into UTM format and added by an adjustable error, which represents the uncertainty of the GPS measurements. The Ground truth position is determined by the high precision laser based system iSpace, which exceeds visual tracking systems in accuracy, sampling frequency and coverage.

Related work in this field mostly addresses position location/determination of mobile robots, by different methods like visual tracking [1], odometry or localization with wireless lans [10], [12]. But as far as the authors know, no work combines ground truth position of a mobile robot with adjustable errors in order to test and verify GPS based algorithms, yet.

We structure this work as follows. The next section defines the requirements, including an analysis of the position error made by common GPS receivers. Section 3 gives an overview over the system design of the testbed and its components. Afterwards the made experiments are described and their results are discussed. Finally we draw a conclusion of the made work and address possible further studies related to this approach.

2 Requirements for GPS Testbed

2.1 Tracking of Moving and Static Object

Several demands are made on a GPS/Galileo testbed for mobile robots. First the system has to have the ability to track moving and static objects and determine their 3-dimensional position in relation to an earth fixed coordinate system. The used tracking system should work at least indoor, though outdoor capabilities would extend the application range of the testbed, i.e. the possibility of direct comparison between positions gathered via GPS and the system presented here. Moreover the tracking system must not influence the analyzed objects in any way. Therefore wireless or passive tracking systems are required. The accuracy of position determination has to outperform not only common commercial off-the-shelf GPS receivers, but also DGPS or military devices, so that a maximum position error in each direction of $\pm 1mm$ is mandatory, even for moving objects. In order to simulate sampling rates of different available receivers by down sampling, the tracking system has to provide at least the sampling rate of fastest receiver that has to be emulated.

2.2 GPS Error Analysis

The data, gathered by a tracking system that fulfills all requirements pointed out before, are considered as the error-free ground truth position of a mobile robot within the test bed. In order to emulate GPS data, an additional error term is required. The random error, added to the ground truth position data by the testbed, and the GPS position error have to have the same distribution, at

least approximately. Thus an analysis of GPS position error was made using a common GPS receiver.

The standard deviation of the user equivalent range errors σ can be defined as square root of three main parts shown in equation 1

$$\sigma = \sqrt{PDOP^2 \times \sigma_r^2 + \sigma_{numerical}^2} \quad (1)$$

Where PDOP is the Position Dilution of Precision, σ_r denotes the standard deviation of receiver position error and σ_{num} is a numerical error, which is usually set to 1m. The dilution of position [6] is a quantity that specifies effects on GPS position accuracy cause by satellite geometry and constellation. The standard deviation of receivers position consists of several terms cause by systematic and unsystematic errors like tropospheric refraction, ionospheric refraction or multi path effects.

The detailed composition of σ_r is beyond the focus of this paper and can be found in [5]. For this approach the distribution of the position error is more important then its source, thus a preliminary data analysis of position data gathered by a common GPS receiver was made. Position information were given in the Cartesian UTM coordinate system $p(t) = (Northing(t), Easting(t))$ $t = 1\dots n$ using the WGS84 ellipsoid as earth model [4]. A GPS receiver was located at a fixed position and data measured by the receiver were recorded. Since the true position, without error is unknown, the median of the measurements shown in equation 2, is used instead.

$$\begin{aligned} p_{true} &= (Northing_{true}, Easting_{true}) \\ &= \left(\frac{\sum_{i=1}^n Northing(i)}{n}, \frac{\sum_{i=1}^n Easting(i)}{n} \right) \end{aligned} \quad (2)$$

where p is the position vector given by the GPS receiver, consisting of the UTM coordinates *Northing* and *Easting* indicating the position in the actual UTM zone. n denotes the number of recorded position samples $p(1)\dots p(n)$. The discrete errors in each direction $\sigma_{Northing}$ and $\sigma_{Easting}$ are given by

$$\begin{aligned} \sigma_{Northing}(t) &= Northing(t) - Northing_{true} \\ \sigma_{Easting}(t) &= Easting(t) - Easting_{true} \end{aligned} \quad (3)$$

An analysis of the Rooted Squared Errors RSE of the unbiased position errors σ in each direction (equation 4) shows that a Rayleigh distribution is a sufficient approximation for the GPS position error distribution, like it was expected according to [13]. The histograms of the RSE are displayed in figure 11.

$$\begin{aligned} RSE_{Northing}(t) &= \sqrt{meanerror_{Northing}^2 + \sigma_{Northing}(t)^2} = |\sigma_{Northing}(t)| \\ RSE_{Easting}(t) &= \sqrt{meanerror_{Easting}^2 + \sigma_{Easting}(t)^2} = |\sigma_{Easting}(t)| \\ \text{with} \quad meanerror_{Northing} &= meanerror_{Easting} = 0 \end{aligned} \quad (4)$$

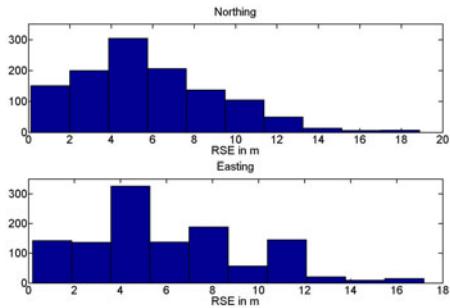


Fig. 1. histograms of Rooted Squared Errors of position errors

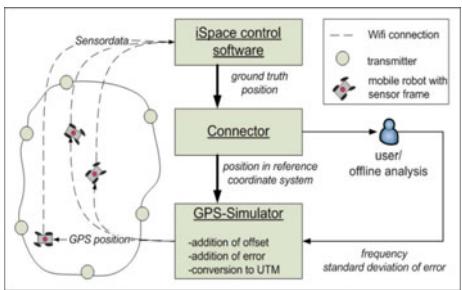


Fig. 2. system overview of GPS testbed

3 System Overview

Figure 2 shows the schematic configuration of the developed system, consisting of the high precision positioning and tracking system iSpace [9], which will be introduced in the following section 3.1, placed in the test area, the software packages iSpace-connector and GPS-Simulator, presented in Section 3.2, and the mobile car like robot MERLIN [3], [2] used for the experiments.

3.1 Positioning and Tracking System iSpace

iSpace is a high-precision optical laser based metrology and tracking system suitable for indoor as well as outdoor use [8], [7]. It consists of passive laser transmitters, placed in the area of operation and wireless sensor frames. The rotating transmitters emit two fan-shaped laser beams at right angles, which are detected by the sensor frames. A sensor frame consists of at least one photo sensor and a small sized radio frequency communication module. The RF-module (PCE) provides a data link, sending the sensor data to the iSpace control software. Even though a sensor frame with one detector would be sufficient measuring three translational axis, four detectors are used, which increases the accuracy.

The final position information is calculated by the iSpace control software on a dedicated PC. Out of the transmitted data from the sensor frame the elevation and azimuth angles between every detector of the sensor frame and each visible transmitter are calculated which define a straight line between transmitter and detector. The length of the line is derived via triangulation. Therefore the knowledge about the location of the transmitters relative to each other is necessary. This is done by a semi-automatic calibration process, measuring a few hundred points inside the target volume with a special sensor frame. The system determines the relative transmitter position by dint of an optimization

process out of the recorded data. The presented application requires a user defined coordinate system which can be applied by measuring 3 points (origin, x-axes, y-axes). ISpace uses this system, if desired, as reference coordinate system and does all necessary transformations.

The accuracy of the position data is measured by the system itself. For typical environments a maximum error of $\pm 0.25\text{mm}$ could be achieved with a sampling rate of 40 Hz. Though due to the necessary processing time for calculating the position, this data rate is only applicable for off-line analyses. For on line applications like this approach sampling rates up to 20 Hz are possible.

3.2 iSpace Connector and GPS-Simulator

The core of the presented GPS testbed are the developed .Net software packages iSpace-Connector and GPS-simulator. ISpace-Connector interacts directly with the iSpace control software. It implements the publisher/observer pattern where at least every wireless sensor frame represents a publisher. In order to gather 3-dimensional position of wireless sensor frames, clients (observer) can connect to any available publisher via UDP. Each Publisher can handle up to 10 clients. If new data are available, they are fetched by iSpace-Connector once and sent to every connected client. For each client the reference coordinate system is user selectable.

For the proposed GPS testbed a special publisher the GPS-simulator was developed. This software behaves like the publishers presented before, fetching position information from a connected sensor frame and distributing them to connected clients, though the data are processed on line in order to adjust the user defined sampling rate (up to 20 Hz), apply adjustable errors and add the necessary offset. The offset is necessary to emulate magnitudes of real GPS position data in UTM format. The error is calculated with the help of a random number generator, which has the capability generating random numbers with different, user selectable, distributions [11]. Preliminary analysis of data, measured by real GPS receivers (section 2.2), showed, that a Rayleigh distribution is a sufficient approximation of the GPS position error distribution.

Thus for this application, a GPS test bed, random number generators that produce rayleigh distributed values are used. The onliest parameter of that distribution σ is set with the help of the standard deviation by the user (equation 5).

$$\sigma_{rayleigh} = \sqrt{\frac{2}{4 - \pi}} std_{rayleigh} \quad (5)$$

Since the errors in each direction are assumed to be independent of each other, three independent random number generators, using the same standard deviation, are used. Equation 6 shows the complete calculation of the position information p_{sim} distributed by GPS-Simulator.

$$\begin{aligned}
p_{sim}(t) = & \begin{pmatrix} Northings_{sim} \\ Eastings_{sim} \\ Altitudes_{sim} \end{pmatrix} = \begin{pmatrix} x_{sim} \\ y_{sim} \\ z_{sim} \end{pmatrix} \\
= & \begin{pmatrix} x_{iSpace} \\ y_{iSpace} \\ z_{iSpace} \end{pmatrix}(t) + (-1^{s_x} -1^{s_y} -1^{s_z})(t) \begin{pmatrix} e_x \\ e_y \\ e_z \end{pmatrix}(t) + \begin{pmatrix} offset_x \\ offset_y \\ offset_z \end{pmatrix} \\
s_i \in \{0, 1\} \text{ and } e_i \in \{ & \frac{re^{-\frac{r^2}{2\sigma_{rayleigh}^2}}}{\sigma_{rayleigh}^2} | r \in \mathbb{R}\} \text{ with } i \in \{x; y; z\} \quad (6)
\end{aligned}$$

Where $\begin{pmatrix} x_{iSpace} \\ y_{iSpace} \\ z_{iSpace} \end{pmatrix}$ denotes the 3 dimensional position vector of wireless sensor frame mounted on a static or dynamic object (mobile robot) gathered by iSpace, s either 0 or 1 (equal distributed) and e is the rayleigh distributed random error.

In summery the tasks of iSpace Connector and GPS-Simulator are gathering 3 dimensional positions of static or dynamic objects in relation to a coordinate system aligned to the UTM grid, adding offsets and random errors with a defined distribution and an user set standard deviation and down sampling the signal to a desired sampling frequency. The results in UTM format are finally sent to the connected clients.

4 Experiments

Several experiments were made in order to show that the GPS testbed works as desired. The system was assembled in a test hall, placing six iSpace transmitters on tripods distributed around the test area (size 10m by 10m). A reference coordinate system was defined by measuring the origin and one point on x-axis respectively y-axis. In order to align this coordinate system to the UTM grid, a compass was used to determine the correct northern alignment of the x-axis. The alignment of y-axis was done by measuring a right angle from x-axis in eastern direction. After setting up iSpace control software, iSpace connector and

GPS-simulator an offset $\begin{pmatrix} offset_x \\ offset_y \\ offset_z \end{pmatrix} = \begin{pmatrix} 570168.4 \\ 5514759.9 \\ 321 \end{pmatrix}$ UTMzone 32U was de-

termined, which is a point inside the test hall. All tests presented here were made with the mobile robot Merlin, attached with a sensor frame and a PCE. The position data are directly logged by the mobile robot. In addition the positions measured by iSpace itself, without added error, are logged for analysis of the test results as well. Since the test hall has nearly perfect flat surface the height (z-axis) is not analyzed here.

4.1 Static Test

The first experiments should verify the distribution of the added error terms. Therefore the mobile robot is placed within the test area but not moved in any

Table 1. Experiment parameters and results

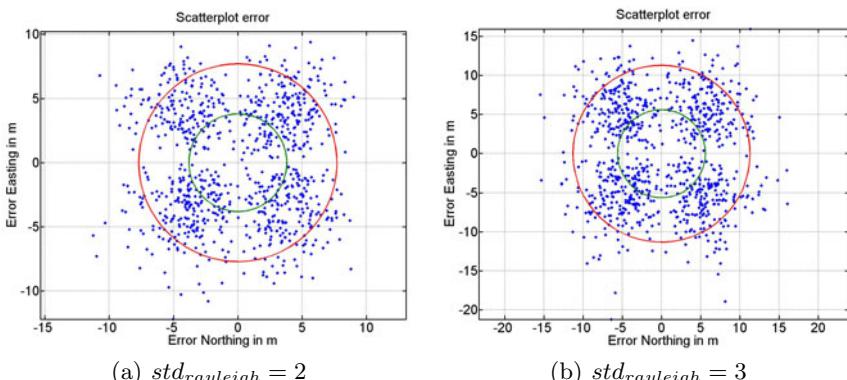
Test	$std_{rayleigh}$	50% percentile in m	95% percentile in m
a	1	1.8	3.9
b	2	3.8	7.7
c	3	5.6	11.3
d	4	7.2	15.8
e	5	8.4	18.9

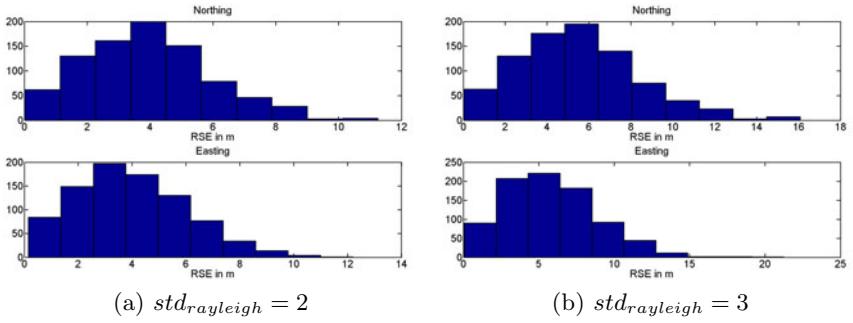
direction. The sampling rate is set to 1 Hz. Different standard deviations for the error are also taken into account in order to analyze their effect on the accuracy (at 50th or 95th percentile).

Table 1 displays the analyzed standard deviations, the 50th percentile (50% of errors are smaller than this value) and the 95th percentile (50% of errors are smaller than this value) respectively. Common GPS receiver reach an 50th percentile of up to 4 m (95th percentile of up to 7.8 m), so that a standard deviation between 2 and 3 seems to be a good parameter for GPS Simulator in order to emulate commercial-of-the-shelf GPS receivers. Figure 3 displays the scatter plots of the position error for standard deviation $std_{rayleigh} = 2$ and $std_{rayleigh} = 3$. The circles represent the 50th percentile (inner circle) and the 95th percentile (outer circle) respectively. The histogram of the rooted squared position error for $std_{rayleigh} = 2$ and $std_{rayleigh} = 3$ in figure 4 show nearly the same error distribution like the one in section 2.2, so the chosen random number generator is reliable for generating the desired error terms.

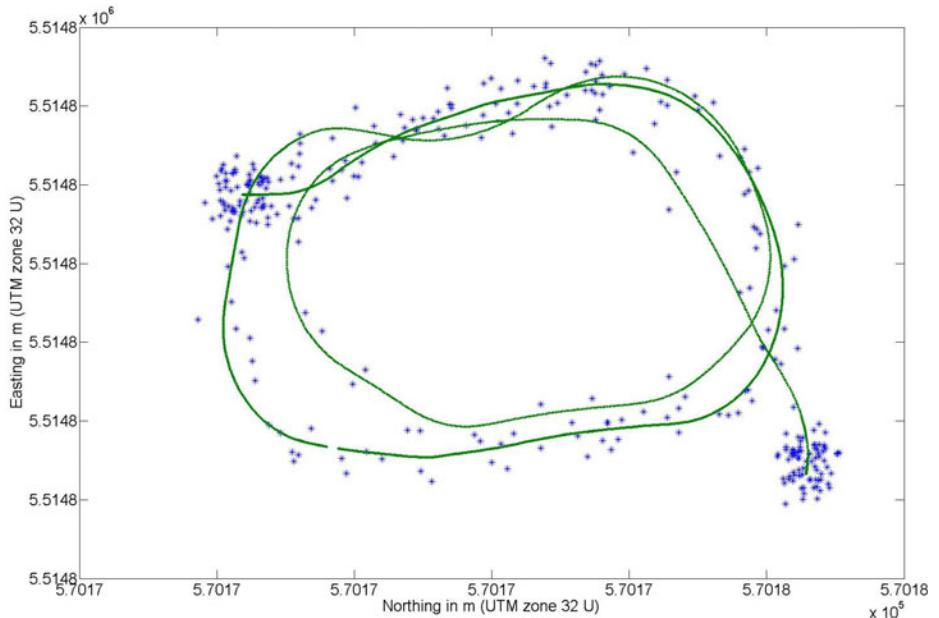
4.2 Dynamic Test

Subsequent to the static experiments a dynamic test was performed in order to verify the functionality of the presented system under real conditions. Therefore the mobile robot was tele-operated within the GPS testbed. The trajectory was randomly chosen by the operator. The sampling rate of the test bed was set

**Fig. 3.** Scatter plot

**Fig. 4.** Histogram of position error

to 1 Hz. Due to the limited space of 10m by 10m in the test area during this test, the standard deviation had to be set to a very small value, otherwise the errors cover the whole area and make analysis impossible. Note that the tracking system iSpace can cover areas up to 12000m (circle with 90 m in diameter), so higher error standard deviation can be applied as well if there is enough space. In addition to the data logged by the mobile robot, the real trajectory is stored by iSpace as well. Figure 5 shows the result of the dynamic test. The reference trajectory is presented by the dotted green line. The blue crosses denote the simulated GPS positions. The result of the dynamic test shows, that the

**Fig. 5.** Results dynamic test

presented GPS testbed is usable for testing and verifying GPS data for dynamic objects, like mobile robots as well.

5 Conclusion and Future Work

This paper introduced a new approach for testing and verifying algorithms that use position information determined by the satellite based Global Position System. The presented GPS testbed tracks the real position of dynamic or static objects in relation to an earth fixed coordinate system as base for an emulation of objects GPS position. The GPS position errors are simulated by a random number generator. The results of the experiments showed, that the used Rayleigh distribution is a good approximation of the GPS position error distribution. It was also demonstrated, that the presented approach is usable for analyzing static and dynamic objects, as well.

In order to improve the presented GPS testbed some future work has to be done. First of all additional analysis of the GPS error is necessary. The vertical error was not taken into account at all in this work. The relation between the one dimensional errors is also from interest. In this work it was assumed, that errors of every direction are independent of each other. Furthermore the system has to be tested in large scale areas and outdoor environment as well. It is also planned to build a black box, that behaves like a real GPS receiver, connected via RS232 or USB, but uses data determined by the GPS testbed. Therefore the whole protocol implemented by a common GPS receiver has to be analyzed and emulated.

References

1. Barman, R., Kingdon, S., Mackworth, A., Pai, D., Sahota, M., Wilkinson, H., Zhang, Y.: Dynamite: A testbed for multiple mobile robots. In: Proceedings of IJCAI 1993 Workshop on Dynamically Interacting Robots, Chambéry, France, Citeseer, p. 38 (1993)
2. Eck, D., Stahl, M., Schilling, K.: The small outdoor rover merlin and its assistance system for tele-operations. In: Proceedings of the 6th International Conference on Field and Service Robotics. Chamonix, France (2007)
3. Frank, M., Busch, S., Dietz, P., Schilling, K.: Teleoperations of a mobile outdoor robot with adjustable autonomy. In: Proceedings 2nd ACIDCA-ICMI 2005 International Conference on Machine Intelligence, Tozeur, Tunisia (2005)
4. Hager, J., Behensky, J., Drew, B.: The universal grids: Universal transverse mercator (utm) and universal polar stereographic (ups). Technical manual, Defense Mapping Agency (1989)
5. Hofmann-Wellenhof, B.: Global positioning system (1997)
6. Langley, R.: Dilution of precision. GPS world 10(5), 52–59 (1999)
7. Lemmer, L., Hess, R., Kraus, M., Schilling, K.: Calibration of a car-like mobile robot with a high-precision positioning system. In: The Second IFAC Symposium on Telematics Applications, TA 2010 (October 2010)

8. Nikon Metrology: Portable Metrology Systems User Manual and Startup Guide. 60 Northland Road, Units 6 & 7, Waterloo, Ontario, Canada, N2V 2B8 (January 2009)
9. Nikon Metrology: ispace (2010),
http://www.nikonmetrology.com/products/large_volume_tracking_positioning/ispace (visited on June 7, 2010)
10. Prasithsangaree, P., Krishnamurthy, P., Chrysanthis, P.: On indoor position location with wireless LANs. In: The 13th IEEE International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC 2002), Lisbon, Portugal, Citeseer (2002)
11. Troschuetz, S.: .net random number generators and distributions (2010),
<http://www.codeproject.com/KB/recipes/Random.aspx> (visited on June 7, 2010)
12. Wang, Y., Jia, X., Lee, H., Li, G.: An indoors wireless positioning system based on wireless local area network infrastructure. In: Proceedings of the 6th International Symposium on Satellite Navigation Technology Including Mobile Positioning & Location Services, Citeseer (2003)
13. Wilson, D.L.: David l. wilson's gps accuracy web page (2010),
<http://users.erols.com/dlwilson/gpsacc.htm> (visited on June 7, 2010)

Validating Vision and Robotic Algorithms for Dynamic Real World Environments

Tobias Kotthäuser and Bärbel Mertsching

GET Lab, Paderborn University
Pohlweg 47 - 49, 33098 Paderborn, Germany
{kotthaeuser,mertsching}@get.upb.de
<http://getwww.uni-paderborn.de>

Abstract. The efficient development of complex algorithms for mobile robot applications is strongly encouraged by appropriate testing and validation opportunities. In this paper, we present SIMORE (Simulation of Multirobot Environments), a simulation framework for simulating sensors, actuators and entire robots, in virtual 3D environments. It provides synthesized sensor data but also additional information emanating from the intrinsic properties of given virtual environments. SIMORE is built upon open-source libraries such as the *OpenSceneGraph* toolkit for scene rendering and the *Open Dynamics Engine* to guarantee a solid physical behavior. It provides easy to use modeling tools and plug-ins for creating and modifying the appearance as well as the dynamic properties of objects to be included in the scene. Our simulation framework possess a module based design capable of supporting further extensions and enhancements such as the definition of individual sensors and actuators. Beyond that, SIMORE offers exemplary interfaces for available robotic middleware applications, such as *Player* or *ROS*, allowing for a fast and flexible transition between simulation and real hardware platforms. Although still under active development, SIMORE is used to support several research propositions in robot vision, navigation and recognition as well as being a proven prototyping tool for robotic competitions, such as the Robocup (Rescue Ligue).

1 Introduction

The usage of virtual environments for simulating mobile robot platforms provides several advantages; virtual environments have – compared to the real world – less limitations to the number of used sensors, actuators and robots or their arrangement in the environment. Virtual environments support the reproducibility of scenes with manually adjustable complexity regarding the number and appearance of objects; different algorithms can be benchmarked with completely identical input scenes.

Simulations offer high debugging capabilities and the ability to neglect disturbing effects such as unintended robot motions or sensor inaccuracies enabling the researcher to focus on the main problems during the implementation phase; virtual sensors provide ground truth data for all sensor measurements.

Of particular importance is the opportunity to test validity and performance of complex algorithms without any risk of damage or even destruction of the system such as

in the case of an incorrect operation. Thus, we have the potentiality to evaluate action-perception-cycles in the virtual environment for efficiency and safety before running them on a real hardware platform.

1.1 Requirements

The huge number of existing robot simulators (compare section I.2) offers a variety of features, models and scenarios for the simulation of mobile robots. However, many existing simulators do not necessarily allow dwelling with specific or individual research questions; their frameworks lack access to certain sensor or internal simulation data due to their restrictive interfaces, they are too complicated to extend or they use specific models or modeling software within their simulation context that possess restrictions to their users.

For our applications it is necessary to not only receive plain synthesized ground truth data, but to further receive labels corresponding to that data or to have direct access to certain internal states of the simulation. Labels may contain intrinsic information about perceived objects in the simulated environment, such as dynamic properties and conditions for instance. Especially for the evaluation of algorithms or learning purposes in general, labeled data is a decisive expedient. Direct access to internal data, for instance retrieving the correspondence between a pixel in a synthesized image and an object in the scene for validation purposes, must be feasible for exterior applications.

Since we want to exert the synthesis of images, we strongly rely on the quality of the rendered scene. The quality can be enhanced by synthesizing photo-realistic rendering in combination with high-resolution textures and by using meticulously modeled objects. In many of the available simulators, we found options to individually create 3D models via descriptions; however, most of them possess a rather abstract and unrealistic appearance. Therefore, we aim for a solution using a professional modeling tool to produce, on the one hand, high-quality models including their visual appearance, kinematic structure and dynamic behavior (physics), while on the other hand, utilizing a comfortable way of transforming each model as one entity directly into the simulation.

Furthermore, we rely on higher-level abstractions providing efficient communication and interoperability mechanisms. Therefore, the demanded simulator should be used in combination with available robotic middleware frameworks to enhance software integration and reuse.

To provide a solid test bench for navigation tasks or semi-autonomous behaviors, the simulation framework should provide a basic user interface for the emission of control commands to the middleware. Desirable is the selection of single goals, paths and objects in the simulator's main view. On the other hand, the simulation should also be capable of displaying high-level information such as estimated goal positions of a robot in order to simplify the debugging process.

1.2 Related Work

A multitude of simulators for mobile robots exists. In this section we will briefly focus on those for mobile robot platforms with respect to the proposed framework.

The open source project *USARSim* [12] is a simulator based upon the *Unreal Engine* 2.0. Its modeling and programming can be done by tools which are part of the engine. This project supports several sensors (cameras, distance, motion, etc.) and robots (wheeled, legged, flying robots, etc.) implemented in the *Unreal Script*. USARSim offers an interface to middleware framework *Player*.

Another simulation environment for mobile systems is *Webots* [3], a commercially available product of Cyberbotics. It offers functionalities such as libraries for different sensors (camera, distance, light, touch, etc.) and robots (wheeled, legged and flying). An interactive manual interface to manipulate the scenes during simulation and a physics simulation is based on the *Open Dynamics Engine*. Additionally, there is a possibility to integrate individual algorithms into the simulator to evaluate them.

Another project is the *Gazebo* [45] simulator related to the *Player* project. Gazebo is a multi-robot simulator for outdoor environments also supporting a range of sensors (cameras, different distance sensors, GPS, etc.) and robots (wheeled, legged, flying, etc.). It features a simulation for rigid-body physics through the *Open Dynamics Engine*. Besides an available client-server interface, Gazebo possess interfaces to middleware frameworks as *Player* and latterly the *Robot Operating System*.

The commercially available *Microsoft Robotics Developer Studio* [6] comes with the *Visual Simulation Environment* as its Simulator. It is using NVidia's physics engine *PhysX* [7] and provides a range of pre-modeled prevalent robotic platforms and sensors and environments.

In this paper, the simulation framework SIMORE (Simulation of Multirobot Environments) is presented. It simulates sensors, actuators and entire robots, in virtual 3D environments for applications such as robot vision as well as navigation and mapping behaviors. Recent advances include: the access of internal simulation data which can be delivered additionally to the sensor data; the implementation of our modeling concept for visual and physics components as one entity; the assembly of high fidelity robot models via XML descriptions and finally the realization of interfaces to common middleware frameworks directly enabling our software to alternate between simulation and real platforms.

2 Simulation with SIMORE

The SIMORE framework, previously conceptualized in [8][10] consists of several components which will be described in detail in this section. A functional overview of the components of SIMORE and their relationships is presented in fig. 1

For individual modeling, OSG files are created with the 3ds Max OSG plug-ins. The output files can contain visual, physics and miscellaneous components (see section 2.3). Based on the available OSG files, the user can use the *SIMORE Editor* in order to configure individual worlds or robots, which can then be exported as OSG files or in the XML format.

The main components within the simulation loop are the physics simulation with the OpenDynamicsEngine (see section 2.1) as well as updating and rendering the scene graph (see section 2.2). Components for handling sensors and actuators manage acquired sensor data and dispatch incoming commands to the simulated actors. All these

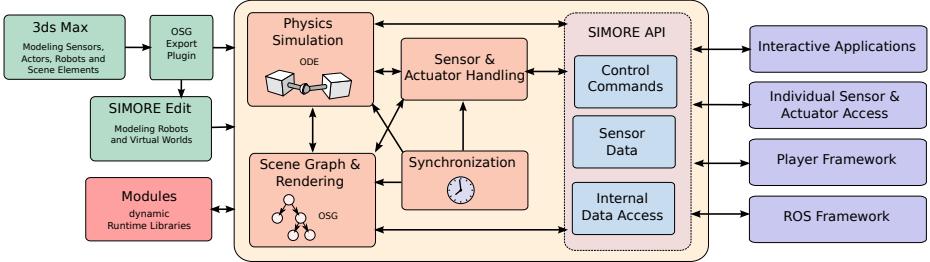


Fig. 1. The SIMORE design and the relations between its components

components are synchronized with a virtual simulation time. The SIMORE API regulates the access for setting commands, receiving sensor data as well as internal simulation data.

The API is used by applications intending to interact with a simulation object. This category exhibits interactive applications that allow the control of the simulator as a stand-alone application (mainly used for testing and demonstration), applications that allow individual access to certain simulation features without using a dedicated communication framework and finally, implementations for the middleware frameworks *Player* and *ROS* (ref. section [2.5]).

In order for a flexible extension of our framework to be possible, we allowed for modules which can be added supplementarily to the simulator and are loaded at run time. These modules are provides access to ODE objects and the scenegraph root as well as callback functions for predefined actions.

2.1 Physics Simulation

The necessary computations for simulating the dynamic characteristics and the physical processes of a scene can be accomplished by a physics engine. SIMORE deploys the Open Dynamics Engine (ODE) for its physics simulation. ODE is a platform independent open source library for simulating rigid body dynamics [1]. It provides many features, including a variety of joint types, assigning arbitrary masses and mass distributions and creating geometric hulls for collision detection. The physical characteristics of an object in ODE are described as a *body*. A body contains information about its mass, position and orientation within the virtual world. When forces are acting on a body, for example gravity, the dynamic behavior of the body is calculated within a predetermined time step. After each simulation (integration) step, the dynamic properties of the body are applied which might affect the body's attributes, such as position, orientation, velocities, etc. The idea behind most simulators is to assign these attributes to graphical objects contained in a scene (see section [2.3]).

Bodies can be connected among themselves by means of joints. Several joints types are provided in ODE such as: slider joints, which permit only a movement along a given axes; hinge joints for connections with one or two rotation axes; and ball joints for rotations around all three axes or fixed joints for fixed connections between two rigid bodies.

The definition of bodies alone does not contain any information about their geometrical structure. In order to allow the evaluation on potential collisions between two or more bodies, collision objects (also named *geoms*) can be assigned to a body. The detection of collisions is now a matter of recognizing the contact or the penetration of two objects in 3D space. However, the computation of collisions for complex geometric objects is a relatively time-consuming task. In order to reduce the computation costs, the geometric structures used for collision detection are mostly represented by simple primitives (e.g. boxes, spheres, cylinders, ...) rather than complex graphical objects (meshes). This coherence is visualized in fig. 2 where we can see the complex visual representation on the right side and its unembellished collision representation on the left.

2.2 Scene Rendering

From the existing open source rendering engines, we decided to use OpenSceneGraph [12][13] instead of OGRE [14] due to its well-structured programming interface and its extension potentials. The 3D graphics toolkit OpenSceneGraph is applicable in fields including visual simulation as well as scientific visualization and modeling. It is based upon OpenGL and supports shading languages such as the OpenGL Shading Language or NVidia's CG [15]. OpenSceneGraph is a multi-threaded library and based on the scene graph concept. A scene graph is a collection of nodes in an acyclic graph where by default each node represents (graphical) entities such as drawable meshes. Since geometrical transformations can be associated to each node, graphical hierarchies can be arranged in a forward kinematic order. OpenSceneGraph's so-called *NodeKit* concept enables the opportunity to extend the scene graph by individual nodes implementing arbitrary datasets and functionalities. For our simulation approach, we extended the scene graph by the aforementioned ODE entities such as collision bodies and dynamic parameters such as mass and mass inertia. In addition, we also considered special nodes for sensor and meta-information as well as sound sources to be included in the scene graph.

The enhancement of the existing scene graph allows us to rely on an existing library and an additional open source project that handles export from 3ds Max¹ modeling software called *OSGExp* (ref. section 2.3). *OSGExp* was extended by our node types in the way that scene data from 3ds Max can be adopted. The NodeKit consists of three libraries, the main node kit library, an export-import library for OpenSceneGraph and finally an object library exclusively for 3ds Max.

2.3 Modeling

The opportunity to create and modify simulated environments according to individual demands and situations is a vital requirement for a mobile robotics simulation framework. A virtual environment usually consists of several object models that are integrated into the simulation. The basic storage and exchange of models in SIMORE is performed through files in the OpenSceneGraph (OSG) file format. Such files basically contain the structure of the scene graph either entirely or as subgraph. We extended the structure of

¹ 3ds Max is a registered trademark of Autodesk, Inc. All rights reserved.



Fig. 2. Assembling of a basic model of a robotic platform: The left side represents the collision objects used for collision detection by ODE. The right side depicts the visual component that is rendered by OpenSceneGraph. This model can be equipped with further sensors and actuators.

OSG files via the OpenSceneGraph NodeKit concept by defining individual node types such as bodies, collision objects and joints for ODE, as well as nodes for sensors and special sensor inputs (we also implemented sound sources for microphone sensors for instance).

In this manner we have established a structure capable of representing complex objects consisting of rigid bodies that can be linked by the joint types defined in ODE, physical shapes defined by ODE collision objects as well as visual representations defined by the nodes that can be natively rendered in OSG. While loading a file into the scene graph, the individual node types are assigned to specific actions; The ODE nodes and their dynamic parameters (masses, frictions, etc.) are extracted from the graph in the OSG file and automatically registered in ODE, assigned to other ODE components as well as to subordinated OSG nodes attached to the superordinated bodies.

For modeling simple objects, sensors, actuators and entire robot platforms, we are deploying the 3D modeling, animation, and rendering software 3ds Max. In order to integrate the aforementioned node types, we created plug-ins in the 3ds Max scripting language. These plug-ins allow us to easily combine and integrate bodies, joints, collision objects and sensors into the scene. Furthermore, they allow to set individual parameters and to apply the default spatial transformations defined in 3ds Max. Finally, the plug-ins assist the user with features, such as aligning ODE nodes relative to visual objects, or automatically approximating given 3D model with appropriate collision objects.

However, not all models which are used have to be modelled in 3ds Max. Another option is to combine existing SIMORE models via XML descriptions assembled by a 3D editor application for SIMORE worlds and robots. Within this tool the user can load individual OSG files and after aligning objects to individual requirements, created scenarios can be exported as XML world descriptions or directly as an OSG file. This way, the user can determine the modeling granularity by either modeling entire scenarios and robots in 3ds Max or modeling more elementary objects which can be fused at run time. Note, that the user does not need deeper knowledge of ODE or programming backgrounds in order to create individual objects or scenes featuring physics properties. As an elementary foundation, we provided pre-modeled sensors, actuators and several robot testing arenas, including Robocup Rescue (including contrastingly textured walls, step-fields, ramps, etc.) and SICK robot day arenas as well as other experimental

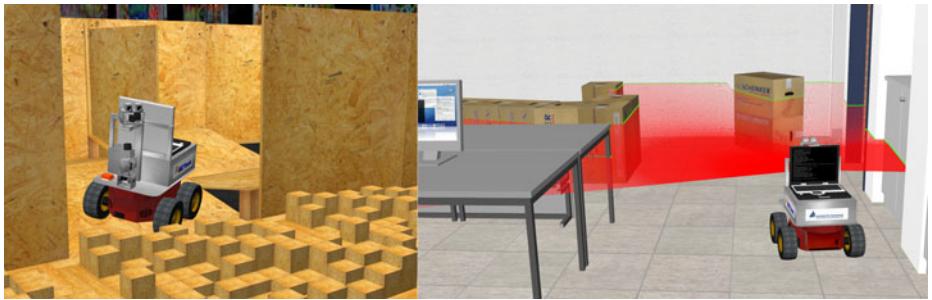


Fig. 3. Screenshots of typical simulation scenarios. The left image shows a rescue robot in a Robocup Rescue arena with fairly cluttered elements such as ramps and step fields. The aim of the application here is to create a map of the arena, determine the trafficability of the underground and detect victims hidden behinds holes in the walls. The right image shows a robot and its visualized laser range finder in an experimental lab environment.

environments, such as lab and kitchen scenes. These testing scenarios can contain miscellaneous objects on a lower granularity objects, such as furniture, lab equipment and robocup elements (see fig. ③).

2.4 Simulated Sensors

Range sensors are implemented based on intersections with ODE collision objects available in the scene. Therefore, the collision of straight lines (simulated rays) are distributed from the sensor's origin into the scene. The calculation of hit points is realized with the collision callbacks available in ODE. Each hit point can inherit additional information from the compound object (visual and physics components) superordinating the corresponding collision object. Therefore, we can provide the scan with additional channels containing labels.

The views of available camera objects are rendered into graphics memory and transferred into main memory. For the realization of this *read back* functionality, different methods can be implemented. Hardware accelerated offscreen rendering techniques such as *frame buffer objects* (FBO) and *pixel buffers* (PBuffer) can be easily used within OpenSceneGraph. These OpenGL extensions provide buffers which can act as render targets, e.g. textures (*render-to-texture*). The content of the textures can be read back and stored in images. PBuffers have a disadvantage as they depend on the operation system and require a render context switch which reduces the performance significantly. Another way to read back images from the graphics memory, which we use in our framework, is to use the frame buffer of the application with multiple viewports. The different viewports are placed behind each other in the direction of their rendering order (from back to front) so that the camera sensor outputs can be read back immediately. OpenSceneGraph provides a functionality to render a scene directly into image.

The synthesis of internal sensors like *Inertial Measurement Units* (IMU), odometry, or encoders are obtained from the physics simulation.

2.5 Integration into Existing Middleware Frameworks

Many of the existing simulation frameworks are designed as client-server architectures that often come with their own network interfaces in order for client applications to acquire sensor data and to set control commands (compare [16]). Since our simulator aims at contributing to research in disciplines, such as object recognition, navigation and mapping, we decided not to focus on developing an individual network interface for our framework, but rather on deploying appropriate robotic middleware applications such as Player [17], OpenRDK [18], Orococos [19] or Robot Operating System (ROS) [20] alleviating the communication between the simulator and its applications.

Until now, we implemented interfaces on two frameworks; Player and the ROS. The widely-used Player framework offers basic functionalities for the exchange of sensor and control data together with the according networking capabilities. For a more distinctive hardware abstraction, flexible message communication and synchronization, we implemented an interface to ROS. It facilitates larger projects requiring multiple and versatilely equipped robots, its offers a more flexible message types and a wide selection of analysis tools making ROS a convincing instrument for our requirements in robotics research. Most of our recent prototyping efforts have used the combination of SIMORE and ROS (see section 3).

3 Applications and Examples

3.1 Research Tool

Over the last years, SIMORE was utilized in several applications for validating and benchmarking machine vision algorithms.

For example, for the evaluation of visual attention algorithms with SIMORE, objects with varying features such as size, color, shape, etc. are included in various scenes. These scenes are then explored by a virtual robot carrying a monocular camera mounted on a pan-tilt unit [21]. We introduced a spatial memory unit connected to SIMORE in order to select a 3D model from the stored data-base matching with the object under the focus of attention. Visual features of the attended object in the current camera image, information about location of robot in the scene and the pan-tilt angles of the camera head are used to validate a spatial inhibition of return [22].

For the purpose of motion segmentation, we simulate a virtual moving robot perceiving its dynamic environment with stereo cameras on a pan-tilt unit. The extracted motion retrieved from stereo images is validated by comparing it to the reference motion performed in the simulator including the ego-motion of the robot [23][24]. SIMORE allows to control the complexity of moving objects and their motions as well as their saliency. Finally, an optimization of the 3D motion parameter estimation could be used to enhance the computational speed of the motion segmentation algorithms [25].

More recent research considers the processing of labeled 3D data created in virtual interior environments. 3D point clouds acquired with tilted laser range finders and time-of-flight cameras are used as labeled (object type, surface properties, etc.) training data for the classification of objects in the scene with methods such as domain adaption

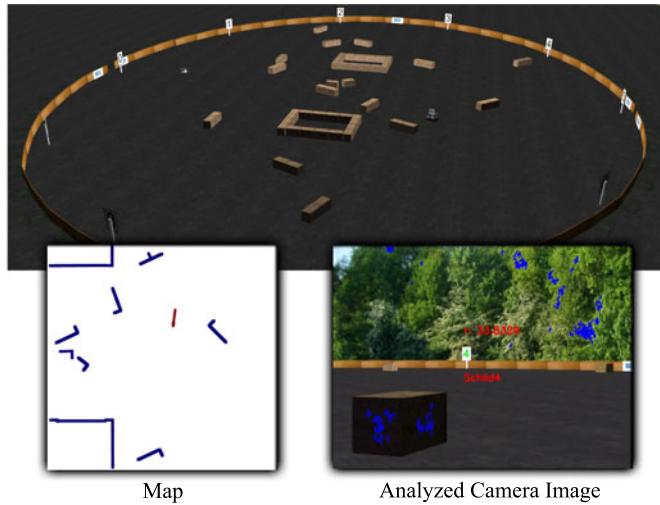


Fig. 4. The *SICK Robot Day* competition in SIMORE. Two robots have to approach given landmarks in a predefined order with avoiding collisions with their environment or their competitor. The figure shows the simulated environment including two robots and processed sensor output in form of a map and an analyzed image containing salient regions, the detected sign and it's distance to the robot.

[26] or conditional random fields [27][28]. In this context we will validate classification results and intermediate steps with the help of the preset labels.

3.2 Scenarios for Robotic Competitions

There are different testing scenarios which are available in SIMORE. The *Robocup Rescue* competition [29] scenario simulates disastrous situations in which robots are employed to map the environment and to detect signs of human life. The scene is abstractly represented by a maze-like arena consisting of wooden elements such as walls, ramps, and step-fields. The availability of these scenarios in SIMORE enables us to prototype our algorithms functionality in environments which are otherwise difficult to replicate in reality due to the dimension of the arenas and the required materials. Furthermore, the recombination of arena elements involved is tedious and time consuming. To be able to not only test our algorithms during the competition but throughout the entire year we replicated virtual arena elements in different variations for SIMORE (see fig. [3]).

Within the simulation we are able to test navigation behavior by parameterizing, for instance, friction and velocities on uneven surfaces such as ramps. We also consider maneuverability testings within the simulated environment, for example an analysis based on laser scans determining whether the robot is technically able to navigate on a specified terrain. Therefore, we reproduced critical terrains such as step-fields and crevices to be perceived from different angles and positions but also safe terrains appearing as alleged false negatives. Additionally, we used virtual cameras in combination with ac-

tutors to scan the environment for victims and to come within close proximity of them using the robot. In this connection, we also started simulating objects containing individual heat signatures in order to be able to synthesize thermal images.

Another available scenario originates from the *SICK Robot Day* [30] competition that combines elementary recognition and navigation tasks. The goal within this competition is to approach a sequence of given landmarks in a preassigned order as fast as possible. The landmarks, in the form of numbered signs, are distributed around the boundaries of an elliptical area covering 30×40 meters. To impede the robots' attempt at reaching the landmarks, the area is equipped with sets of irregularly shaped obstacles. Training for this competition is challenging due to the requisites in shape and dimension of the area. Eventually, we can test single recognition and approaching behaviors in reality however, not at any time during the entire application cycle. By virtually replicating the competition scenario and integrating it into SIMORE (see fig. 4), we were finally capable to prototype the interaction between our algorithms and their results for complete runs. For this application the virtual robot platform was equipped with a laser range finder and a virtual camera mounted on a pan-tilt unit. Benefits emerged for debugging when considering ground truth sign positions and orientations for the vision based recognition and position estimation as well as the behavior to approach a sign with an adequate distance and orientation. For navigation and obstacle avoidance, we used SIMORE to develop a *closest gap* based reactive behavior [31]. This was abetted by the opportunity to interactively select goal positions or aimed paths to be reached by the robot.

4 Conclusion

We have presented a simulation framework for mobile robot platforms providing pre-modeled scenarios, modeling tools and exemplary interfaces to common middleware applications allowing a robust portability of algorithms to physical robot platforms. The main advantages are the comfortable scene modeling techniques using the modeling software 3ds Max in combination with a 3D editor, the consideration of labeled data and its extensibility via individual modules that can be included by shared libraries.

Future research concentrates on enhancing virtual sensors to receive more realistic data. In this connection we will advance our present scheme to integrate appropriate noise models for range sensors and to consider distortions in the scan data resulting from ego-motions of robots. The simulation of such sensor inaccuracies is currently adapted outside the framework (as ROS node). We plan to integrate them as modules within the framework in order to particularly take account of errors depending on the simulated system. Additionally, we would like to extend the scene graph for the possibility of synthesizing thermal images.

We shall continue to speed up the available 3D scanning mechanisms to provide a basis for object recognition applications and semantic scene mapping. Furthermore, we will consider the intelligent generation of scenes via high level descriptions to achieve higher variations and clutter in simulated environments.

References

1. Carpin, S., Lewis, M., Wang, J., Balakirsky, S., Scrapper, C.: Usarsim: a robot simulator for research and education. In: ICRA, pp. 1400–1405. IEEE, Los Alamitos (2007)
2. Balakirsky, S., Scrapper, C., Carpin, S., Lewis, M.: Usarsim: Providing a framework for multi-robot performance evaluation. In: Proceedings of the Performance Metrics for Intelligent Systems Workshop, p. 1 (2006)
3. Cyberbotics Ltd: Webots (2010),
<http://www.cyberbotics.com/products/webots/>
4. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (2004)
5. Rusu, R., Maldonado, A., Beetz, M., Gerkey, B.: Extending player/stage/gazebo towards cognitive robots acting in ubiquitous sensorequipped environments. In: ICRA Workshop for Networked Robot Systems, Rome, Italy, Citeseer (2007)
6. Microsoft: Robotics developer studio 2008 (2010),
<http://www.microsoft.com/robotics>
7. NVIDIA Corporation: Physx (2010),
http://www.nvidia.com/object/physx_new.html
8. Baudry, A., Bungenstock, M., Mertsching, B.: Architecture of a 3D-simulation environment for active vision systems and mobile robots. In: Proceedings of First International Symposium on 3D Data Processing Visualization and Transmission, pp. 198–201 (2002)
9. Mertsching, B., Aziz, M., Stemmer, R.: Design of a simulation framework for evaluation of robot vision and manipulation algorithms. In: Proceedings of Asian Simulation Conference/The 6th International Conference on System Simulation and Scientific Computing (2005)
10. Kutter, O., Hilker, C., Simon, A., Mertsching, B.: Modeling and Simulating Mobile Robots Environments. In: 3rd International Conference on Computer Graphics Theory and Applications, Funchal, Madeira, Portugal (2008)
11. Smith, R.: Open Dynamics Engine, Version 0.8 (2007), <http://www.ode.org/>
12. Burns, D., Osfield, R.: Tutorial: Open scene graph. In: Proceedings Virtual Reality, pp. 265–265 (2004)
13. OpenSceneGraph (2010), <http://www.openscenegraph.org/projects osg>
14. OGRE 3D (2010), <http://www.ogre3d.org>
15. Mark, W., Glanville, R., Akeley, K., Kilgard, M.: Cg: A system for programming graphics hardware in a C-like language. In: ACM SIGGRAPH 2003 Papers, p. 907. ACM, New York (2003)
16. Zahedi, K., von Twickel, A., Pasemann, F.: YARS: A Physical 3D Simulator for Evolving Controllers for Real Robots. In: Carpin, S., Noda, I., Pagello, E., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2008. LNCS (LNAI), vol. 5325, pp. 75–86. Springer, Heidelberg (2008)
17. Gerkey, B., Vaughan, R., Howard, A.: The player/stage project: Tools for multi-robot and distributed sensor systems. In: Proceedings of the 11th International Conference on Advanced Robotics, pp. 317–323. Citeseer (2003)
18. Calisi, D., Censi, A., Iocchi, L., Nardi, D.: OpenRDK: a modular framework for robotic software development. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2008, pp. 1872–1877 (2008)
19. Smits, R., De Laet, T., Claes, K., Soetens, P., De Schutter, J., Bruyninckx, H.: Orocos: A software framework for complex sensor-driven robot tasks. IEEE Robotics and Automation Magazine (2008)

20. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.: ROS: an open-source Robot Operating System. In: Open-Source Software Workshop of, ICRA (2009)
21. Aziz, M., Mertsching, B., Salah, M., Shafik, E., Stemmer, R.: Evaluation of visual attention models for robots. In: IEEE International Conference on Computer Vision Systems, ICVS 2006, pp. 20–20 (2006)
22. Aziz, M.Z., Mertsching, B.: Visual Attention in 3D Space - Using a Virtual Reality Framework as Spatial Memory. In: ICINCO-RA, pp. 471–474 (2009)
23. Shafik, M., Mertsching, B.: Fast Saliency-Based Motion Segmentation Algorithm for an Active Vision System. In: Blanc-Talon, J., Bourennane, S., Philips, W., Popescu, D., Scheunders, P. (eds.) ACIVS 2008. LNCS, vol. 5259, pp. 578–588. Springer, Heidelberg (2008)
24. Shafik, M., Mertsching, B.: Real-Time Scan-Line Segment Based Stereo Vision for the Estimation of Biologically Motivated Classifier Cells, p. 89 (2009)
25. Shafik, M., Mertsching, B.: Enhanced motion parameters estimation for an active vision system. Pattern Recognition and Image Analysis 18, 370–375 (2008)
26. Lai, K., Fox, D.: 3D laser scan classification using web data and domain adaptation. In: Proceedings of Robotics: Science and Systems, Seattle, USA (2009)
27. Lim, E., Suter, D.: Conditional random field for 3D point clouds with adaptive data reduction. In: International Conference on Cyberworlds, CW 2007, pp. 404–408 (2007)
28. Lim, E., Suter, D.: 3D terrestrial LIDAR classifications with super-voxels and multi-scale Conditional Random Fields. Computer-Aided Design 41, 701–710 (2009)
29. RoboCup: German Open (2010), <http://www.robocup.org>
30. SICK AG: SICK Robot Day (2010), <http://www.sick.com>
31. Mujahad, M., Fischer, D., Mertsching, B., Jaddu, H.: Closest Gap Based (CG) Reactive Obstacle Avoidance Navigation for Highly Cluttered Environments. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2010 (2010)

OpenGRASP: A Toolkit for Robot Grasping Simulation

Beatriz León¹, Stefan Ulbrich², Rosen Diankov⁵, Gustavo Puche¹,
Markus Przybylski², Antonio Morales¹, Tamim Asfour², Sami Moisio³,
Jeannette Bohg⁴, James Kuffner⁵, and Rüdiger Dillmann^{2,*}

¹ Universitat Jaume I, Robotic Intelligence Laboratory, Castellón, Spain
`{len,puche,morales}@icc.uji.es`

² Karlsruhe Institute of Technology, Institute for Anthropomatics,
Karlsruhe, Germany
`{ulbrich,przybyls,asfour,dillmann}@ira.uka.de`

³ Lappeenranta University of Technology, Centre of Computational Engineering and
Integrated Design, Finland
`smoisio@lut.fi`

⁴ Royal Institute of Technology, Computer Vision and Active Vision Laboratory,
Stockholm, Sweden
`bohg@csc.kth.se`

⁵ Carnegie Mellon University, Institute for Robotics, USA
`{rdiankov,kuffner}@cs.cmu.edu`

Abstract. Simulation is essential for different robotic research fields such as mobile robotics, motion planning and grasp planning. For grasping in particular, there are no software simulation packages, which provide a holistic environment that can deal with the variety of aspects associated with this problem. These aspects include development and testing of new algorithms, modeling of the environments and robots, including the modeling of actuators, sensors and contacts. In this paper, we present a new simulation toolkit for grasping and dexterous manipulation called *OpenGRASP* addressing those aspects in addition to extensibility, interoperability and public availability. OpenGRASP is based on a modular architecture, that supports the creation and addition of new functionality and the integration of existing and widely-used technologies and standards. In addition, a designated editor has been created for the generation and migration of such models. We demonstrate the current state of OpenGRASP’s development and its application in a grasp evaluation environment.

Keywords: software toolkit, grasping simulation, robot modeling.

1 Introduction and Related Work

Robot simulators have accompanied robotics for a long time and have been an essential tool for the design and programming of industrial robots. Almost all

* The research leading to these results has received funding from the European Community’s Seventh Framework Programme under grant agreement n° 215821.

industrial manipulator manufacturers offer simulation packages accompanying their robotic products. These tools allow the users to program and test their applications without using the real hardware or even building it since such tools allow the analysis of behaviours and performance beforehand. In robotics research, simulators have an important role in the development and demonstration of algorithms and techniques in areas such as path planning, grasp planning, mobile robot navigation, and others. There are several reasons for the use of robot simulations. First, they allow exhaustive testing and tuning of mechanisms and algorithms under varying environmental conditions. Second, they avoid the use, or misuse, and wearing of complex and expensive robot systems. Finally, simulation software is cheaper than real hardware.

Often, simulation tools used to support research are specifically developed for particular experiments. However, there have been some successful attempts to develop general robot simulators specifically for mobile robotics.

Stage and Gazebo are respectively 2D and 3D simulator back-ends for Player [14], which is a widely used free software robot interface. Gazebo [2] in particular, implements a 3D multi-robot simulator which includes dynamics for outdoor environments. It implements several robot models, actuators and sensors. US-ARSim [5] has a similar functionality. It is a free mobile robot simulator based on a gaming engine. Microsoft provides its Robotics Studio [3], a framework for robot programming that includes a visual simulation environment. OpenHRP [7] is an open software platform with various modules for humanoid robot systems such as a dynamics simulator, a view simulator, motion controllers and motion planners. OpenHRP is integrated with CORBA, with each module, including the dynamics simulator implemented as a CORBA server. Commercial options include Webots [6], a product which has been widely successful in educational settings.

The variety of simulation tools for robotic grasping is rather limited. The most renowned and prominent one is *GraspIt!*, a grasping simulation environment [9]. *GraspIt!* includes models of several popular robot hands, implements the dynamics of contacting bodies, includes a grasp planner for a Barrett hand and has recently included a simple model of the human hand. However, *GraspIt!* has several limitations. Its rather monolithic and less modular architecture makes it difficult to improve, add functionality and integrate with other tools and frameworks. In addition, it does not provide a convenient Application Programming Interface (API), which allows script programming. Furthermore, it does not include sensor simulation.

Another existing and publicly available software framework is OpenRAVE, the Open Robotics and Animation Virtual Environment [10]. It has been designed as an open architecture targeting a simple integration of simulation, visualization, planning, scripting and control of robot systems. It has a modular design, which allows its extension and further development by other users. Regarding robot grasping simulation, it provides similar functionality to *GraspIt!* and various path planning components. It provides the models of several robot arms and

hands and allows the integration of new ones. It also enables the development of virtual controllers for such models.

In this paper we introduce a simulation toolkit specifically focused on robot grasping and manipulation. The principal design issues have been extensibility, interoperability and public availability (see Sec. 2). The core of the toolkit is an improved version of OpenRAVE, which has been enhanced with a Robot Editor and the adoption of the COLLADA file format [12] and Physics Abstraction Layer (PAL) [11] to enable standardization and flexibility (see Sec. 3). Finally, several applications which demonstrate the utility of the toolkit are presented (see Sec. 4).

This work is the result of a team funded by the European Commission within the project GRASP [21]. Within the project the toolkit is used as the main tool for reasoning and prediction of object properties as well as task constraints of manipulation and grasping activities executed by robots. Thus, it is a key tool for constructing a world model and understanding the robot environment.

2 Requirements for a Grasp Simulator

From a scientific point of view, a novel simulator for robot grasping should provide primarily a realistic simulation of dynamic properties of, at least, rigid objects and advanced contact models including soft contacts and deformable surfaces. From a practical and user point of view, it should include the models of the most popular robot hands, and provide the possibility of easily creating and adding new ones. Furthermore, it should provide realistic simulations of real actuators and sensors, which would enable the use of the same API as in their real counterparts. Regarding sensors, a grasping simulator has to provide simulations of specific grasping sensors, such as force/torque, contact and tactile. Finally, it should provide a rich and detailed visualization of simulations.

With respect to software engineering, a novel robot grasping simulator must be implemented in a modular way that enables on the one hand, an easy extension by both developers and users and on the other hand, the integration with commonly-used robotics software frameworks. In order to increase its opportunities of being adopted and used in the scientific community, the simulator should be open source and make use of open standards for file formats and other representations. In addition, the simulator should have appropriate tools to import/export robot and object models to/from standard representations.

To our best knowledge, none of the existing simulation tools and software packages fulfill all of these requirements. Therefore, we present a software toolkit for grasping simulation *OpenGRASP* [8], which is built on top of OpenRAVE to meet the requirements discussed above.

3 Toolkit Description

In order to develop a tool that meets the requirements listed in the previous section, we adopted a basic practical principle: *Do not reinvent the wheel*. This

means, first to review the existing software paying special attention to those that already meet part of the requirements and second to make use of existing open and widely-available software packages and standards.

After a wide review of existing simulators, physics engines, 3D render engines, and CAD 3D modelers, we concluded that OpenRAVE is the tool that most closely meets our requirements. So our efforts have focus on improving and extending OpenRAVE capabilities and features towards the realization of an advanced grasping simulator. In addition, we developed a robot editor allowing to create and modify new robot models. In the following sections we describe these extensions in more detail.

3.1 OpenRAVE Architecture

OpenRAVE is a planning architecture developed at the Carnegie Mellon University Robotics Institute. It is designed for autonomous robot applications and consists of three layers: a core, a plugins layer for interfacing to other libraries, and scripting interfaces for easier access to functions (see Fig. 1).

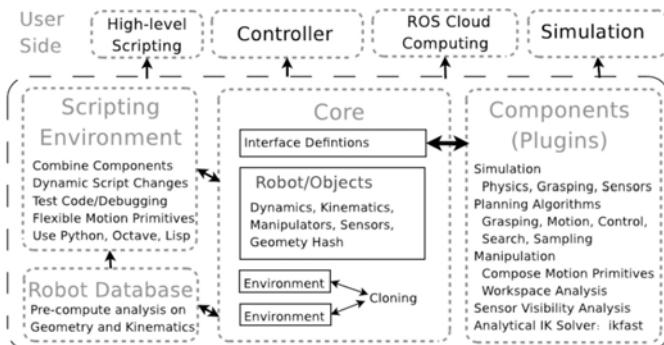


Fig. 1. OpenRAVE Architecture (Picture reproduced from [22])

The Scripting Layer enables network scripting environments like Octave, Matlab and Python to communicate with the core layer in order to control the robot and the environment. It is possible to send commands to change any aspect of the environment, read any of its information, move real robots, or change physics/collision libraries. The scripts also enable the control of multiple OpenRAVE instances across the network, thus allowing different users to independently see and interact with the environment.

OpenRAVE is designed as a plugin-based architecture, which allows the creation of new components to continuously improve its original specifications. Each plugin is an implementation of a standard interface that can be loaded dynamically without the need to recompile the core. Following this design, different plugins can be created for components such as sensors, planners, controllers or

physics engines. The core layer communicates with the hardware through the plugins using more appropriate robotics packages such as Player and the Robot Operating System (ROS) [30].

A GUI can be optionally attached to provide a 3D visualization of the environment. It periodically queries the core to update the world's view and allows the user to change the position of the objects in the scene. As viewers are provided through plugins, a single OpenRAVE instance can allow multiple viewers to communicate with multiple copies of the environment. A more detailed description of the OpenRAVE architecture can be found in [10,22].

Although many plugins are already implemented to provide basic functionality, the current grasp simulation functionality has several shortcomings. In order to make OpenRAVE suitable for our purposes, we require:

- Implementation of plugins for specific sensors used to improve the grasping capabilities of the robot.
- Implementation of more physics engines and collision checkers that help to compare and improve simulation performance.
- Implementation of a standard plugin interface for a basic actuator and implementations of different motors. This would allow us to accurately simulate the robot's arm and hand articulations.

We have taken these considerations into account in our toolkit design. We have developed two new sensor plugins to be used mainly for anthropomorphic robot hands. One is a tactile sensor, commonly used in finger tips such as in the Barrett hand, which detects and calculates the forces on the predefined sensory area and returns them as an array. The other is a force sensor, placed for example in the wrist, to measure the forces applied while grasping.

Additionally, as models for actuators were not included in OpenRAVE, we have developed a new plugin interface called ActuatorBase. Using this interface, we implemented a new plugin to simulate the motor of the arm and hands joints which can be controlled using angles, velocities or voltages.

We have created a model of the Schunk PG70 parallel jaw gripper using the tactile sensor plugin to simulate the Weiss Robotics sensor (DSA 9205) attached to each finger. We have also provided the Gripper Sensor plugin which returns the distance between fingers, their current and velocity. Finally, a model of the gripper actuator was also included which can control the velocity of the fingers and the maximum current applied.

In order to use different physics engines, we have implemented a plugin which makes use of the physics abstraction layer (PAL), which is addressed in more detail in the following section.

Visualisation is an important part of the simulation. At the moment OpenRAVE uses Coin3D/Qt to render the environment, but we are extending it to communicate with Blender given that our RobotEditor (see Section 3.4) is developed on top of it. Because both Blender and OpenRAVE provide a Python API, it is possible to use Blender as a front-end for not just visualization, but also for calling planners, controlling robots, and editing robot geometry.

3.2 Physics Simulation

Nowadays, there exist many available physics engines, both commercial and open-source. Some of them are high-precision engines that require higher computational power while others sacrifice this accuracy to work in real time. The methods they use to simulate physics are also different. Some of them use penalty methods, some rely on physical laws using constraint equations, and others use methods based on impulses.

None of these engines are perfect, they all have advantages and disadvantages which makes it very difficult to decide which one to use for a simulator. It basically depends on what we want to simulate in addition to what the application of the simulator will be.

The Physics Abstraction Layer (PAL) [11] is a software package created by Adrian Boing that saves us from having to decide at the start what physics engine to use. This layer provides an interface to a number of different physics engines allowing us to dynamically switch between them. This functionality adds incredible flexibility to our simulator offering us the possibility to, depending on our specific environment and use, decide which engine would provide us the best performance [14]. Using their interface, it is also possible test and compare our own engines with existing ones.

The OpenRAVE Physics Engine interface allows the simulator to run using different engines. It also has an interface to implement different collision checkers. Each one of them has to be created as a plugin, extending either the PhysicssEngineBase or the CollisionCheckerBase class. The basic version of OpenRAVE only offers the ODE (Open Dynamics Engine) implementation within the oderave plugin. We have created a new plugin to use PAL, called palrave. This plugin is able to initialize PAL with the specific engine we want to use, without the need of creating different plugins for each one of them.

3.3 COLLADA File Format for Robot Models

For the storage of models of robot manipulators and hands, we were looking for an open, extensible and already widely accepted file format that supports the definition of at least kinematics and dynamics. This is necessary in order to enable the exchange of robot models between supporting applications, leading to greater flexibility in the selection of appropriate tools. Another important aspect was the ability to convert to and from other formats. Among the large variety of file formats for 3D models, there are only a few that are both public domain and not limited to storing only geometric information. Typically, a simulator environment does not only rely on geometric structures but also, for instance, on information about dynamics, kinematics, sensors and actuators of the robot.

Its acceptance as an industry standard and its wide distribution (3D Studio, Blender, OSG, OGRE, Sony, etc.), in addition to a clear and extensible design, led to the choice of COLLADA [12] as the preferred file format for the simulator. In addition, there are open source frameworks available that facilitate integration into new applications.

Since version 1.5, the standard contains many useful constructs dedicated to describing kinematic chains and dynamics that can be used directly for the description of robot models. COLLADA is an XML-based file format that enables and encourages developers to extend the specification to their needs without having to violate the underlying schema definition.

In order to support specific robot features like sensors and actuators, we have used this mechanism to extend COLLADA partially using the original OpenRAVE file definition. These additions are specific to the simulator and are hidden to all other applications so that compatibility remains guaranteed. So far, basic support for COLLADA import and export has been included in the simulator.

3.4 Robot Editor

With the creation of a simulator for grasping the need also arises for a large data base of geometrical, kinematic and dynamic models of robot arms and manipulators. To fill this gap, the development of a modeling tool, the Robot Editor, has been started. Its main goal is to facilitate modeling and integration of many popular robots. The development is driven by the following key aspects:

- **Geometric modeling:** The creation of new robots models requires a tool that excels in modeling of the geometric components (i.e., meshes).
- **Semantic modeling:** Even more important is the ability to allow the description of semantic properties, such as definitions of kinematic chains, sensors and actuators, or even specify algorithms.
- **Dynamics modeling:** Another necessary aspect is the ability to define physical attributes of the robot's elements. At the moment, the focus lies on the dynamics of rigid bodies.
- **Conversion:** Robot models usually come in a variety of different file formats. The modeling tool needs to be capable of processing these formats and converting them into the COLLADA standard. GraspIt! files in particular, being an already widely-used standard with many conform models available, should be readily usable by the simulator.

To our knowledge, there is no existing solution openly available that could meet all these requirements. Therefore, we decided to develop a new modeling tool based on available open source software. The conceptual design of the Robot Editor hence relies on two techniques: on the one hand the open data format COLLADA and on the other hand on the open source project Blender [15]. Blender is a very versatile, powerful and extensible 3D editor that has been chosen because of its convenient 3D modeling capabilities and the built-in support for many CAD file formats and rigid body kinematics. Furthermore, it can be easily extended via a Python scripting interface and offers high-quality ray tracing.

Blender itself, however, lacks the functionality and the appropriate interface for the convenient definition of robot components. In addition, conversions between certain file formats need to be improved or implemented, namely the import of the COLLADA format and GraspIt! robot models.

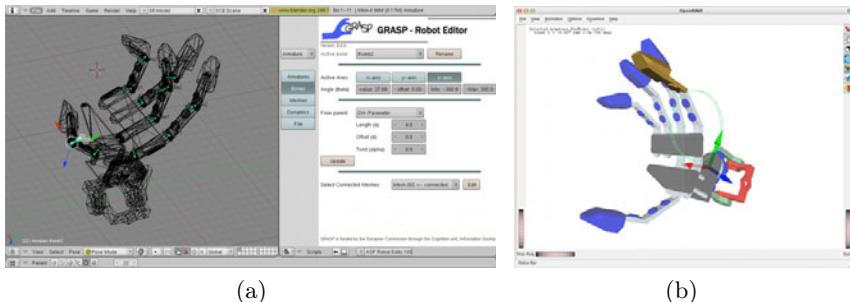


Fig. 2. Modeling the Karlsruhe anthropomorphic robot hand. a) The user interface of the Robot Editor and b) Screenshot of the complete model in the simulator.

The scripting interface mechanism mentioned above allowed us to build the modeling tool on top of Blender. On the scripting level, one gains access to all relevant data structures. The robot model can be augmented by the required data structures and conserved within the native file format. The scripting mechanism also allows the creation of user-interface that is highly specialized for use in robotics (see Fig. 2(a)). For instance, you can define kinematics via Denavit-Hartenberg parameters. In the long run, the Robot Editor will provide interfaces for essential robotics algorithms, such as the computation of dynamics characteristics from the geometric meshes and conversions between kinematics representations. Adjacency information of joints and the impact of joint movements to the robot are additional computational information which, in the future, will be useful for developers' planning algorithms.

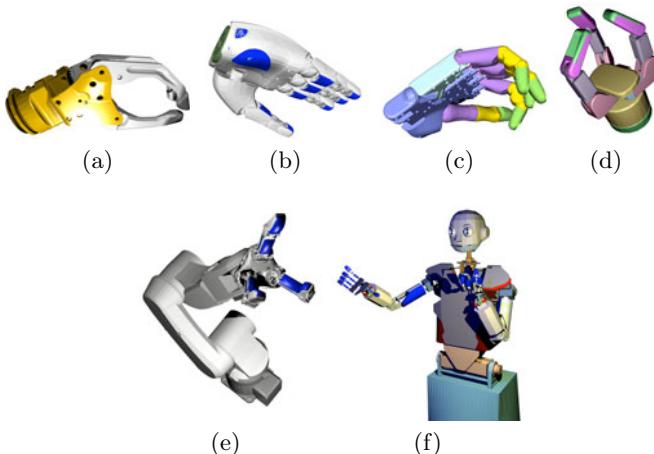


Fig. 3. Different robot models generated with the Robot Editor (ray-traced images). (a) A myoelectric upper extremity prosthesis of Otto Bock, the Schunk (b)SAH and (c)SDH hands, (d) the Shadow hand, (e) the Barrett hand and the Kuka KR5 sixx R850 arm, and (f) the humanoid robot ARMAR-III.

In the current Blender version (2.49), COLLADA support [7] is limited to documents in the older specification 1.4 which excludes the newly introduced kinematics and dynamics. The additional data required by the simulator also needs to be included in the resulting document. This led to the further development of COLLADA compatibility which now enables the Robot Editor to create valid documents suitable for simulation. Fig. 2(a) shows a functional model of the Karlsruhe anthropomorphic hand [16] modified using the Robot Editor and Fig. 2(b) the resulting COLLADA file loaded into the simulator.

Robot Models. As stated in Section 2 it is of great importance to have models of the most popular robot hands included in the toolkit. The modeling capabilities of the Robot Editor already enable quick and easy creation of new models. So far, a selection of robot hands has been transformed into COLLADA 1.5 for use within the simulator (see Fig. 3). In addition to these new models, there are various models available in the older XML file format which is still supported.

4 Applications

4.1 Planning and Grasping

Using the functions provided by OpenRAVE, we can easily build a set of stable grasps and quickly get our robots to manipulate various objects in their environment. Figure 4 shows the grasp simulation process by analyzing the contact points between the robot and the target object. Recently Przybylski et al. [26] used OpenGRASP to develop a new grasp planning method based on the Medial Axis of objects to reduce the search space of candidate grasps. The algorithm exploits structural and symmetry information contained in the Medial Axis in order to reduce the search space for promising candidate grasps.

In order to get the robot to autonomously manipulate objects in the environment, we would need an inverse kinematics solver that can quickly map grasp locations into robot configuration joints. Recently, OpenRAVE started providing an analytical inverse kinematics equation solver called *ikfast*. With it we can generate C++ code that can return all possible IK solutions while simultaneously handling degenerate cases. By combining the automatically generated grasp sets, inverse kinematics solvers and planners, robots developed in our RobotEditor can manipulate everyday objects.

Grasping Known Objects. Using the planning and grasping capabilities of the simulator, we have successfully grasped known objects in the real world with the robotic platform consisting of the Karlsruhe Humanoid Head (see [27][28]) and a Kuka KR5 sixx R850 arm equipped with a Schunk Dextrous hand 2.0.

A complete description of known objects is provided by the KIT object model database [29]. The object description consists of geometry described by a mesh and physical properties like mass. Using the grasp simulation process, different grasp hypotheses are tried out off-line. Only those that result on force closure are then saved to a data base. Using an active vision system [23] in combination

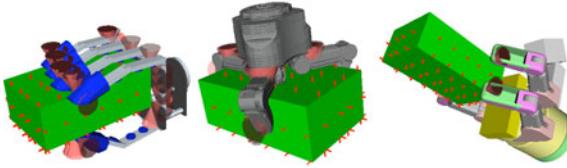


Fig. 4. Grasping simulated for several robot hands

with rigid point set registration [24], the robot can explore the current scene and recognize where and in which pose these known objects are (see an example in Fig. 5). This information is sent to OpenGRASP which reproduces the state of the environment. Using the planning plugins, the selected force closure grasps are executed. The ones that collide with the environment or that are not reachable given the current configuration of the robot are discarded.

Grasping Unknown Objects. We have also used OpenGRASP to manipulate unknown objects with the above mentioned robotic platform. This task is more complex since no information about the objects is available. Thus, the grasp simulation process cannot be executed off-line. The robot uses the above mentioned active vision system [23] to explore the scene and segment unknown objects. From the reconstructed stereo point cloud, a mesh is created as an approximation of the object's geometry. This mesh and its location are sent to the simulator which uses an available library for decomposing it into a reduced number of convex hulls to simplify its collision detection. Given this mesh, there is an unlimited number of grasp configurations that can be applied to the object and tested for force closure. For reducing the search space and thereby online processing time, we apply a grasp point detection method [25] to the segmented object. The simulator then tries only grasp configurations with the fingers of the robotic hand being aligned with the grasp points. Those that are in force closure are saved in a database before being executed with the planner to check for collisions and reachability. An example of the successful grasping of an unknown object is shown in Fig. 6.

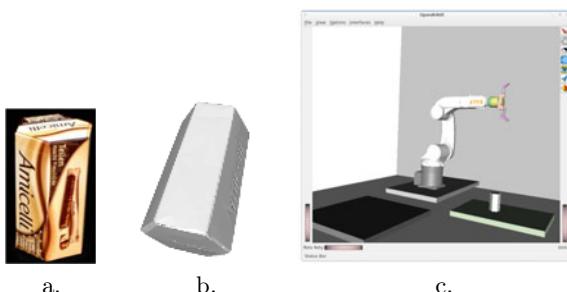


Fig. 5. Grasping a known object: a) real object, b) mesh geometry and c) example of a scene visualization after its recognition

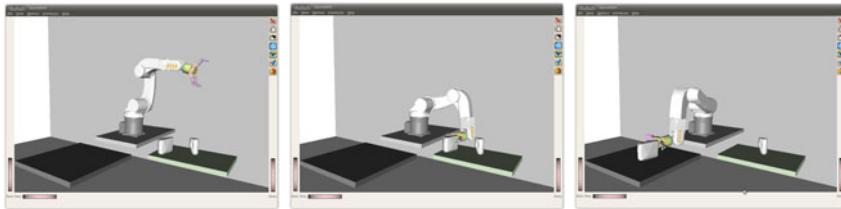


Fig. 6. A successful grasp of an unknown object with the Kuka KR5 R850 arm and a Barret hand

5 Conclusion and Future Work

In this paper we have presented a fully operational simulation toolkit for robot grasping and manipulation. Its main design principles are extensibility, interoperability and public availability. In its development we have used existing and widely-available components to ensure its standardization and easy adoption. We have also emphasised providing additional tools and features that provide users with a fast start to enhance utility through features such as the robot editor based on Blender, COLLADA file format, a Physics Abstraction Library, and models of existing robot hands. The utility of OpenGRASP is demonstrated through a series of applications cases.

In future, we are looking to improve the toolkit in three directions. First a novel contact modelling for soft contacts and body deformation is being implemented and validated. It will be included as a library to allow the modelling of complex contact interactions. Second, OpenGRASP is being extended to offer full support for ROS thus offering a common control interface for simulated and real robots. Finally, several interfaces are being created in order to offer the user several options to visualize and record the simulated systems.

References

1. Gerkey, B., Vaughan, R., Howard, A.: The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In: 11th International Conference on Advanced Robotics (ICAR 2003), Coimbra, Portugal, pp. 317–323 (2003)
2. Gazebo, <http://playerstage.sourceforge.net/index.php?src=gazebo>
3. Microsoft Robotics Studio, <http://msdn.microsoft.com/robotics>
4. The Player Project, http://playerstage.sourceforge.net/wiki/Main_Page
5. USARSim, <http://sourceforge.net/projects/usarsim/>
6. Webots, <http://www.cyberbotics.com/>
7. OpenHRP, <http://www.is.aist.go.jp/humanoid/openhrp/>
8. OpenGRASP, <http://opengrasp.sourceforge.net/>
9. Miller, A., Allen, P.: GraspIt!: A versatile simulator for robotic grasping. IEEE Robotics & Automation Magazine 11, 110–122 (2004)
10. Diankov, R., Kuffner, J.: OpenRAVE: A Planning Architecture for Autonomous Robotics. Technical report, Robotics Institute, Pittsburgh, PA (2008)
11. PAL (Physics Abstraction Layer), <http://www.adrianboeing.com/pal>

12. COLLADA, <http://collada.org>
13. Khronos Group, <http://www.khronos.org/>
14. Boeing, A., Bräunl, T.: Evaluation of real-time physics simulation systems. In: 5th international Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia (GRAPHITE 2007), Perth, Australia, pp. 281–288 (2007)
15. Blender, <http://Blender.org>
16. Gaiser, I., Schulz, S., Kargov, A., Klosek, H., Bierbaum, A., Pylatiuk, C., Oberleand, R., Werner, T., Asfour, T., Bretthauer, G., Dillmann, R.: A new anthropomorphic robotic hand. In: IEEE/RAS International Conference on Humanoid Robots (Humanoids), pp. 418–422 (2008)
17. Illusoft: Blender Collada Plugin, <http://colladablender.illusoft.com/cms/>
18. Blender - Google Summer of Code (2009),
<http://www.blendernation.com/blender-google-summer-of-code-2009/>
19. OpenCOLLADA, <http://www.opencollada.org/>
20. Otto Bock, http://www.ottobock.de/cps/rde/xchg/ob_de_de/hs.xsl/384.html
21. GRASP Project, <http://www.csc.kth.se/grasp/>
22. OpenRAVE website, <http://openrave.programmingvision.com/>
23. Björkmann, M., Kräigc, D.: Active 3D scene segmentation and Detection of Unknown Objects. In: International Conference on Robotics and Automation (ICRA 2010), Anchorage, Alaska, USA (2010)
24. Papazov, C., Burschka, D.: Stochastic Optimization for Rigid Point Set Registration. In: Bebis, G., Boyle, R., Parvin, B., Koracin, D., Kuno, Y., Wang, J., Pajarola, R., Lindstrom, P., Hinkenjann, A., Encarnaçāo, M.L., Silva, C.T., Coming, D. (eds.) ISVC 2009. LNCS, vol. 5876, pp. 1043–1054. Springer, Heidelberg (2009)
25. Richtsfeld, M., Vincze, M.: Grasping of Unknown Objects from a Table Top. In: ECCV Workshop on 'Vision in Action: Efficient strategies for cognitive agents in complex environments', Marseille, France (2008)
26. Przybylski, M., Asfour, T., Dillmann, R.: Unions of Balls for Shape Approximation in Robot Grasping. To appear in IROS (2010)
27. Asfour, T., Regenstein, K., Azad, P., Schröder, J., Vahrenkamp, N., Dillmann, R.: ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control. In: IEEE/RAS International Conference on Humanoid Robots (Humanoids), pp. 169–175 (2006)
28. Asfour, T., Welke, K., Azad, P., Ude, A., Dillmann, R.: The Karlsruhe Humanoid Head. In: IEEE/RAS International Conference on Humanoid Robots (Humanoids), pp. 447–453 (2008)
29. Kasper, A., Becher, R., Steinhaus, P., Dillmann, R.: Developing and Analyzing Intuitive Modes for Interactive Object Modeling. In: International Conference on Multimodal Interfaces (2007), <http://i61www.ira.uka.de/ObjectModels>
30. ROS, <http://www.ros.org/wiki/>

NERD

Neurodynamics and Evolutionary Robotics Development Kit

Christian Rempis, Verena Thomas, Ferry Bachmann, and Frank Pasemann

Institute of Cognitive Science
University of Osnabrück, Germany

{christian.rempis, frank.pasemann}@uni-osnabrueck.de,
thomas@cs.uni-bonn.de, bachmann@informatik.hu-berlin.de
<http://ikw.uni-osnabrueck.de/~neurokybernetik/>

Abstract. The aim of Evolutionary Robotics is to develop neural systems for behavior control of autonomous robots. For non-trivial behaviors or non-trivial machines the implementation effort for suitably specialized simulators and evolution environments is often very high. The *Neurodynamics and Evolutionary Robotics Development Kit* (NERD), presented in this article, is a free open-source framework to rapidly implement such applications. It includes separate libraries (1) for the simulation of arbitrary robots in dynamic environments, allowing the exchange of underlying physics engines, (2) the simulation, manipulation and analysis of recurrent neural networks for behavior control, and (3) an extensible evolution framework with a number of neuro-evolution algorithms. NERD comes with a set of applications that can be used directly for many evolutionary robotics experiments. Simulation scenarios and specific extensions can be defined via XML, scripts and custom plug-ins. The NERD kit is available at nerd.x-bot.org under the GPL license.

Keywords: Simulation, Evolutionary Robotics, Neuro-evolution.

1 Introduction

The development of behavior controllers for autonomous robots with methods from the field of evolutionary robotics is by now well established [4][5][7]. Instead of analytical construction, evolutionary techniques – sometimes together with other learning methods – are used to search for suitable solutions. Inspired by the properties of biological brains, artificial recurrent neural networks turned out to be effective control structures, especially for more complex behaviors. With respect to evolutionary algorithms they are also suitable because of their simple (syntactical) structure that allows the construction and adaptation of controllers with a set of simple modification operators.

Usually controllers for physical robots are evolved in simulation, (1) because evolutionary approaches and learning methods require experiments with many comparable iterations, (2) to protect the physical robot from wear-out or evolved,

potentially harmful control, and (3) to speed up evolution by running the simulation faster than real-time, for instance on a computer cluster. Furthermore (4) simulations are useful to reduce cost, e.g. by evolving multi-agent systems with way more robots than are actually available in a lab, or by co-evolving morphological aspects that may lead to structural enhancements to the physical robots without the need to build all intermediate steps in hardware.

However, using a simulator for evolution may also lead to problems: Evolution usually optimizes its solutions exploiting all the details of a simulated robot, including modeling differences and simplifications. Such controllers are often difficult or impossible to transfer to the targeted physical robot, because the differences are too large. Therefore critical elements of the robot simulation have to be modeled accurately, often very specifically for a given robot or experiment. But this is sometimes difficult to accomplish with closed-source applications or with simulators having only a fixed set of components. Furthermore, evolution of controllers is usually an iterative process in which the scenarios, environments and the robots are iteratively modified and changed until a suitable experiment is defined. Thus, a simulator should be simple to adapt and it should be extensible also by very specific models. Hereby it should simplify the reuse of frequently used program parts to speed up the simulator development.

The evolution or optimization techniques used to evolve neuro-controllers can – and should – not be fixed because the development of non-trivial control networks often requires the utilization of new or adapted algorithms. Keeping the actual evolution algorithm open to be chosen freely by the user, while allowing the reuse of an entire integration framework – such as logging, statistics calculations, configuration, user interfaces, computer cluster support, evaluation and analysis tools – facilitates the rapid realization of new algorithms.

The *Neurodynamics and Evolutionary Robotics Development Kit* (NERD) [15], described in this publication, is a framework to rapidly create simulations and applications in the context of evolutionary robotics. Its design focuses on extensibility and flexibility to allow for a wide variety of control applications without the need to re-implement commonly used features. NERD is open source (under GPL license) and comes with a number of ready-to-use applications for simulators, neural network analysis and evolutionary algorithms. Providing libraries instead of only fixed applications allows users to build upon existing code to create their own, specialized neural control applications, without being forced to use a specific physics engine, rendering engine, controller type, genome representation or evolutionary algorithm. However, the provided applications themselves are already flexible enough to allow the definition of complex evolutionary experiments without or with only little programming skills.

The library features a flexible application core with plug-in support, object management and network interfaces, a simulation library with exchangeable physics engine, implementations of neuro-evolution algorithms, an extensible neural network model, an interactive editor for neural networks, computer cluster support, and optional graphical user interfaces for visualization and configuration. Details are described in the remaining chapters.

The NERD kit is under continuous development since 2007 and is used for experiments in scientific and educational areas, such as motion control of humanoid robots, biologically inspired behaviors of animats and in teaching evolutionary robotics and neuro-dynamics.

2 Concepts and Library Organization

2.1 Library Organization

The NERD library was primarily developed for the structure evolution of neural networks serving as controllers for physical robots. Nevertheless the library was designed to be as universal and extensible as possible. This is important, because at the beginning of a project it is often difficult, if not impossible, to know all eventual requirements in advance. In the field of evolutionary robotics, this is especially true for the accurate simulation of rigid body dynamics, the optimization algorithms and the neural network models.

The libraries are written in standard C++ and make use of the QT library (Version 4.5) [9]. QT is a C++ framework, that hides platform dependent functionality – like networking, file management, graphical user interfaces and

Table 1. NERD Libraries and Dependencies

Library	Description	Dependencies
core	Basic core library, GVR, events, plug-ins, none configuration, logging, basic GUI	
simulator	Simulation components, simulation visualization and physics abstraction layer	core
odePhysics	ODE implementation of the physics abstraction layer	core simulator libOde (external)
ibdsPhysics	IBDS implementation of the physics abstraction layer	core simulator libIbds (external)
neuralNetwork	Neural network models and components	core
networkEditor	Graphical network editor and analysis tools	core neuralNetwork
evolution	Evolutionary algorithm framework	core
evaluation	Components to evaluate individuals of the evolution	core evolution
neuroEvolution	Neuro-evolution algorithms	core evolution neuralNetwork
neuroSimEvaluation	Evaluation library for neural networks on simulated agents	core simulator evaluation

OpenGL rendering – behind a platform independent API. This allows NERD applications to be compiled for Windows, Linux and Mac without modifications. Apart from optional physics libraries for the simulation of rigid body dynamics no additional external libraries are required.

The framework can be extended by additional, custom libraries or by plug-ins to be loaded at runtime to extend applications with custom elements. The NERD kit also comes with a number of concrete applications (Sect. 3), that bundle the functionality of the libraries for specific tasks. Users of the NERD kit may either implement custom applications, or use the enclosed applications and extend their functionality via plug-ins, XML and scripts to fit the application to a specific scenario.

2.2 Basic Features

System Core. All NERD applications are built on top of a single, global core object, accessible as a singleton from everywhere in the code. All objects, managers and extensions should be registered during their creation at the core to be initialized, set up, shut down and destroyed properly. Objects can be registered as *global* objects by giving them unique names. Such objects are accessible via the core from any other part of the code using their name. This makes it easy to add and use shared objects.

Global Variable Repository. The global variable repository (GVR) implements a blackboard architecture that allows components of an application to exchange information without actually knowing each other directly. This makes it easy to add or exchange software components, even if they are very different from each other, because the actual class or interface type of each component is not of relevance (Fig. 1). What matters is only the variable interface provided to the GVR. All software components can register local variables at the GVR, hereby giving a global name to the local variable. The names are organized like directory names, dividing the global name-space into a hierarchy of smaller name-spaces. During startup of the application, all components may *bind* to variables of the GVR by name, hereby obtaining a pointer to each desired variable object. With this pointer a component can read or write to and from this variable during the rest of the application lifetime without performance loss. The GVR is registered as global object at the core.

Variables in the GVR also implement a notification mechanism. With this, objects register as observer to selected variables and can react on variables changes immediately.

Each variable object also provides methods to read and write the variable using strings. Therefore loading, saving, displaying, transmitting and manipulating variables is possible in a type-independent unified way. New components do not need to implement specific mechanisms for variable manipulation and configuration, because this is already provided by the NERD standard tools.

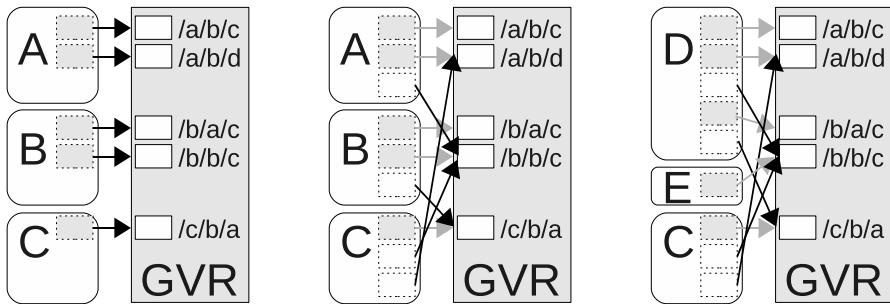


Fig. 1. Global Variable Repository. Left: Three components registering local variables at the GVR. Mid: Components bind to variables of other components using the global names of the variables (black arrows). Right: Components A and B are replaced by D and E. C works without changes, even if the replaced components are very different.

Events. NERD applications are event driven. These events are provided by the so called event manager, that is registered as global object at the core. Any component can use the event manager to create global events or to register for events of other components. Events can be triggered by any component, leading to a notification of all components registered for this event. Thus, all components are loosely coupled through events and contribute their functionality as reaction to these. Adding a new component is simple: The component just has to register for a specific existing event to do the right thing at the right time without even knowing the entire context of the application. NERD applications typically consist of a number of such separate components – connected by the GVR and global events – and an application dependent event loop, that triggers the events in the correct order.

Standard Graphical User Interface. The core library provides a set of optional GUI elements to control, visualize, customize and debug applications. These include a property editor to load, save, modify and observe variables in the GVR, plotters to observe variables over time and loggers.

2.3 Simulation

The optional simulation library was designed to provide all common tools and mechanisms to simulate autonomous mobile robots in dynamic environments.

Physics Abstraction Layer. Special emphasis has been placed on the accuracy of the robot simulation. The rigid body dynamics of a simulation usually is processed by external physics libraries [1][2], that all have their strengths and weaknesses concerning accuracy and performance. Because our goal is the evolution of neuro-controllers for physical robots, the simulation of the actual behaviors has to be as accurate as possible, so that evolved controllers can be

transferred to the physical machine. Therefore NERD does not depend on a single, fixed physics engine. Instead all simulated objects are represented within a physics independent abstraction layer. The concrete implementation of the physics simulation and collision handling is detached from the rest of the simulation system, so that a replacement of the physics engine is possible with little impact on the other parts of the system. NERD therefore remains open for enhancements of existing engines, new approaches or commercial systems when this is required for the simulation of a particular scenario.

Currently the abstraction layer provides prototypes for boxes, spheres, capsules, rays, different kinds of joints and motors, light sources, and sensors for light, acceleration, forces, contacts, joint angles and distances.

Environment XML. Simulation scenarios are described in XML files that are loaded at runtime. The XML describes and configures all objects in the simulation. Different kinds of randomization techniques are supported to randomize all object parameters in a deterministic way, i.e. that all individuals of a generation during evolution face the same randomized parameter sets. The XML can also be used to set parameters in the GVR, to allow or prevent collisions between objects, and to add so called *collision events*. The latter are events that are triggered when a certain set of objects collide, which is a useful information for fitness functions.

Agent Control. Agents are defined by grouping simulated objects together. Each agent automatically collects all motors and sensors of its body and provides an interface for behavior controllers. These interfaces provide a simple collection of typed, named variables, that can be accessed by plug-ins to read out the sensors and to control the actuators. Furthermore these variables are available through the GVR and thus allow manipulations in many different ways. With this, in principle, any kind of control is possible by adding simple plug-ins with control code, mapping the input of the interface to its output with any

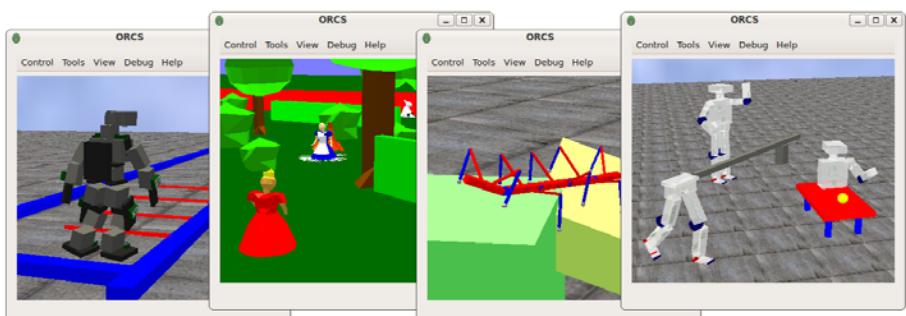


Fig. 2. Example Environments: (1) A-Series humanoid walking scenario (2) A-life experiment “Alice in Wonderland” (3) A walking machine with variable number of segments (4) Experiments with the modular M-Series humanoid

desired approach. However, the main control paradigm in NERD is the control with recurrent neural networks, which is supported by an extensible framework (Sect. 2.5). There the neurons of the input and output layer of a neural network are automatically connected to the interface variables, mapping between native variable values and neural activations.

A useful standard tool in NERD are the so called *control sliders*, which allow the control of any actuator of an agent using sliders. This helps to test robot models or to get a feeling for sensor and motor ranges and the overall body behavior of a robot.

The NERD simulation can also be used as pure simulation environment, running the control and perception of a robot in separate applications. This is beneficial if one wants to study the behavior of a physical robot with respect to an already given control application. In this case NERD and its agents may be controlled via UDP or other network protocols. NERD comes with a powerful own protocol (Sect. 2.4), that allows to read, stream and write all variables of the GVR and to react to and trigger events. The addition of other communication protocols via plug-ins helps to fit the applications to very specific interface demands.

Visualizations and Camera Support. NERD simulations can be visualized with OpenGL (Fig. 2). NERD supports any number of cameras, either stationary or attached to objects of the simulation, for instance to follow an agent. Each camera allows a separate configuration, e.g. to set the viewing angle, zoom factor, or position. Cameras may be used to observe a simulation scenario from different perspectives, but can also be applied as camera sensors of a robot. Cameras may also output their images to files, so that external vision applications can process the images like normal camera images.

To make the visualizations more lifelike, objects in NERD can have textures and special visualization masks. These masks can have complex shapes, but are considered only for visualization purposes. Thus, the robot will be simulated with simple body shapes for high performance, while the cameras still show the complex shapes of the robot, which is often important for a reliable features detection during vision processing.

2.4 Interfaces and Protocols

Plug-Ins. Existing NERD applications are designed to be extended by plug-ins written in C++. A plug-in just has to be added to a special folder to be loaded at startup, hereby simply attaching its objects to the core, the GVR and the event manager. This allows the extension of an application with arbitrary code. The configuration of plug-ins is simplified with a framework for command line options and the GVR. Thus plug-ins do not need to implement custom configuration interfaces and can simply use the provided standard tools. Plug-ins are frequently used to add custom (motor and sensor) models to the simulation, to extend the neural network model, to add learning rules or evolution algorithms, or to extend the application by communication protocols to interface with external programs.

NERD Standard Protocol (NSP). The NSP is a UDP based communication protocol that allows external applications to access the main coordination parts of NERD, namely the GVR and events. The UDP protocol allows a fast communication on the same computer, where package losses are not to be expected. For control over a network the protocol should be wrapped with the slower TCP protocol to prevent package losses. With the NSP protocol clients can read and write any variable of the GVR and therefore control all main functions of the NERD system. Clients can register to get informed about any change of a variable or to stream its content continuously. Clients can trigger events to drive a NERD application, or receive notifications about triggered events. With this, interacting applications may synchronize with each other and exchange information. Details of the protocol are provided at the website [15].

Specialized Protocols. The integration of existing applications is achieved by implementing their communication protocol. Because of the transparency of the GVR these protocols can access all relevant parts of the NERD environment and simply serve as a mediator between NERD and the communication protocol. With such interfaces, the evolution application may use external simulators (such as the YARS simulator [14]), or a NERD simulator may receive commands from external control programs.

2.5 Neural Network Library

The neural network library provides a flexible, object oriented framework for recurrent neural networks. If the standard model is not sufficient, it can be replaced by custom neuron models via plug-ins. The graphical user interfaces, plotters and manipulation tools even work for most of such extensions. This supports rapid prototyping of network models and the use of the model of choice.

Recurrent Neural Network Model. The NERD network model comprises neurons, synapses, neuron-groups and neuro-modules. The activation and transfer function of each neuron can be chosen and configured separately. The same is true for the model of the synapses. This enables local learning rules at the synapse level or higher-order synapses, i.e. synapses modulating other synapses. Extensions of the set of activation, transfer and synapse functions to adapt the neuron model to specific needs are possible via plug-ins. Each synapse supports the optional control of their execution order, which is useful in feed-forward structures to reduce the delays of neural subnetworks.

Neuron-groups and neuro-modules are part of the *constrained modularization approach* [10] and are used to group and constrain parts of neural networks, as is required by the evolution algorithm ICONE [10]. This allows the automatic enforcement of, for instance, symmetries, topological relations or structure repetitions.

Neural Network Editor. The design of neuro-controllers is supported by a graphical network editor. This editor allows the construction and examination

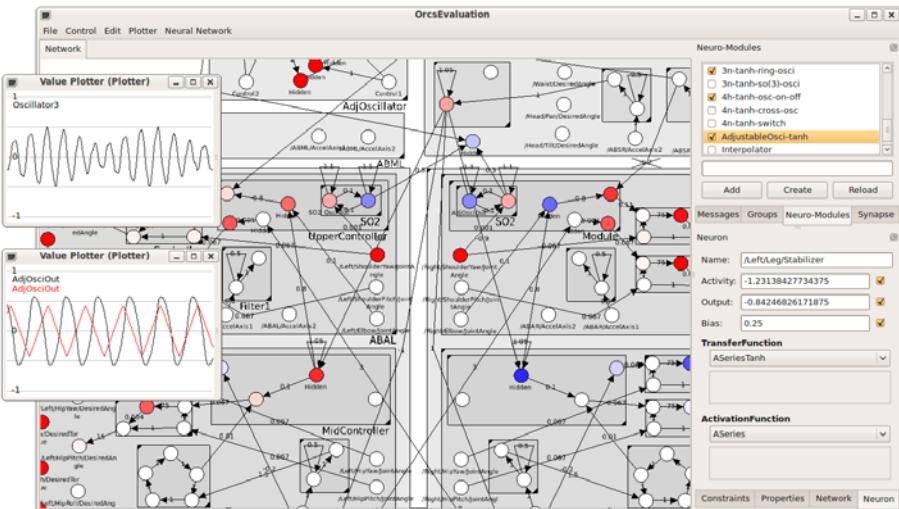


Fig. 3. Neural Network Editor

of recurrent neural networks with mouse and keyboard. The editor assists the user with layout and visualization aids to keep even large networks analyzable. Neurons and modules may be named to indicate their function or role. Visual feedback about bias values, interface neurons and other properties, as well as a search function for neurons and synapses by name or weight, help to cope even with large and complex networks.

To ease the construction of neural networks the editor also provides a library of neural building blocks, used to extend a network by functional units instead of only single neurons. Known neural structures, such as oscillators, memory units, filters or controllers, can be reused with this library even without a deep understanding of neuro-dynamics.

The network editor is also a valuable tool to analyze and understand the usually complex evolved neuro-controllers: The activation of each neuron is visualized on-line to support an intuitive comprehension of the network dynamics. Furthermore, pruning experiments – the selective disabling of synapses – in running systems are possible, allowing the experimenter to find the relevant synapses for a behavior in a recoverable way. Also, different plotter types allow the visualization of the activation of any neuron to investigate the role and relation of neurons in the network and the dynamical properties of certain network areas.

Networks in the editor can be exported in numerous output formats, such as scalable vector graphics or native code for physical machines. Additional export formats, e.g. as execution code for a specific robot, may be added via plug-ins.

2.6 Evolution Library

Evolution Framework. The evolution library provides a framework to create custom evolution scenarios. The genomes are not restricted to a specific neural

network representation. Thus, the genome representation can be adapted to a certain evolution algorithm or problem, as for instance the co-evolution of body morphology and controller network. Custom evolution algorithms and evaluation strategies can be embedded in the system with little effort, enabling rapid testing of diverse approaches. The framework supports the parallel usage of different evolution algorithms, as well as co-evolution and island models. The evaluation strategy may be freely implemented, e.g. by evaluating each individual separately, by cloning a controller to multiple agents (as in swarms) or by evaluating individuals of a generation or different populations together in the same evaluation scenario (as in predator-prey experiments or with heterogeneous swarms). Also the set of selection methods is extensible via plug-ins and works in combination with most evolution methods. During evolution (customizable) statistics are calculated on the generation and individual level. Statistics information is visualized with the user interface and is logged to files during evolution to allow a detailed analysis of the evolution course.

Fitness Functions. Fitness functions have to be adapted very often until a good strategy is found. Therefore fitness functions can not only be added via C++ plug-ins, but can also be scripted directly in the application (ECMAScript [3]). In both cases the user has full access to all variables of the GVR and events, and herewith to all properties of the simulated environment. These variables can also be modified from within the fitness functions, which allows complex interactions between fitness functions and evaluation scenarios.

3 NERD Standard Applications

The NERD kit comes with a number of pre-build applications for a variety of evolution experiments.

NerdSim is a simulator controlled with the NERD Standard Protocol (NSP) (Sect. 2.4). Thus, its execution is controlled by an external application. The simulation scenario is described with XML. Agents in the simulation are controlled either by an external controller via NSP, or with custom control plug-ins.

NerdNeuroSim is the standard stand-alone simulator for neuro-evolution. Agents are controlled with neuro-controllers or with custom controller plug-ins. This simulator uses the evaluation library to rate the performance of behavior controllers with fitness functions. The simulator also integrates the neural network editor to allow the construction and analysis of neuro-controllers running on the simulated agents.

NerdNeuroEvo is a neuro-evolution application combining the functionality of NerdNeuroSim with the evolution library. The application provides several neuro-evolution algorithms, including ENS3 [8][6], NEAT [12] and ICONE [10]. It features a built-in editor for fitness functions, logging, plotting and real-time

observation of evaluations. The NerdNeuroEvo application is especially suited for *interactive neuro-evolution*: The user permanently readjusts the evolution process depending on the current state of the evolution and counteracts undesired local optima by changing the fitness function, the scenario or other aspects of the neural network evolution.

NerdNeuroEvo can also be used with external simulators via network protocols. In this case only the evolution and execution of the neural networks takes place in NerdNeuroEvo, while the physics simulation of the agents is processed in the external simulator.

NerdClusterNeuroEvo is a neuro-evolution application that distributes the evaluation of individuals to a computer cluster using the Sun Grid Engine [13]. The controller evaluation is done in separate instances of NerdNeuroSim, thus NerdClusterNeuroEvo does not include an own simulator. Using a computer cluster speeds up evolution significantly and facilitates interactive evolutions, because the results of parameter or experiment adoptions can be observed much faster.

4 Conclusions

The *Neurodynamics and Evolutionary Robotics Development Kit* (NERD) is a free collection of open source libraries and applications for evolutionary robotics experiments. The library provides (1) a core library designed for rapid prototyping and simple integration of new components, (2) a simulator library to simulate arbitrary robots with different physics engines in dynamic environments, (3) a neural network library with a customizable neural network model and a graphical neural network editor, and (4) a general evolution framework with libraries for evolution and evaluation of neural networks with different neuro-evolution algorithms. NERD supports run-time extensions via plug-ins, which allows users to extend existing NERD applications with custom components, such as specific motor and sensor models. In combination with the physics abstraction layer, the simulation accuracy of robots in NERD can be fine controlled to facilitate more similar behaviors between simulated and physical robots. This will ease the transfer of evolved controllers. NERD also supports computer clusters to allow evolutions for complex robots that require simulations with higher accuracy.

Currently the NERD kit is used for the evolution of neural behavior control for different kinds of complex robots, e.g. humanoid robots and multi-legged walking machines.

Acknowledgments. This work was partly funded by EU-Project Number ICT 214856 (LEAR Artificial Language Evolution on Autonomous Robots. <http://www.lear.eu>). Thanks go to Chris Reinke, Lucas Theis, Arndt von Twickel and all other contributors of the NERD kit.

References

1. Bender, J., Schmitt, A.: Fast dynamic simulation of multi-body systems using impulses. In: Virtual Reality Interactions and Physical Simulations (VRIPhys), Madrid, Spain, pp. 81–90 (2006)
2. Bullet Physics Engine, <http://bulletphysics.org> (last visited May 2010)
3. ECMAScript Language Specification, ECMA-262 Standard, 5th edn., ECMA International, Rue de Rhone 114, CH-1204 Geneva (December 2009)
4. Floreano, D., Husbands, P., Nolfi, S.: Evolutionary Robotics. In: Siciliano, B., Khatib, O. (eds.) Springer Handbook of Robotics, pp. 1423–1451. Springer, Heidelberg (2008)
5. Harvey, I., Paolo, E., Wood, R., Quinn, M., Tuci, E.: Evolutionary robotics: A new scientific tool for studying cognition. *Artificial Life* 11(1-2), 79–98 (2005)
6. Hülse, M., Wischmann, S., Pasemann, F.: Structure and function of evolved neurocontrollers for autonomous robots. *Connection Science* 16(4), 249–266 (2004)
7. Nolfi, S., Floreano, D.: Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines. Bradford Book (2004)
8. Pasemann, F., Steinmetz, U., Dieckman, U.: Evolving structure and function of neurocontrollers. In: Angeline, P.J., et al. (eds.) Proceedings of the Congress on Evolutionary Computation, Washington, D.C., vol. 3, pp. 1973–1978. IEEE Press, Los Alamitos (1999)
9. QT Application Framework 4.5 <http://qt.nokia.com> (last visited May 2010)
10. Rempis, C., Pasemann, F.: Search space restriction of neuro-evolution through constrained modularization of neural networks. In: Madani, K. (ed.) Proceedings of the 6th International Workshop on Artificial Neural Networks and Intelligent Information Processing (ANNIIP), In conjunction with ICINCO 2010, Madeira, Portugal, pp. 13–22. SciTePress (2010)
11. Smith, R.: ODE - Open Dynamics Engine (2007), <http://www.ode.org>
12. Stanley, K.O., Miikkulainen, R.: Evolving neural network through augmenting topologies. *Evolutionary Computation* 10(2), 99–127 (2002)
13. Sun Grid Engine, <http://gridengine.sunsource.net> (last visited May 2010)
14. Zahedi, K., von Twickel, A., Pasemann, F.: YARS: A physical 3D simulator for evolving controllers for real robots. In: Carpin, S., Noda, I., Pagello, E., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2008. LNCS (LNAI), vol. 5325, pp. 75–86. Springer, Heidelberg (2008)
15. Neurodynamics and Evolutionary Robotics Development Kit, <http://nerd.xbot.org> (last visited August 2010)

Simulation and Evaluation of Mixed-Mode Environments: Towards Higher Quality of Simulations

Vinay Sachidananda, Diego Costantini, Christian Reinl, Dominik Haumann,
Karen Petersen, Parag S. Mogre, and Abdelmajid Khelil*

Technische Universität Darmstadt, Research Training Group “Cooperative,
Adaptive and Responsive Monitoring in Mixed-Mode Environments”,
64289 Darmstadt, Germany

<http://www.gkmm.tu-darmstadt.de>

Abstract. For rescue and surveillance scenarios, the Mixed-Mode Environments (MMEs) for data acquisition, processing, and dissemination have been proposed. Evaluation of the algorithms and protocols developed for such environments before deployment is vital. However, there is a lack of realistic testbeds for MMEs due to reasons such as high costs for their setup and maintenance. Hence, simulation platforms are usually the tool of choice when testing algorithms and protocols for MMEs. However, existing simulators are not able to fully support detailed evaluation of complex scenarios in MMEs. This is usually due to lack of highly accurate models for the simulated entities and environments. This affects the results which are obtained by using such simulators. In this paper, we highlight the need to consider the Quality of Simulations (QoS), in particular aspects such as accuracy, validity, certainty, and acceptability. The focus of this paper is to understand the gap between real-world experiments and simulations for MMEs. The paper presents key QoS concepts and characteristics for MMEs simulations, describing the aspects of contents of simulation, processing of simulation, and simulation outputs. Eventually, a road map for improving existing simulation environments is proposed.

1 Introduction

Mixed Mode Environments (MMEs) describe the wide range of scenarios that are heterogeneous with respect to the physical environment (which can be static and structured or highly dynamic and unstructured), and the involved agents (mobile robots with heterogeneous motion, sensing and communication capabilities, heterogeneous static sensors, and humans-in-the-loop). Possible scenarios in this context are monitoring and surveillance tasks using heterogeneous sensors, but also the coordination of multiple autonomous vehicles in a search and

* This research has been supported by the German Research Foundation (DFG) within the Research Training Group 1362 “Cooperative, adaptive and responsive monitoring in mixed-mode environments”.

rescue scenario. In the considered applications, robots equipped with heterogeneous sensors and networking capabilities play an essential role (see Sec. 2 for more examples of MMEs).

Testing of the algorithms and protocols developed for such MMEs before their real-world deployment is vital. However, due to cost and safety issues, real-world tests are often replaced by simulations (1). Simulators are essential tools that permit thorough testing in the considered multi-sensor-actor systems in MMEs. Evaluation based on simulation is one of the keys to efficient development of dependable algorithms for these systems, ensuring that they perform as expected in the target environment (2).

As already proposed with “MM-ulator” (3), the specific benefits of using simulators in a common platform open the door to systematical studies, evaluations, and development. However, the complexity of the addressed scenarios results in very complex simulators, which are usually a combination of other simulators specific for different research disciplines. With these growing combined simulation and evaluation platforms, the importance of systematically ensuring specific levels of accuracy arises.

The Quality of Simulations (QoS) needs to be evaluated in order to characterize the consistency of the simulation results with respect to the real world. QoS is meant to encompass the degree of quality characteristics, focusing on i) accuracy, ii) validity, iii) certainty, and iv) acceptability. The terminology of QoS is new. However, the inspiration of QoS derives from the concepts of Quality of Information (QoI), which is typically used in areas such as database management, management studies and Wireless Sensor Networks (WSNs) (4-8).

Unfortunately, existing simulators are not always satisfactory in terms of the simulation characteristics i)-iv) for MMEs. Hence, this leads us to bring the concepts of QoS to simulators in robotics, WSNs, autonomous vehicles, and distributed systems in general in MMEs.

Simulators are based on models that abstract the reality. The abstraction often is intentional for simplification. However, a question that arises is: “How confident are the results of simulators for transferring them into the real world?” Often, the goal of a simulation is to test accurately only one part of the system, loosing constraints for the rest of the components and skipping their validation process. This equals to ignoring some simulation parameter; sometimes by intention to avoid unnecessary “input noise”, or due to lack of support in the simulator. Hence, it is unclear whether such approximate models still provide the desired level of QoS accuracy in the real world. To this end, we must know the validity of the model to be acceptable in the real world.

Bringing simulated results close to real environments such as MMEs and satisfying their requirements will be further improved by considering QoS. Some of the real benefits of existing simulators (9-16) are discussed in surveys, e.g., (17-19). It comes down to advantages such as reduced cost, higher level of details and safety. However, there are disadvantages as well, such as results not respecting all real-world requirements due to simplified models (20), or the lack of parameters. Moreover, one of the core objectives is to satisfy the set of user

requirements, and to assess the usefulness of the simulated results for a real environment. Consequently, the concept of QoSIM reflects an extension to the concept of “MM-ulator” (3).

Instead of simple validation of simulation results, QoSIM has a broader meaning with respect to the relation to real-world experiments – namely the aspects i)–iv). Hence, we discuss this set of simulation requirements to show that they are significantly important. QoSIM is rarely taken into account in the design of simulators. Thus, we classify the simulations with respect to: (1) the model view, (2) the application view, and (3) the user view. This helps us to map the simulation characteristics with the provided classification to understand why the aspect of QoSIM is important for simulations.

Besides providing the QoSIM definition, characteristics, and classification, we also suggest the use of QoSIM for simulations based on the following aspects: (1) the simulation content, which takes into account the different models and parameters provided by simulators, (2) the process of simulation, based on the duration of simulations, models used, and (3) the simulation results, which provide the level of abstraction to the developer.

In particular, the scientific contributions are: (a) a systematic analysis of QoSIM for MMEs, (b) providing simulation characteristics, (c) the classification of simulations and their mapping to QoSIM characteristics, and (d) the conceptualization of QoSIM.

The paper is organized as follows. After characterizing specific requirements and features of MMEs simulations in Section 2, and after a definition of QoSIM in Section 3, in Section 4 simulation characteristics are discussed. In Section 5 the classification of simulations is presented and mapped to the simulation characteristics. In Section 6, we conceptualize QoSIM and in Section 7 the road map for future research directions is depicted.

2 Simulation and Evaluation in Mixed-Mode Environments

In MMEs, the application scenarios vary from search and rescue, to exploration of hostile terrain, to planetary exploration, etc. In (3), requirements of inter-disciplinary simulation were already discussed, benefits of using multi-disciplinary knowledge were presented, and road-maps for a common evaluation platform were proposed.

Still there is no implemented solution satisfying the whole spectrum of needs in MMEs-scenarios. These needs contain different time-scales and abstraction levels (cf. Fig. 1), various interfaces, possibly to be combined with real-world experiments (including the known elements of software- and hardware-in-the-loop tests), physical details like dynamic temperature distribution, interaction and manipulation of robots in their environment, etc. The basic corresponding modular architecture consists of components for simulating node properties, and components for simulation and interaction with the physical environment. Thereby, the idea of using a network of sensing platforms, i.e., combining low-end monitoring

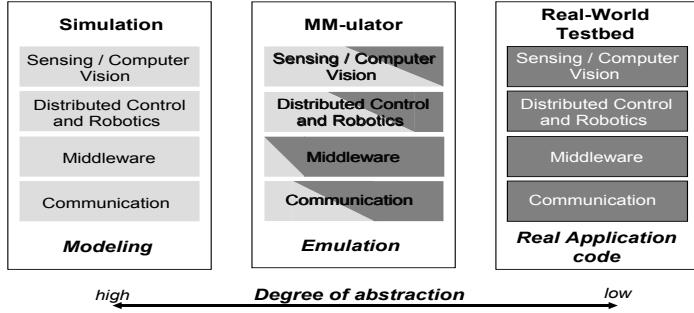


Fig. 1. Simulators from different research disciplines are using a common validation platform “MM-ulator” for emulation

equipment with high-end data gathering functionality, has to be reflected into the architecture of each simulated node and its interfaces.

For illustration, we are mentioning a disaster scenario after an earthquake, where multiple distributed sensors and mobile robots are assisting humans in mapping and monitoring the environment, and in detection of victims. The scenarios ask for communication among nodes (for coordination and sharing data), for applying principles of distributed sensing and feature detection, for dependable middleware in the multi-agent system, and for elaborated locomotion and cooperative control of robots. Evaluating the collaboration of these components requires a detailed simulation on different levels of abstraction. For example, the physical model of the robot can either be a bounding box around the whole robot, with just a point mass, or a more detailed model with bounding boxes around the body parts that are connected with joints, or a detailed physical model based on the CAD data, with exact masses and inertial tensors. Since robots are equipped with sensors, also their readings are simulated on an adjustable accuracy. The simulation of a laser scanner, for example, can either return ground truth distances, or can add noise, or consider different reflectance properties from different materials or effects occurring at fast motions.

State-of-the-art simulators in robotics like [12, 13] provide functionalities to simulate robotic environments. Evaluation in the whole spectrum of MMEs lies beyond these capabilities, and requires information from additional models as implemented, e.g., in WSNs simulators like TOSSIM [9]. Combined simulations environments for sensors and robots with detailed models for wireless communication, enable to study effects in controlling network topology by positioning of robots as well as cooperative data aggregation and feature detection.

As a consequence, the availability of a holistic evaluation platform based on simulations allows for more flexible and multi-disciplinary performance metrics, and problems can be seen from new perspectives. For instance, by migrating wireless network constraints into robot control, new metrics like coordination stability are emerging.

All these aspects give a further motivation in the current phase of building and connecting more and more complex simulation and emulation environments for a more systematical consideration of QoSim concepts, and it draws out that QoSim is one of the key points to apply these complex evaluation platforms for substantiated answers to new difficult scientific questions.

3 Quality of Simulations: Forms and Definitions

We are introducing the concepts of simulations to formalize the definition of QoSim from the basics of QoI. With the set of characteristics associated with QoSim, different levels of abstraction can be obtained, which helps the developer in evaluating cost, safety, duration, and results of the simulation.

Developing robots, particularly cooperative multi-robot systems, strongly relies on simulations to test hardware and integrated software. In particular, many algorithmic tests must be done within a controlled simulated environment to detect failures and misbehaviors without cost and safety concerns, therefore simulations must consider detailed physical 3D properties, like obstacles or gravity.

Sensor nodes in WSNs or robots equipped with sensor nodes also have a set of specific requirements. There are a few simulators, such as TOSSIM (9), where the evaluated code can be directly brought into the real world. However, the simulated results vary in real environments due to perturbations and environmental conditions, leading to a lack of quality. Moreover, using existing simulators for MMEs sometimes does not satisfy the requirements of the applications.

The core objectives of a MME-simulator is to satisfy complex requirements needed by the user, and assess how close the simulated results are to a real environment. Our main focus is on quality aspects of simulations, and since we are motivated by QoI (7)(8)(4), let us brief through some of QoI definitions in WSNs. We choose these definitions of QoI in WSNs because of their relevance of quality aspects in MMEs. QoI definition in regard to monitoring environments (5) is defined as “the difference between the data that the output of the WSNs produces concerning some environment that is being monitored, and the actual events in that environment which one wishes to observe or track”. The multi-dimensional concept (6) of QoI on application view is “the collective effect of the available knowledge regarding sensor derived information that determines the degree of accuracy and confidence by which those aspects of the real world (that are of interest to the user of the information) can be represented by this information”.

In this paper, from the inspiration of QoI we define QoSim in regard to its attributes in MMEs. We therefore propose the following definition:

Definition 1. *Quality of Simulations (QoSim) represents the discrepancy between the simulation results and those expected by the user.*

This definition is meant to encompass the degree of quality characteristics accuracy, validity, certainty, and acceptability.

4 Abstracting the Features of Quality of Simulations

After defining the QoSIM with a set of characteristics, we now detail these characteristics and also brief their importance. In literature, several quality characteristics exist, such as stability, robustness, and consistency. Although we acknowledge their importance, we did not address them in this paper because they go beyond our scope. They depend mostly on the application to simulate rather than the simulator. We assume simulator developers to be more experienced programmers than common users, therefore already considering such important aspects which have been thoroughly addressed in several areas. We considered instead accuracy, validity, certainty, and acceptability (cf. Table I). The main reason for this choice is the relevance in simulations and for the developer.

Accuracy is the characteristic of simulations in which results are similar to real-world values. Therefore, it is most relevant to consider accuracy of models and of simulation output. The developer would accept the results of simulation only if achieving the same level of accuracy of real-world experiments.

Validity is based on the models used for simulation. The validity of the model reflects the consistency to the real world. If the selected model in the simulator is not satisfying the developer's requirements or if a specific model developed by the user is not satisfying the real-world conditions, then the model lacks validity for certain simulations. However, the level of validity strongly depends on the application; if the application is requiring only a small set of parameters, then the simulation can be simple enough to be proved for its validity.

Certainty is the characteristic of simulations on which the developer can place a certain level of belief about the model and simulation itself. Certainty is closely related to accuracy and validity of model and simulation. After simulating an application, and if the results are comparable to real-world experiments, then it fulfills the aspect of certainty. In robotics, during some simulation, the results are good enough to validate the model, but to prove the same model in real world, it could get less belief, or sometimes the real-world results completely deviate from the simulated results. In this case, one can accept the model with lower level of certainty.

Acceptability is the degree of satisfaction in terms of factors such as cost, safety, time and results of simulation. The selected model and the simulations can

Table 1. Summary of main QoSIM characteristics

Characterizing feature	Definition
Accuracy	The accuracy of simulation results compared to the real-world results
Validity	The prediction of the model consistency from simulation to real world
Certainty	The belief and level of abstraction of the developer on the model and simulations
Acceptability	The acceptance of simulations based on cost, safety, time and results

be accepted by the developer based on these factors. The most important are cost and safety, which can be reliably accepted. However, sometimes computational complexity and simplification in the model used make the results insufficient, and they can be barely accepted by the developer.

5 Simulation Classifications

This section gives a QoS-based classification of simulations, not implemented so far to the authors' knowledge. This is relevant to achieve accuracy levels and helps to understand different perspectives of simulations as well as it shows how existing simulators can be improved by adopting the classification from low to high level of abstraction. We classify the simulations into three classes, the model view, the application view, and the user view. Fig. 2 illustrates the mapping of different characteristics to them.

Model view. This view is solely dependent on the combination of selected model and the simulation environment. The model view can be directly mapped with the *validity* characteristic. The model has to satisfy all the user's requirements. This view strongly affects the next two classes. If the chosen model is not satisfactory in providing all the parametrization required by the application, it will be hard to bring the same model into real-world applications. Therefore, the model must be accurate enough to provide results comparable to real-world experiments.

Application view. Sometimes, in MMEs, application scenarios get complex by exploding the amount of requirements that comes up with a simultaneous use of robots and WSNs. In this case, the simulation results depend on the simulator which can simulate both robotics and WSNs models together. Sometimes this creates a hard situation for the developer searching for the right simulator to satisfy the requirements of the application. On the other hand, the model view of the simulations plays an essential role and affects the application view of the simulations. This class is mapped with the *certainty* characteristic. Since the certainty gives the level of abstraction, if the simulator is capable of fulfilling all the required parameters for the environments (consisting of robots and WSNs), then the developer can have a strong belief on his application and simulation results.

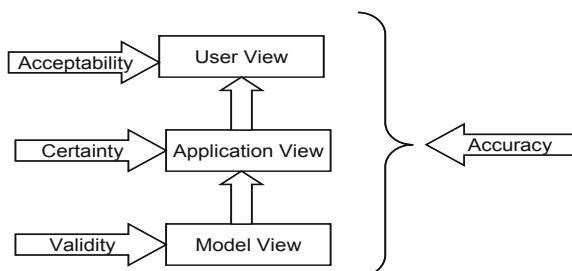


Fig. 2. Simulation Classifications and requirements of corresponding characteristics

User view. This class of simulations is about performing experiments using simulations and how they can be applied in real world. Usually, when the user is working on real-world experiments and has simulated the model in advance, the developer needs to understand how well these results match the accuracy level of real-world experiments. This can be achieved only with a good model and it is specific to the application; hence, the above two classes affect this class. Moreover, this class also depends on the aspects of cost, safety, time, and results. Therefore, it is mapped with the *acceptability* characteristic. This class provides the acceptance rate of the simulation results to work with the same prerequisites in real world.

6 Conceptualizing Quality of Simulations

After presenting the QoSIM characteristics and its mapping with different classes of simulations, we now conceptualize QoSIM.

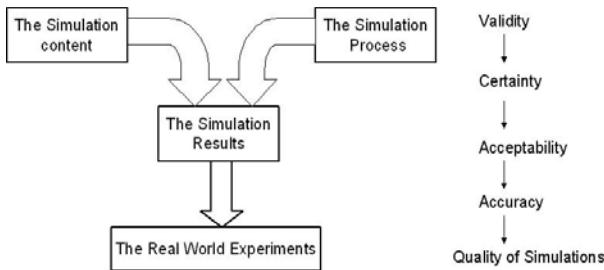


Fig. 3. Conceptualizing Quality of Simulations: related aspects and characteristics

The first and foremost aspect is the *simulation content*. One of the main objectives of simulations is to understand and validate the model in order to bring the same to real world. Here, simulation content represents the available models, parameters and level of details of the simulation environment. Moreover, the simulation content is based solely on the chosen simulator. Generic simulators such as ns-2 [10] and OMNeT++ [11] provide most of the models and parameters, but lack support for WSN platforms. Thus, the simulation content is not satisfying the complex requirements of MMEs. Simulators such as COOJA [21] and MSPSim [22] simulate WSNs but do not provide all required models. Hence, it is essential that simulation content in MMEs is verified with the QoSIM characteristics.

The second aspect we consider is the *simulation process*, which depends mainly on the developer and the usage of the simulator itself. If a simulator meets all the necessary requirements to satisfy the application, then the QoSIM depends on how well the developer uses its characteristics in order to obtain simulated results comparable to real-world experiments. The simulator chosen, the duration of simulation, and how well the algorithm was simulated – including different

models – are the main factors during the entire simulation process. However, in MMEs, with increased complexity of the requirements, the developer must be thoroughly aware of the abstraction level achievable by using QoSim characteristics. If the developer *validates* the model in an efficient way and has a *certain* belief on the simulation process, then the results can be carefully checked for the aspects of *acceptability*.

The third aspect is the *simulation results*, based on the usage of results and their acceptability. The developer can predict the use of simulations results in real cases. As shown in Fig. 3, the simulation results are based on the simulation content and simulation process, which lead to real-world experiments. Once the chosen simulator and its content is satisfying the QoSim aspects in terms of validity, the developer has to perform simulations in such a way that the aspect of certainty is fulfilled. Later, achieving the simulated results, the developer can know the level of acceptability.

7 Discussion and Future Directions

QoSim is highlighted as a vital factor to be considered in simulations for MMEs. Since to the authors' knowledge there is no such simulator which simulates applications with all the characteristics presented, the gap to fill is to develop new simulators or improve the existing ones to achieve reasonable results that can be brought into real-world experiments.

While replacing real-world experiments with simulations, a realistic simulation also has to contain effects like disturbance, noise, chaotic behavior of non-controllable agents. These factors help in bridging the gap between real world and simulation. This research focus helps in building up a new simulation environment and hence leading to an efficient simulator satisfying environmental requirements.

In this paper, QoSim is simplified. Considering the provided classification, one of the improvements that can be applied to existing simulators is adapting the different views mapped with QoSim characteristics. This provides levels of abstraction for adopting the simulations to real-world experiments.

Though some existing simulation environment like ns-2 and OMNeT++ is generic and provides a few models, it is difficult to use them in MMEs including robots, WSNs, and possibly humans-in-the-loop. Hence, a necessary step is to draw attention towards developing a simulator satisfying the MMEs requirements while taking into account the aspects of QoSim.

8 Conclusion

The complexity of requirements in MMEs evolves depending on many factors such as environmental and operational conditions. Using simulators to evaluate approaches for multi-disciplinary research, for the benefit of cost, safety, scalability, repeatability, etc., is well-accepted and often the only possible way to evaluate new methods in early stages of development. However, taking simulated models,

algorithms, and results to the real world degrades QoS characteristics. Sometimes, these characteristics are not even considered. The focus of this paper was to identify the gap and provide a road map to apply QoS concepts in holistic evaluation and simulation platforms. We discussed that considering QoS is necessary in order to achieve comparability to real-world experiments. We also provided sufficient QoS characteristics and conceptualized them. However, this paper does not give a specific solution to the mentioned problem, but details the gap existing between real-world environments and simulations. Thus, for very complex MMEs, we highlight the importance of systematically applying QoS concepts for the goal of studying new fundamental multi-disciplinary research questions in a significantly more reliable way.

References

1. McHaney, R.: Computer simulation: a practical perspective. Academic Press Professional, Inc., San Diego (1991)
2. Eriksson, J.: Detailed simulation of heterogeneous wireless sensor networks. PhD thesis, Uppsala University, Department of Information Technology (May 2009)
3. Kropff, M., Reinal, C., Listmann, K., Petersen, K., Radkhah, K., Shaikh, F.K., Herzog, A., Strobel, A., Jacobi, D., von Stryk, O.: MM-ulator: Towards a common evaluation platform for mixed mode environments. In: Carpin, S., Noda, I., Pagello, E., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2008. LNCS (LNAI), vol. 5325, pp. 41–52. Springer, Heidelberg (2008)
4. Zahedi, S., Srivastava, M.B., Bisdikian, C.: A computational framework for quality of information analysis for detection-oriented sensor networks. In: Military Communications Conference, MILCOM 2008., pp. 1–7. IEEE, Los Alamitos (2008)
5. Gelenbe, E., Hey, L.: Quality of information: An empirical approach. In: 5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems, MASS 2008, pp. 730–735 (September 2008)
6. Zahedi, S., Bisdikian, C.: A framework for QoI-inspired analysis for sensor network deployment planning. In: WICON 2007: Proceedings of the 3rd International Conference on Wireless Internet, pp. 1–8. ICST, Brussels (2007)
7. Thornley, D.J., Young, R.I., Richardson, P.J.: From mission specification to quality of information measures-closing the loop in military sensor networks. In: ACITA 2008 (2008)
8. Bisdikian, C., Damarla, R., Pham, T., Thomas, V.: Quality of information in sensor networks. In: ITA 2007 (2007)
9. Levis, P., Lee, N., Welsh, M., Culler, D.: TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In: Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, pp. 126–137. ACM, New York (2003)
10. The Network Simulator NS-2, <http://www.isi.edu/nsnam/ns/>
11. Varga, A., Hornig, R.: An overview of the OMNeT++ simulation environment. In: Simutools 2008: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, pp. 1–10. ICST, Brussels (2008)
12. Friedmann, M., Petersen, K., von Stryk, O.: Adequate motion simulation and collision detection for soccer playing humanoid robots. Robotics and Autonomous Systems 57, 786–795 (2009)

13. Lewis, M., Wang, J., Hughes, S.: USARSim: Simulation for the Study of Human-Robot Interaction. *Journal of Cognitive Engineering and Decision Making* 2007, 98–120 (2007)
14. Gerkey, B.P., Vaughan, R.T., Howard, A.: The Player/Staging Project: Tools for Multi-Robot and Distributed Sensor Systems. In: *Proceedings of the 11th International Conference on Advanced Robotics*, pp. 317–323 (2003)
15. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2149–2154 (2004)
16. Michel, O.: Cyberbotics Ltd. webots tm: Professional mobile robot simulation. *Int. Journal of Advanced Robotic Systems* 1, 39–42 (2004)
17. Friedmann, M.: Simulation of Autonomous Robot Teams With Adaptable Levels of Abstraction. PhD thesis, Technische Universität Darmstadt (November 30, 2009)
18. Curren, D.: A survey of simulation in sensor networks. Technical report, University of Binghamton
19. Mekni, M., Moulin, B.: A survey on sensor webs simulation tools. In: *Second International Conference on Sensor Technologies and Applications, SENSORCOMM 2008*, pp. 574–579 (25–31, 2008)
20. Mogre, P.S., Hollick, M., d'Heureuse, N., Heckel, H.W., Krop, T., Steinmetz, R.: A Graph-based Simple Mobility Model. In: *4th Workshop zu Mobilen Ad Hoc Netzen, KiVS 2007* (2007)
21. Österlind, F., Dunkels, A., Eriksson, J., Finne, N., Voigt, T.: Demo abstract: Cross-level simulation in cooja. In: *Proceedings of the First IEEE International Workshop on Practical Issues in Building Sensor Network Applications* (2006)
22. Eriksson, J., Österlind, F., Finne, N., Tsiftes, N., Dunkels, A., Voigt, T., Sauter, R., Marrón, P.J.: Cooja/mspsim: interoperability testing for wireless sensor networks. In: *Simutools 2009: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, pp. 1–7. ICST, Brussels (2009)

Evaluating a Physics Engine as an Ingredient for Physical Reasoning

Erik Weitnauer, Robert Haschke, and Helge Ritter

Bielefeld University, Neuroinformatics Group and the Research Institute for Cognition and Robotics, Universitätsstr. 25, 33615 Bielefeld, Germany
`{eweitnau,rhaschke,helge}@techfak.uni-bielefeld.de`

Abstract. Physics engines have been used in robotics research for a long time. Beside their traditional application as a substitute for real world interactions due to their higher speed, safety and flexibility, they have recently also been used for motion planning and high level action planning. We propose to further explore the idea of using a physics engine as means to give a robot a basic physical understanding of its environment. In this paper, as a preliminary step, we study, how accurately the process of pushing flat objects across a table with a robot arm can be predicted in a physics engine. We also present an approach to adapt the engines parameters to enhance the simulation accuracy.

Keywords: physics engines, physical simulation, physical reasoning, robotics.

1 Introduction

One prominent research objective in robotics is to build autonomous service robots that are fit to share our common living space. However, we are still very far from robots that can perform everyday human tasks like tidying a room as smoothly as we do. A major challenge is the integration of the different abstraction layers of information the robot has to deal with. While goals and actions are usually represented at a high level of abstraction, the sensory input and motor control information are represented and processed on a much lower level of abstraction.

To allow reasoning about goals and actions an understanding of the world's dynamics and the effects of actions is necessary. For example, when asked to arrange books on a table it is good to know how they behave – they might fall down if placed too close to the edge of the table. To interact with objects a physical understanding of the world would be very beneficial, as it would provide a very general and powerful way for both interpreting perceptions and planning actions. Such a model of the world's behavior should ideally be learned automatically from low-level sensor inputs, so that the robot can adapt to new situations and no hand-crafted models are needed.

In fact, humans also acquire ‘physical’ concepts and rules, which help them to interpret the world. The very basic concepts are already learned at a quite

early age: Young infants of only three months already have a concept of solid objects. They are surprised when they see solid objects passing through each other or not falling down when they lack support [1]. This kind of ‘naive physical understanding’ of the world certainly plays an important role in structuring their perception and actions.

The artificial intelligence community has long been concerned with finding appropriate computational models for human-like physics knowledge and reasoning. The developed theories and algorithms such as qualitative process theory [2] and QSIM [3] are all based on qualitative representations. Their power lies in their ability to infer all possible behaviors of the modeled systems without the need for complete, quantitative knowledge of the system’s initial state. In improved versions of the original QSIM algorithm, available numerical data can be additionally fed into the qualitative simulation to further restrain the number of possible behaviors [4].

Physics engines provide an alternative way of simulating physical interactions quantitatively and have become popular in the robotics community in the last ten years. Traditionally applied in the design and monitoring of actual physical systems, they have recently also been used in the context of evolutionary optimization and motion planning. Lipson et al. [5] use physical simulations to repeatedly evaluate the fitness of a large number of individuals in an evolutionary process to develop blueprints and motor programs of robots. The motion planning community employs physics engines to simulate the temporal progress of controlled motions of complex robots where kino-dynamic constraints must be taken into account [6].

In contrast to qualitative simulation techniques, physics engines operate at a lower abstraction layer and can be used for robot control more directly. However, they are so far not considered as means to endow a robotic system with the ability to reason about physical world interactions. We propose to take a new perspective on physics engines in this regard.

Reasoning processes guiding an autonomous robot naturally have to cope with questions concerning the stability of manipulated objects such as whether they could tip over or start to roll or how to shift several objects without changing their configuration. Other questions are related to the controllability of objects and the estimation of uncertainty in the prediction of the actions’ effects. Physics engines provide answers to these questions in a general and detailed way. They can therefore complement classical rule-based reasoning systems by acting as a source of information or ‘oracle’ for them. Pasula’s work [7], in which she uses a physics engine to build a simplistic qualitative world model, provides an example along this line.

As a first step towards the long-term objective of employing physics engines in physical reasoning, in this paper we closely analyze the Bullet engine, which is an arbitrary choice among current, freely available engines. We present our study on the accuracy of predicting outcomes of real world interactions in a well-defined scenario: a robot arm pushing flat objects across a table. In the following section, we describe the experimental setup and the data acquisition

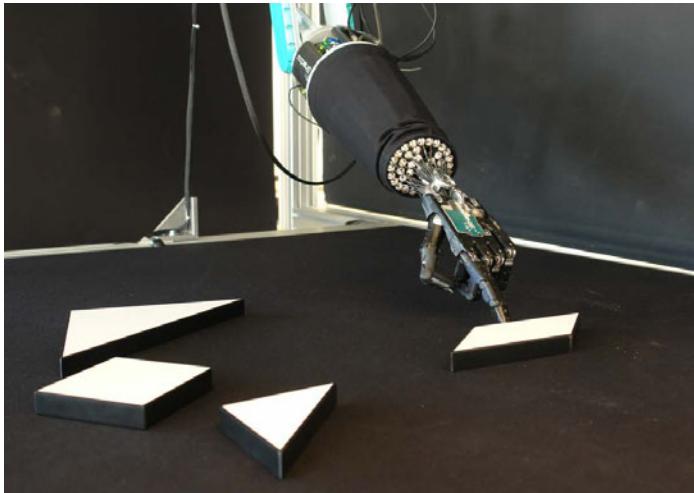


Fig. 1. The robot pushes one of the tangram pieces across the table. The results of this interaction are recorded and used to improve the accuracy of the according physical simulations.

process. Section 3 gives an overview of the Bullet physics engine, its capabilities and restrictions, many of those being typical for physics engines in general. In section 4 we present an optimization method to choose simulation parameters, which improve the resemblance of simulated and real world actions. Finally, we discuss the results in section 5 and give an overview of possible applications of the proposed techniques in section 6.

2 Experimental Scenario

2.1 Setup

We decided to test the simulation accuracy of a current physics engine in a scenario of flat objects being pushed across a table. This scenario was chosen, because it is simple and still several physical properties like friction, mass and shape of the objects have to be taken into account in simulations. Specifically, we used the four wooden tangram pieces shown in Fig. 2. In each pushing trial, one of the tangram pieces was placed on the table at a predefined position and then pushed against its side with a robot hand. The hand was attached to a redundant 7 DoF Mitsubishi PA10 robot arm.

The trials were conducted with the square, parallelogram, small triangle and large triangle placed at varying initial positions x_0 (see Fig. 2). For each trial, the kind of tangram piece, its initial position, its final position and the trajectory of the pushing fingertip were recorded. The path of the hand movement was always identical, although the duration it was actually pushing the pieces varied for their different initial positions (Fig. 3). In total we had 19 different start

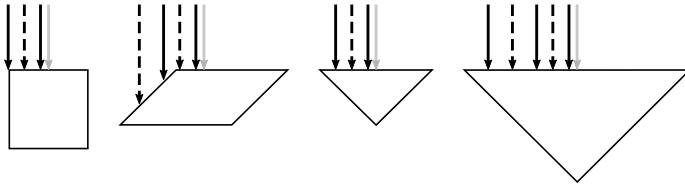


Fig. 2. The arrows in the figure mark the positions where the robot hand started to push the tangram pieces. The tangram pieces were positioned at $x_0 \in \{10 \text{ cm}, 11 \text{ cm}, 13 \text{ cm}, 15 \text{ cm}, 18 \text{ cm}, 21 \text{ cm}\}$, while the hand trajectory followed a straight line of 40 cm length at $x = 10 \text{ cm}$. The pushing trials with the dashed start positions were used as test data, the solid ones as training data in the optimization process outlined in Section 4

conditions across the objects, for each of which we performed five (identical) pushing trials. Repeating the trials was done to find out how much the final poses of a pushed object vary under identical conditions, providing a baseline for the subsequent predictions.

2.2 Data Analysis

The camera system used to recognize the pushed object's shape as well as its initial and final pose provides an accuracy of $\pm 1 \text{ mm}$ for position and $\pm 1^\circ$ for rotation. The fingertip trajectory was obtained from applying the forward kinematics to the joint angle trajectory recorded with 100 Hz. Due to controller inaccuracies, the deviation of the fingertip trajectory from its ideal straight line amounts to $\pm 2 \text{ mm}$.

An interesting aspect of the recorded pushing actions is the variance within the five identical trials for each of the starting conditions. Due to the limited

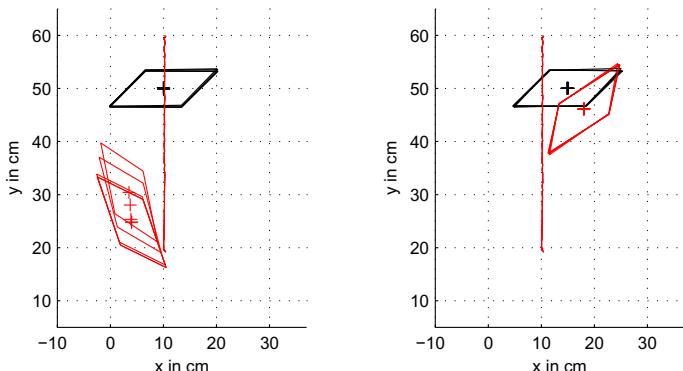


Fig. 3. The two figures show pushing trials recorded for the parallelogram-shaped object, which was placed at a different initial position in the left and in right figure. In each figure the final object positions of five repeated trials are plotted. The vertical red line marks the trajectory of the fingertip. When pushing close to the symmetry axis of the object, the variance of final position becomes bigger.

precision with that the tangram pieces can be placed at the designated initial position and the controller inaccuracies, it is not possible to exactly reproduce a single trial. However, the variance between identical trials strongly depends on the starting condition. It is highest when the pieces are pushed close to their center (see Fig. 3). The standard deviations for all shapes are plotted in the results section, where we compare them to the simulation results.

3 Action Prediction with Physics Engines

3.1 Choosing a Physics Engine

The design of physics engines heavily depends on their intended application and focuses either on high precision (e.g. for weather forecasting, climate calculations or airplane design) or on real-time performance (e.g. for computer games). Since we want to use the physics engine for continuous simulation and planning of robot behavior, real-time capability of the engine is essential.

 compared ten major real-time physics engines with regard to five categories including material properties, constraint stability and stacking behavior. Although there was no physics engine that performed best in all categories they concluded that “of the open source engines the Bullet engine provided the best results overall, outperforming even some of the commercial engines.”

Based on this statement and considering the fact that it is freely available as open source¹, we decided to use the Bullet engine for our simulations.

This engine is capable of simulating rigid bodies as well as soft bodies which can be used to model cloth, ropes or deformable volumes. The engine implements Newton’s laws of motions and uses a simple friction model to simulate Coulomb friction. Bodies are prevented from penetrating each other by automatically introducing constraints, which are solved together with all user defined constraints, e.g. representing joints.

3.2 Stability and General Restrictions

One important issue about the physics simulations is how quickly two simulations diverge, when there are small differences in their initial conditions. There are two reasons for divergence. One is the limited numerical accuracy of floating-point operations, so that tiny numerical errors might accumulate in the simulation and cause different final results. The second reason are bifurcation points in the physical simulation state space. When placing a pyramid on its tip or pushing against one of the tangram pieces on one of their symmetry axes, the eventual direction of motion changes dramatically even for tiny changes in the initial conditions. Obviously, this also applies to real world experiments. However, the divergence of simulations can be reduced by changing parameters of the simulation algorithm, as this influences the way numerical errors accumulate.

¹ The code is open source under the MIT License, which permits anyone to use it for any purpose. The code is hosted at <http://code.google.com/p/bullet/>

The implementations of most physics simulations focus on high performance at the cost of less accuracy. In fact, the physical laws like energy conservation or the constraint that objects should not intersect, are not *exactly* fulfilled. In most cases the resulting errors are quite small, but for certain parameter combinations, the simulated scene might become unstable and even ‘explode’. This behavior will obviously also result in quick divergence of repeated simulations.

To address these issues, we did an iterative linear search over those simulation parameters, that significantly influenced the simulation stability to determine the values leading to the smallest variance of end results in repeated simulations. The obtained values are listed in the “global parameters” part of Table II.

Another important question is how to adequately model real world objects in the physics simulation. As the Bullet engine is supposed to work best with objects not too big or small, a scaling factor applied to the dimensions of each of the real world object did improve the simulation for us. Last but not least, the friction is computed using a simple Coulomb friction model which is applied to each of the contact points between two bodies. In the case of the tangram pieces, the number of contact points is no higher than four and they are usually located at extremal points of the surface, in our case the corner points of the shapes. For an object rotating on the ground, this simplification leads to a strong overestimation of the friction-induced torque, because instead of correctly calculating the torque as

$$\tau_{\text{real}} = \mu F_N \frac{1}{A} \int r dA, \quad (1)$$

it is calculated as

$$\tau_{\text{sim}} = \mu F_N \sum r_i, \quad (2)$$

where μ is the dynamic friction coefficient, F_N is the normal component of the objects weight, A is its ground area and r is the distance of a point in A to the rotation center. Instead of integrating and thus averaging over all distances, only the extremal distances of the corner points r_i are taken into account by the engine. Therefore, in simulations the rotation of objects will slow down too fast. We solved this problem by slightly modifying the tangram objects in the simulations. Underneath each of the objects we attached a thin plate of identical shape, but reduced dimensions, thereby moving the contact points closer towards the rotation center, such that the friction-induced torque becomes smaller. Since the overall shape of the objects is merely affected, their behavior during collisions remains unchanged. We will refer to the shrinking factor as *shape factor* from here on. Every surface has a geometry-specific shape factor that can determined analytically.

4 Parameter Adaption

Our aim is to find a parameter setting for the Bullet engine so that its simulations yield the same results as observed in reality. We use the previously recorded data to define a cost function as the distance between simulation results and real world results. The recorded data consists of 19 sets of trials in which a single tangram

piece was placed on the table and then pushed by the robot arm moving on a straight, well defined trajectory, as outlined in Section 2.

4.1 Simulation Parameters

Altogether we determined 17 parameters that affect the physical simulation. They can be divided into two groups: The first group comprises all parameters that reflect physical properties of objects. The parameters of the first group are friction and restitution of materials, the inertia tensor of rigid bodies, linear and angular damping of moving bodies, linear and angular factors that scale the forces acting on a rigid body per simulation step as well as the shape factor we introduced earlier. The second group contains global simulation parameters, most of them fine-tuning the Bullet engine's simulation algorithm: Gravity, the collision margin spanned around bodies, the simulation step size, the world scaling factor, the number of allowed solver iterations and the error reduction parameter (ERP).

By simulating pushing actions while varying these parameters one by one, we identified nine parameters that have significant influence on the simulation: angular factor and angular damping, friction, shape factor, gravity, world scaling factor and collision margin. All of these parameters are candidates for optimization. The latter three of them however, strongly influence the stability of the simulation, so they were fixed to their default values. The angular factor and angular damping parameters change the ratio between rotation and translation of the moved tangram pieces. They do not directly correspond to physical phenomena and the rotation-translation ratio can also be influenced by the shape factor, therefore we can exclude them from the training candidates, too. This leaves us with the three parameters to be adapted, listed in order of decreasing influence:

- shape factor of each tangram piece
- tangram/ground friction
- tangram/finger friction

Table 1. Parameter Settings for Optimization

Parameter Name	Value	Parameter Name	Value
<i>object parameters (adapt)</i>		restitution ground	0.2
shape factor square	0.541	restitution finger	0
shape factor parall.	0.552	correct inertia tensor	on
shape factor tri. (s, l)	0.461	<i>global parameters (fixed)</i>	
friction tangram/ground	0.3	collision margin	0.04
friction tangram/finger	0.3	simulation stepsize	1/60
<i>object parameters (fixed)</i>		erp	0.2
angular, linear factor	1	world scaling factor	3
angular, linear damping	0	gravity	-9.81
restitution tangram	0.2	solver iterations	10

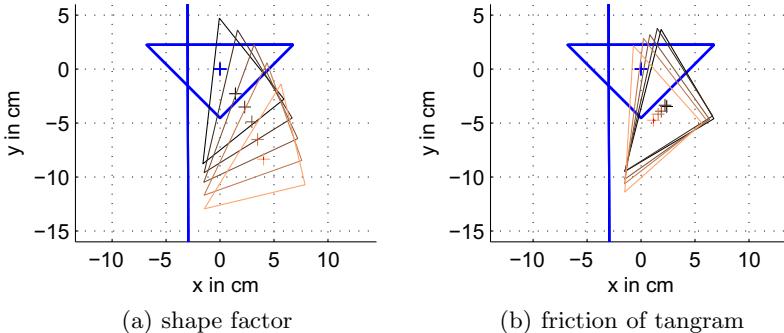


Fig. 4. In both plots the final positions of the pushed triangle as a result of varying one simulation parameter are shown. In the left, the shape factor was varied between 0.3 and 1. In the right, the tangram/ground friction was varied between 0.05 and 1. With increasing values of the parameters, the final positions are plotted in a brighter color.

To show their effect on pushing simulations, in Fig. 4 typical simulation results for two of them are plotted. Before the actual optimization, all parameters were set to the values listed in Table I. For the parameters to be optimized, these values were just a starting point.

4.2 Cost Function

For optimization we need a cost function, that maps each parameter setting onto a quality of the simulation, describing how well the physical interaction in the training data is reproduced. To judge the quality of a parameter setting π , a simulation with the Bullet engine is conducted for each of the trials using the corresponding tangram piece, the given initial position and the recorded trajectory of the fingertip. Afterwards, the final position P_{sim} of the tangram piece in the simulation is compared to the real final position P_{real} in the training data. The polygon position P is described by a tuple of corner coordinates (c_0, \dots, c_n) . The distance between the two final positions is measured as the mean distance of these corner points:

$$E(P, P') = \frac{1}{n} \sum_{j=1}^n \|c_j - c'_j\|.$$

The error on the whole training set with N trials is defined as the mean distance between simulated and real final position:

$$E_D(\pi) = \frac{1}{N} \sum_{i=1}^N E(P_{\text{sim}}(\pi), P_{\text{real}})$$

4.3 Training and Testing Data Set

We divided the recorded data set into a training set D_{train} and a test set D_{test} as follows:

$x_0 =$	10 cm	11 cm	13 cm	15 cm	18 cm	21 cm
D_{train}		×		×		×
D_{test}			×		×	

We do not use the trials where the objects were pushed at their axis of symmetry ($x_0 = 10 \text{ cm}$), since for them the final positions in the real world vary too strongly. Although the number of trials in the training set is quite small (5×2 for square and small triangle), we still expect that the found parameter values will generalize well. This is due to two reasons:

1. We strongly limited the number of parameters that are trained and they relate directly to physical phenomena.
2. The mapping between parameters and final positions of the tangram pieces is complex and computed in a huge number $N \gg 1$ of consecutive steps. So the simulation of one trial can also be interpreted as N consecutive one-step mappings from one world state to the next. To come up with the correct simulation result, all of these mappings have to be rather accurate as well. This reduces the danger of over-fitting.

4.4 Optimization Algorithms

We combined the simplex algorithm [9] and differential evolution [10] to search for optimal parameter values. While the simplex algorithm only performs a single simulation per step, the differential evolution algorithm does one simulation for individual in the current population. With a population of 30 individuals, the differential evolution took about 50 times longer to converge than the simplex algorithm. However, it is also less prone to get stuck in local optima than the simplex algorithm. To combine the strength of both algorithms, we ran 100 iterations with the differential evolution first and then took the resulting parameter vectors as initial conditions for further search using the simplex algorithm.

5 Results

We have chosen two different settings for the training. In the first setting, the shape factor and friction parameters were optimized separately for each of the tangram shapes. In the second setting, only the shape factor was optimized separately for each shape, while all shapes had to share the same friction values which is more realistic. The obtained parameter values and corresponding error values are summarized in Table 2.

The optimized shape factors for the combined training of all shapes are (0.6279, 0.7131, 0.46, 0.5597) for the square, the parallelogram, the small triangle and the large triangle, respectively. When optimizing the three parameters

Table 2. Optimization Results

Shape	Friction T./Ground	Friction T./Finger	Shape Factor	$E_{D_{\text{train}}}$	$E_{D_{\text{test}}}$
Square	0.1246	0.7372	0.5274	0.62 cm	0.97 cm
Parall.	0.1055	0.4403	0.7178	0.59 cm	1.07 cm
Tri. (S)	0.1320	0.7346	0.4615	0.37 cm	0.77 cm
Tri. (L)	0.5951	0.7143	0.5658	0.49 cm	0.69 cm
All	0.3065	0.7495	see bel.	1.25 cm	1.42 cm

individually for all shapes, the predicted final corner positions are only about 0.5 cm off the real positions in the training data. For testing data, the errors are about double that size, which shows that, although experiencing some over-fitting, the optimized parameters still generalize well over new pushing actions. The over-fitting is most likely due to the very small training set. In the chosen distribution of the data into training and testing set, all test trials are situated ‘between’ the training trials, and the optimized parameter vector successfully interpolates between them. In order to explore the ability to extrapolate we additionally trained the parameters for the large triangle with a different data distribution. We used the $x_0 = 18$ cm trial for training and the $x_0 = 21$ cm trial for testing, so extrapolation is necessary to achieve low test errors. In fact, the resulting errors are as low as the ones in the interpolation setting.

In the second optimization setting where the recorded training data of all four tangram pieces is used, the difference between training and testing error becomes much smaller. Except for the tangram/ground friction in second setting, the optimal parameter values did not resemble the empirically determined ones, that are shown in Table II. The reason for this is most likely the imperfect modeling of the real interaction in the physics world, which mainly stems from the simplifications of the physical laws used in the simulations.

We further compared the prediction errors of the trained parameters π_{opt} with those of the default parameter set π_0 on the complete data set D . The errors are listed in Table 3, together with an upper limit E_{base} which is the average distance the tangram corners moved in the real world and would be obtained when simply using their initial positions as predictions. As a measure of how

Table 3. Comparison of Results

Tangram Shape	E_{base}	$E_D(\pi_0)$	$E_D(\pi_{\text{opt}})$	σ_{real}
Square	12.55 cm	3.28 cm	0.74 cm	0.46 cm
Parallelogram	8.39 cm	1.69 cm	0.82 cm	0.23 cm
Triangle (Small)	8.5 cm	0.99 cm	0.51 cm	0.26 cm
Triangle (Large)	17.36 cm	2.73 cm	0.57 cm	0.31 cm
All shapes	12.35 cm	2.23 cm	1.32 cm	0.33 cm

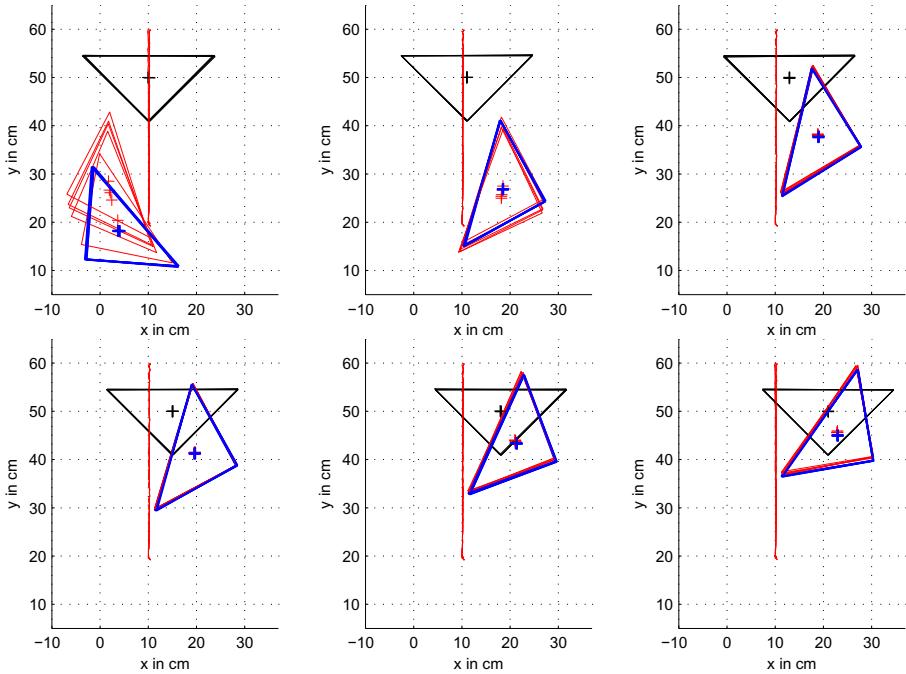


Fig. 5. The six figures show the recorded pushing trials of the large triangle-shaped tangram piece placed at different initial positions $x_0 \in \{10 \text{ cm}, 11 \text{ cm}, 13 \text{ cm}, 15 \text{ cm}, 18 \text{ cm}, 21 \text{ cm}\}$. The initial positions are printed in black, while the final positions and the trajectories of the finger are in red. The bold blue triangles are the final positions predicted by the Bullet engine. In each figure, five trials with identical start conditions are plotted.

reproducible the trials were in the real world, which is a lower limit for the prediction error, we list the standard deviation σ_{real} of tangram final positions in trials with identical start conditions, averaged over all start conditions.

Fig. 5 shows the exact final positions predicted by the Bullet engine together with the actual final positions reached using the robot arm for the large triangular object. For the other shapes, the simulations provide equally good predictions of the places to which the tangram pieces were pushed.

6 Conclusions and Future Work

Our general objective is to explore the use of physics engines in robot's physical reasoning systems. We showed that accurate prediction of real world interactions is possible with a current physics engine for the case of flat objects that are pushed across a table. By adapting the simulation parameters, the predictions errors were reduced to less than half of the errors obtained with the default parameters. Apparently, the engine does a better job in predicting the objects'

motions than humans are capable of, which would have to be evaluated in an additional study.

One interesting next step is to apply the physics engine to more complex scenarios and to adopt its parameters continuously. Enabling physics engines to handle incomplete data about a scene or making more of their inner workings available to learning processes also seems promising. We believe that a successful approach to physical reasoning in robotics will eventually have to combine the strengths of both qualitative simulations systems and quantitative simulation systems.

References

1. Baillargeon, R.: How do infants learn about the physical world? *Current Directions in Psychological Science* 3, 133–140 (1994)
2. Forbus, K.D.: Qualitative process theory. *Artificial intelligence* 24, 85–168 (1984)
3. Kuipers, B.: Qualitative simulation. *Artificial intelligence* 29, 289–338 (1986)
4. Berleant, D., Kuipers, B.J.: Qualitative and quantitative simulation: bridging the gap. *Artificial Intelligence* 95, 215–255 (1997)
5. Lipson, H., Pollack, J.: Evolving physical creatures. In: *Artificial Life VII: Proceedings of the Seventh International Conference on Artificial life*, pp. 282–287. MIT Press, Cambridge (2000)
6. Sucan, I., Kavraki, L.: Kinodynamic Motion Planning by Interior-Exterior Cell Exploration. In: *Intl. Workshop on the Algorithmic Foundations of Robotics*, Guanajuato, Mexico (2008)
7. Pasula, H., Zettlemoyer, L., Kaelbling, L.: Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research* 29, 309–352 (2007)
8. Boeing, A., Braeunl, T.: Evaluation of real-time physics simulation systems. In: *GRAPHITE*, Perth, Western Australia (2007)
9. Nelder, J., Mead, R.: A simplex method for function minimization. *The Computer Journal* 7, 308 (1965)
10. Storn, R., Price, K.: Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces. In: *International Computer Science Institute – publications* (1995)

Simulating Vehicle Kinematics with SimVis3D and Newton

Jens Wettach, Daniel Schmidt, and Karsten Berns

Robotics Research Lab, Gottlieb-Daimler Straße, 67663 Kaiserslautern, Germany

{wettach,d_smith,berns}@cs.uni-kl.de

<http://rrlab.cs.uni-kl.de>

Abstract. This paper discusses the simulation of vehicle kinematics with SIMVIS3D and the NEWTON Game Dynamics Engine. As running example a Pioneer¹ like robot is used. First its differential drive is simulated *manually*, without physical effects. Then the dynamics engine is applied to simulate this drive mechanism based on rolling friction between wheels and ground. Comparing the effort of application code for both approaches shows the benefit of delegating the calculation of rigid body motions. This fact is further stressed by simulating a trailer as a passive vehicle without any code extensions. Finally, a standard vehicle kinematic system consisting of Ackermann steering and rear wheel drive is realized². The paper concludes with an application of this model for simulating the drive system of a bucket excavator as real world scenario.

Keywords: 3d robot simulation, simulated sensors and actuators.

1 Introduction

Simulating a robot system in a realistic scenario is an inevitable tool during the development process of complex control behaviors. It facilitates repeatable validation tests under well defined and controllable conditions concerning environmental properties, e.g. layout of the working space or sensor noise. In order to achieve a good match between simulation scenario and real world the tool has to provide the means of 3D scenery representation and physical effects. This is necessary in a variety of applications, for instance for applying wheel slippage to an outdoor robot running in rough terrain, for analyzing the biomechanics of a walking biped and for manipulating goods with an autonomous forklift.

In [1] the flexible 3D simulation framework SIMVIS3D that is based on Open-Inventor(OIV)³ has been presented. It provides means for simulating common sensors as cameras, laser scanners, infrared, ultrasonic, and PMD sensors. A simulation scene can contain arbitrary 3D models according to the well established OpenInventor and VRML standard, e.g. created with CAD programs as

¹ <http://mobilerobots.com/ResearchRobots/ResearchRobots/PioneerP3DX.aspx>

² The complete C++ code of the Pioneer example realized with an MCA2 integration of SIMVis3D can be downloaded at <http://rrlib.cs.uni-kl.de/downloads>

³ Using Coin3D (<http://www.coin3d.org>) as reference implementation.

*ProEngineer*⁴ or 3D modeling software as *Blender*⁵. It is defined via an XML scene description file that is parsed and translated into a scenegraph during simulation start-up. At runtime the user can affect the scene, e.g. the position of a robot vehicle, at code level by changing numeric values in a data array that parametrize the scenegraph, e.g. the (x,y,z) position and (roll, pitch, yaw) angle of the robot pose. This is achieved by wrapping a selection of OIV nodes in the framework code.

Consequently, there is no restriction on the 3D models that can be introduced in the simulation scene and the framework capabilities are easily extensible by making other OIV nodes accessible through user code. Furthermore, external implementations of simulation capabilities can be integrated with reasonable effort. For instance, [9] discusses the simulation of acoustic properties and walking humans in a human-robot interaction scenario. The first extension is based on [2], the second one on the H-ANIM⁶ standard. And [6] introduces the integration of NEWTON⁷ as physics engine for the simulation of a dynamically walking biped.

This paper starts with a discussion of existing 3D simulation frameworks with optional support for system dynamics, shows their limitations, and emphasizes the need for a flexible and extensible framework (section 2). Then it demonstrates the benefits of using a physics engine for simulating vehicle kinematics regarding straight forward implementation and realistic motion effects. As running example a Pioneer like robot with a 2D scanner is used. First its differential drive system is modeled and simulated manually, by transforming given motion commands into changes of wheel encoder values and corresponding updates of the robot pose in the SIMVIS3D framework (sec. 3). Then the drive mechanism is modeled using the NEWTON dynamics engine for calculating the rolling motion of the driving wheels on the ground under the presence of friction (sec. 4). The power of the physics engine is further demonstrated by introducing a trailer as passive vehicle: its motion is completely computed by NEWTON, without the need for additional user code (sec. 5). Finally, a standard drive concept, consisting of Ackermann steering and rear wheel drive, is realized on a modified Pioneer robot to show the full power of the combined SIMVIS3D and NEWTON simulation framework (sec. 6). This concept is then applied to an autonomous bucket excavator to demonstrate how easy real world vehicles can be simulated with reasonable effort and realism (sec. 7). Conclusion and hints for future enhancements are given in section 8.

2 State of the Art

This section gives a brief overview of state of the art simulation frameworks, most of them with optional support for system dynamics, but all with limited

⁴ <http://www.ptc.com/products/proengineer>

⁵ <http://www.blender.org>

⁶ <http://www.h-anim.org>

⁷ <http://www.newtodynamcis.com>

flexibility and/or extensibility. GAZEBO [3] is a 3D simulator for multiple robots that contains several models of real machines as PioneerP2DX and Segway RMP as well as a variety of sensors like sonar, laser scanners, cameras, GPS, and inertial systems. It provides realistic noisy sensor data and a physically plausible simulation of rigid body dynamics, e.g. for manipulation tasks. Robots and sensors are defined as plugins and the scene is described in XML.

SIMROBOT [5] also uses an XML based scene description with predefined generic bodies, sensors (laser scanner, bumper, camera), and actuators. Dynamics are simulated via ODE⁸. EYESIM [4] is a 3D simulation tool for EYEBOTS, an experimental platform for walking, flying, and vehicle robots, providing sensors as bumpers, cameras, and PSDs, but no support for dynamics.

UASRSIM [10] is a simulation tool used in the *RoboCup rescue competition*⁹. It is based on the *Unreal Tournament* game engine, so 3D scenes can be modeled via the *Unreal Editor* and dynamics are calculated by the *Karma*¹⁰ engine. Supported sensors are sonar, laser scanners and FLIR (forward looking infrared).

WEBOTS [7] is a commercial tool with support for real robots as *Aibo*, *Khepera*, and *Koala* and for light, touch, force, and distance sensors, cameras, and GPS. Dynamics are calculated by ODE. VORTEX¹¹ is another commercial framework for 3D simulations that can be integrated with different scenegraph systems, e.g. OSG¹². It provides sensors as sonar, laser scanners, cameras, depth and force sensors and contains a physics engine for realistic collisions, contact properties and fluid interaction. Furthermore, a particle simulation is available for simulating fluid flow, dust and smoke.

Although most of these frameworks facilitate a realistic 3D simulation of arbitrary vehicles with standard sensors and support for system dynamics there is still a need for a flexible and extensible simulation framework as SIMVIS3D. Commercial tools as VORTEX – although being powerful and rich of features – are not suitable for research purposes for reasons of cost and closed source. Open source frameworks as GAZEBO explicitly support user extensions regarding new robot models and sensors, but do not allow to exchange the physics engine or to integrate simulation tools for unforeseen aspects as acoustics or particles. Furthermore, the support for spreading several simulation processes via a network communication layer on different host computers provides a cheap and straight forward speed up of complex simulation scenarios (cf. [1] for details).

3 Simulating a Differential Drive Manually

In this section the differential drive of a Pioneer like robot, shown in fig. [1], is simulated using SIMVIS3D without support for dynamics. The needed effort is demonstrated by the corresponding XML scene description file (see fig. [2]).

⁸ Open Dynamics Engine, <http://www.ode.org>

⁹ <http://www.robocuprescue.org>

¹⁰ <http://wiki.beyondunreal.com/Legacy:Karma>

¹¹ http://www.vxsim.com/en/software/vortex_core/index.php

¹² OpenSceneGraph, <http://www.openscenegraph.org/projects/osg>

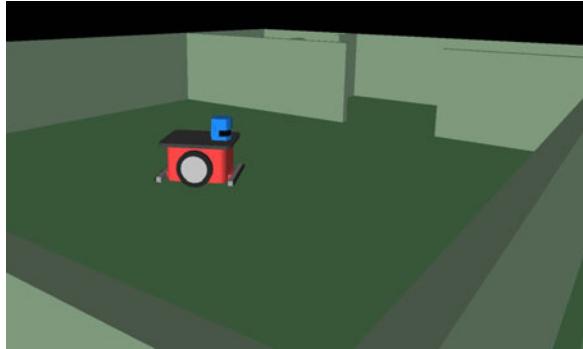


Fig. 1. Pioneer like robot equipped with differential drive and laser scanner in a simple test environment

```

<part file="floor.iv" name="WORLD" anchor="ROOT" offset="0 0 0 0 0 0"/>
<part file="robot.iv" name="ROBOT" anchor="WORLD" offset="0 0 0 0 0 0"/>
<part file="wheel.iv" name="LEFT_WHEEL" anchor="ROBOT"
    offset="0 0.23 0.15 0 0 0" />
<part file="wheel.iv" name="RIGHT_WHEEL" anchor="ROBOT"
    offset="0 -0.23 0.15 0 0 0" />
<part file="scanner.iv" name="SCANNER" anchor="ROBOT"
    offset="0.2 0 0.35 0 0 180" />
<element name="pose" type="3d Pose Tzyx" position="0 0 0" orientation="0 0 0
    anchor="ROBOT"/>
<distance_sensor name="scanner" max_distance="20" scan_angle_range="180"
    angular_resolution="0.5" sensor_offset="0.2 0 0.41 0 0 0" anchor="ROBOT"/>
```

Fig. 2. XML description for the scene shown in figure [1](#)

that has to be provided by the user, as well as by describing the interfaces and exchange of data between functional units of the simulation and control program that have to be implemented.

The scene consists of five OIV parts: the floor as working space, robot body, left and right wheel, and laser scanner. Wheels and scanner are attached with a suitable pose offset $p = (x, y, z, roll, pitch, yaw)$ to the robot (values in m and degree). The core element for influencing the robot pose at runtime is the type **3d Pose Tzyx** element that introduces an *SoMatrixTransform* OIV node between floor and robot and exports the six adjustable values of a 3D pose to the common SIMVIS3D data array. The suffix **Tzyx** specifies the order of applied transformations, i. e. first rotation around x-, y- and z-axis, then translation along these axes. Finally, the distance sensor tag tells the framework to calculate distance data for a simulated laser scanner with maximum range of 20m, scan range of 180°, and angular resolution of 0.5°. It is positioned wrt. the center of the robot body in order to set the sensor center at the correct position within the scanner. Alternatively, the distance sensor could be placed relative to the scanner part itself. For more information about general setup of a scene description cf. [11](#).

The benefit of this SIMVIS3D only simulation scene is to provide continuously updated distance data of the laser scanner, affected by an adjustable amount of

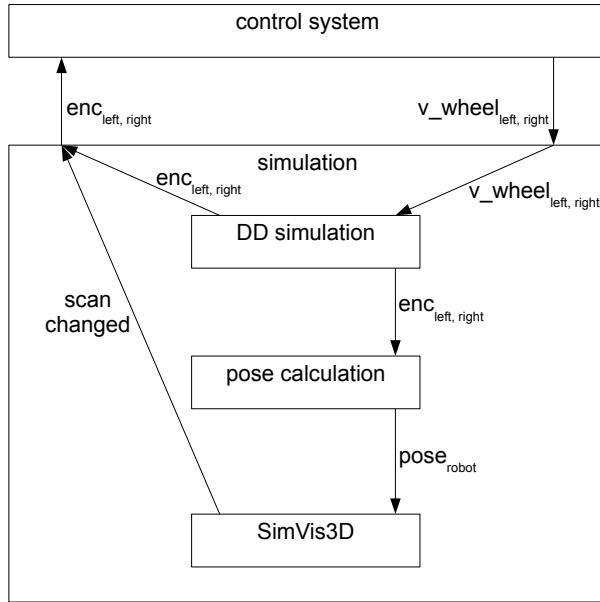


Fig. 3. Functional units and data flow for the simulation of the differential drive

Gaussian noise, while the robot is moving through the scene. This can be used for instance to test an obstacle avoidance system offline without the danger of damaging a real robot. However, the robot motion has to be calculated manually as the simulation requests the current robot pose as input.

Figure 3 shows the corresponding data flow between control program and simulation. The simulation gets desired velocities of the left and right driving wheel in meter per second as input. The *Differential Drive simulation* unit transforms this input into continuous changes of corresponding incremental wheel encoders, e.g. 16 bit counter values. This way the $\Delta\text{enc} = \text{enc}(t_{i+1}) - \text{enc}(t_i)$ reflects the traveled angular distance of the wheel between time t_i and t_{i+1} . This information is transformed by the *pose calculation* unit into the current 3D pose of the robot by the following steps: $\Delta\text{enc}_{\text{left, right}} \rightarrow v\text{-wheel}_{\text{left, right}} \rightarrow v\text{-robot}_{\text{trans, rot}} \rightarrow \text{pose}_{\text{robot}}$. Hence, a complete odometry calculation chain has to be performed to feed the consecutive *SimVis3D* unit with the current robot pose. After each update of the robot pose in the simulation scene, a new set of scanner data is passed in a blackboard (chunk of memory) to the control program. This event is signaled by the *scan changed* counter.

The wheel encoder values are also passed as simulation output to the control program which has to perform the same odometry calculation to *estimate* the robot pose. Thus, the data interface between control and simulation is the same as between control and real robot. In this way the simulation and robot can be

easily exchanged. Naturally, wheel slippage has also to be simulated manually by the *DD simulation* unit, e.g. by falsifying the encoder values.

4 Using a Physics Engine for the Differential Drive

In the following the same simulation scenario as in section 3 is realized, but now using NEWTON as physics engine for simulating the robot dynamics. The core idea is to connect the wheels by a hinge joint to the robot body and apply a certain motor torque on these joints. Then the robot motion is the result of the rolling motion of the wheels on the ground regarding a certain kind of friction between materials of wheels and ground which is computed by the physics engine automatically.

```
<same part and distance sensor section as in previous example/>
<element name="world" type="Physics Static" collision_geom="floor.col"
  anchor="WORLD" />
<element name="robot" type="Matrix Physics" mass="10.0" velocity="0 0 0" omega
  ="0 0 0" material="robot1" collision_geom="robot.col" anchor="ROBOT"/>
<element name="left_wheel" type="Matrix Physics" mass="1.0" velocity="0 0 0" omega
  ="0 0 0" material="rubber" collision_geom="wheel.col" anchor="LEFT_WHEEL"/>
<element name="right_wheel" type="Matrix Physics" mass="1.0" velocity="0 0 0" omega
  ="0 0 0" material="rubber" collision_geom="wheel.col" anchor="RIGHT_WHEEL"/>
<element name="scanner" type="Matrix Physics" mass="2.0" velocity="0 0 0" omega
  ="0 0 0" material="robot1" collision_geom="scanner.col" anchor="SCANNER"/>

<element name="scanner_fixed" type="Physics Joint" joint_type="fixed" parent=
  "robot" child="scanner" pivot="0.0 0.0 0.0" direction="1 0 0" anchor="ROBOT"/>
<element name="left_motor" type="Physics Joint" joint_type="wheel" parent="robot"
  child="left_wheel" pivot="0.0 0.0 0.0" direction="0 1 0" motor="100 0"
  anchor="LEFT_WHEEL"/>
<element name="right_motor" type="Physics Joint" joint_type="wheel" parent="robot"
  child="right_wheel" pivot="0.0 0.0 0.0" direction="0 1 0" motor="100 0"
  anchor="RIGHT_WHEEL"/>
```

Fig. 4. XML description for the dynamics simulation of the scene shown in figure 1

Figure 4 shows the corresponding scene description. The part and distance sensor tags are the same as in fig. 2. Now for each visual OIV model a corresponding collision model with a certain mass has to be provided to the physics engine to perform collision and contact calculations. As the collision primitives available in NEWTON (sphere, cube, cylinder) are also available in OIV, a converter tool has been implemented to generate these models. The materials are defined in a separate file where for each combination of materials the softness, elasticity, as well as static and kinetic friction have to be defined, e.g.

```
<interaction mat1="default" mat2="rubber" softness="0.9" elasticity="0.2"
  static_friction="0.9" kinetic_friction="0.8" callback="generic"/>
```

for the interaction between environment (**default**) and wheel (**rubber**). The environment or world is defined as static object because it cannot be moved within the physics world. Its collision geometry is defined as triangular mesh that can also be automatically extracted from the OIV model.

Of particular interest are the joints between the introduced physical objects. The scanner (**child**) is mounted by a **fixed** joint on the robot (**parent**) as it should not change its position relative to the vehicle. The wheels are mounted by special **wheel** joints on the robot. The **direction** vector defines the y-axis as the rotation axis of the joint since the x-axis is oriented according to the motion direction of the robot. The **motor** value defines the maximum motor torque that can be applied on this joint. As the masses of the collision objects and the torque values are given in artificial simulation units, they have to be adapted in relation to each other until the resulting dynamics are reasonable. Of course, this depends heavily on the used physics engine.

As can be seen from the description file, there is no explicit joint between the robot and the environment: due to gravity and masses the robot automatically hits the ground with its wheels and the friction between wheels and ground translates the applied motor torque into a robot motion. In this example a certain trick has been used to simplify the scenario: as the scanner is mounted at the front end of the robot, but there are only two wheels at its middle axis, the robot would always topple around this axis while moving. To avoid this effect the (invisible) collision geometry of the robot body goes down to the ground while the friction between robot and ground is minimized. This way the robot body invisibly slides on the ground. As an alternative solution, a small castor wheel could have been attached at the rear end of the robot.

The scene description is parsed by the simulation framework and all tags that are not relevant to SIMVIS3D are passed to the physics framework. This one is organized using generic classes for collision models and joints and a specialization based on NEWTON to perform the calculations of the dynamics. This way the physics engine may be exchanged straight forward via subclassing. A detailed presentation of this approach goes beyond the scope of this paper.

Figure 5 shows the resulting simulation unit. The interface to the control program is the same as in the previous example. Now the differential drive is computed by the *physics engine* so there is only a *data conversion* module needed to arbitrate the data flow between simulation interface and the engine. In the top-down direction the translational velocities of the wheels $v_{\text{wheel left, right}}$ have to be converted into rotational velocities $\omega_{\text{wheel left, right}}$ as this is the input of the defined **wheel** joints. In the bottom-up direction the physics engine provides the angular position of the wheels at the joint as $\alpha_{\text{wheel left, right}}$ that has to be converted to a corresponding counter value of the simulated encoders $enc_{\text{left, right}}$.

The physics engine calculates the position of all movable objects in matrix form and directly passes this information to the data array of the SIMVIS3D framework. Thus the engine only tells the SIMVIS3D module to use these values to perform a scene update. Finally, the distance data of the laser scanner is calculated and propagated as in the previous example. The main benefit of using the physics engine is the fact that there is no need for calculating the differential drive simulation manually. Furthermore, the effects of wheel slippage and vehicle

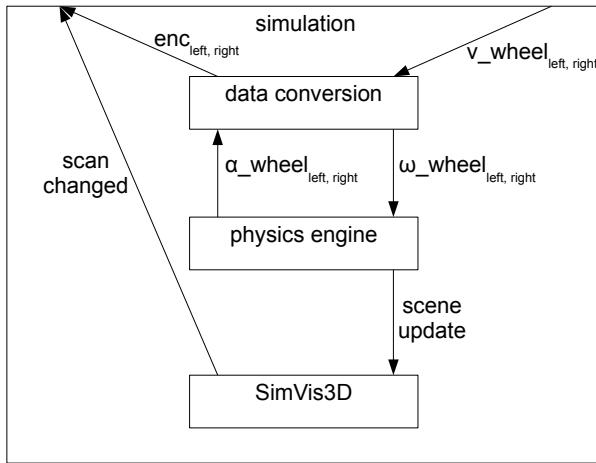


Fig. 5. Functional units and data flow for the simulation of the differential drive using the physics engine

offsets due to collisions are automatically computed by NEWTON which renders the simulation result even more realistic.

5 A Trailer as a Passive Vehicle

This section introduces the simulation of a trailer as completely passive vehicle with the physics engine to further stress its benefits. That means there is no need for additional user code for this extension. Figure 6 shows the necessary additions to the scene description. As the definitions for the left and right wheel are symmetric, the latter ones are omitted.

The part tags define the visual models for coupling and axle (cylinders), the trailer body and the wheels. For body and wheels the same models as for the robot are used. The first section of element tags of type **Matrix Physics** defines the corresponding collision models and masses. Then the coupling is mounted with a **fixed** joint at the rear end of the robot whereas the coupling *axle* is mounted fixed at the front end of the trailer. These two objects are connected by an unactuated **ball** joint so that – apart from the coupling restriction – the trailer can be positioned free in space with respect to the robot. I. e. it will always follow the robot but has not to move on the same ground plane, for instance. Finally, the wheels are mounted via a **free** hinge joint (no motor) at the trailer.

These definitions result in a passive vehicle that cannot be moved by applying any forces on the wheel joints (see fig. 7). The only way of actuation comes from the outside, e. g. by pushing the trailer via the coupling. Such a stimulus will result in a rolling motion of the free wheels on the ground and thus in a motion

```

<part file="coupling.iv" name="COUPLING" anchor="ROBOT"
  offset="-0.35 0.0 0.2 0 90 0"/>
<part file="axle.iv" name="COUPLING_AXLE" anchor="ROBOT"
  offset="-0.69 0.0 0.2 0 0 90"/>
<part file="robot.iv" name="TRAILER" anchor="ROBOT" offset="-1.265 0 0 0 0 0"/>
<part file="wheel.iv" name="TR_LEFT_WHEEL" anchor="ROBOT"
  offset="-1.265 0.23 0.05 0 0 0"/>

<element name="coupling" type="Matrix Physics" mass="0.2" velocity="0 0 0" omega
= "0 0 0" material="robot1" collision_geom="coupling.col" anchor="COUPLING"/>
<element name="coupling_axle" type="Matrix Physics" mass="1.0" velocity="0 0 0"
omega="0 0 0" material="robot1" collision_geom="axle.col"
anchor="COUPLING_AXLE"/>
<element name="trailer" type="Matrix Physics" mass="20.0" velocity="0 0 0" omega
= "0 0 0" material="robot1" collision_geom="robot.col" anchor="TRAILER"/>
<element name="tr_left_wheel" type="Matrix Physics" mass="1.0" velocity="0 0 0"
omega="0 0 0" material="rubber" collision_geom="wheel.col"
anchor="TR_LEFT_WHEEL"/>

<element name="coupling_fixed" type="Physics Joint" joint_type="fixed" parent="robot"
child="coupling" pivot="0.0 0.0 0.0" direction="1 0 0" anchor="COUPLING"/>
<element name="coupling_axle_fixed" type="Physics Joint" joint_type="fixed"
parent="coupling_axle" child="trailer" pivot="0.0 0.0 0.0" direction="1 0 0"
anchor="COUPLING"/>
<element name="coupling_axle_ball" type="Physics Joint" joint_type="ball"
parent="coupling" child="coupling_axle" pivot="0.0 0.0 0.0" direction="0 0 1"
anchor="COUPLING"/>
<element name="tr_left_free" type="Physics Joint" joint_type="hinge"
parent="trailer" child="tr_left_wheel" actuation="none" pivot="0.0 0.0 0.0"
direction="0 1 0" motor="0 0" anchor="TR_LEFT_WHEEL"/>

```

Fig. 6. XML description for the dynamics simulation of the trailer shown in figure [7](#)

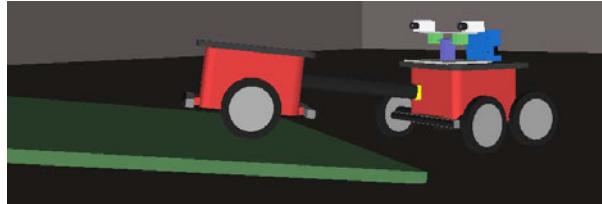


Fig. 7. Modified Pioneer like robot, now equipped with Ackermann steering, rear wheel drive, laser scanner, stereo camera, and trailer. The trailer is moving on an inclined plane to show the degrees of freedom using a ball joint at the coupling (yellow cylinder).

of the whole trailer. Due to the coupling axle the only stimulus comes from the robot that pulls the trailer through the working space. All these calculations are done by the physics engine automatically. Hence, enhancing the scene description file suffices for simulating the dynamics of the trailer.

6 Simulation of a Standard Vehicle Kinematic System

Before introducing the simulation of a bucket excavator as real world scenario in this section the Pioneer example is extended to a combination of Ackermann

```

<part file="wheel.iv" name="FL_WHEEL" anchor="ROBOT" offset="0.2 0.35 0.05 0 0 0"/>
<part file="steering.iv" name="FL_STEERING" anchor="ROBOT"
      offset="0.2 0.30 0.05 10 90 0"/>
<element name="fl_wheel" type="Matrix Physics" mass="1.0" velocity="0 0 0"
        omega="0 0 0" material="rubber" collision_geom="wheel.col" anchor="FL_WHEEL"/>
<element name="fl_steering" type="Matrix Physics" mass="0.2" velocity="0 0 0" omega
        ="0 0 0" material="robot1" collision_geom="steering.col" anchor="FL_STEERING"/>
<element name="left_free" type="Physics Joint" joint_type="hinge"
        parent="fl_steering" child="fl_wheel" actuation="none" pivot="0.0 0.0 0.0"
        direction="0 1 0" motor="0 0" anchor="FL_WHEEL"/>
<element name="fl_steering_joint" type="Physics Joint" joint_type="hinge"
        actuation="pos" motor="100 0" min="-0.4" max="0.4" parent="robot"
        child="fl_steering" pivot="-0.05 0.0 0.0" direction="1 0 0" anchor="FL_STEERING"/>

```

Fig. 8. XML description for the steerable front left wheel of the Ackermann vehicle

steering for the free-running front wheels and a rear wheel drive (fig. 7). This standard vehicle concept is exactly the same for both crafts. Hence, the two driven wheels of the differential drive are now mounted at the rear part of the robot and two free wheels as in the trailer example at the front part. To make the latter ones steerable there is a hinge joint between front wheel and robot. As this is the only new feature of this scenario the corresponding enhancements to the scene description are shown in fig. 8 again only for the left wheel.

The definition of the parts and collision models for the wheel and steering joint, a small cylinder serving as “wheel suspension”, are straight forward. The wheel is connected by an unactuated hinge joint `left_free` to the steering cylinder `fl_steering`. This cylinder is then mounted by an **actuated** hinge joint `fl_steering_joint` on the robot. The actuation type `pos` and corresponding maximum `motor` torque define the type of actuation, i. e. using an absolute angle in radians as input, that is valid for the given `[min, max]` interval. The `pivot` is used to adjust the actual center of rotation.

As these six joints in total (drive, free, steering) directly define the interface of the physics engine, there are only marginal changes of the simulation unit shown in fig. 5: the wheel velocities now affect the driven rear wheels, so only the two desired absolute angles for the left and right steering joint have to be provided as additional input. Output of the simulation are the encoder values for all four wheels and the actual steering angles. This way the simulation interface is again the same as for a corresponding real robot and the changes in the code compared to the differential drive example are limited to the exchange of additional data.

Naturally this vehicle concept requires a reasonable set of input values because the steering angles and wheel velocities have to match to each other to avoid mechanical deadlocks. I. e. in case of a differential drive “mismatching” velocities result in a turning motion of the robot whereas in the Ackermann case they may result in unexpected motions. At least effects of over- or understeering can be tested this way. The necessary calculation of the Ackermann kinematics in the control unit is out of the scope of this paper.

7 Real World Scenario: The Bucket Excavator

A complex real world application scenario is the autonomous bucket excavator project. The long-term goal is to let a real 18 ton wheeled bucket excavator from VOLVO Construction equipment (VOLVO EW-180C) perform landscaping tasks fully autonomous. As this machine can seriously harm human beings, buildings, or itself, a virtual environment is needed for performing safe offline tests. Hence, a SIMVIS3D simulation containing a realistically shaped excavator and environment model has been realized. Due to the high dynamic of the excavator NEWTON is used to simulate all the machine masses, undercarriage and boom joints, and environmental interaction forces. The vehicle kinematics are modeled the same way as for the robot described in section 6.



Fig. 9. Screenshot from the simulated test scenario showing the excavator including joints (green), interacting with the heightfield soil simulation

In addition to common obstacle avoidance capabilities this machine should also interact directly with the surface for moving soil from one point to another. For this purpose, a complex particle soil simulation running on an off-the-shelf GPU via the OpenCL¹³ interface has been implemented (see [8]) which delivers a visual heightfield back to the simulation environment. Additionally, the material resistance is handled by delivering a bucket-soil interaction force vector to the NEWTON physics engine. Figure 9 shows a screenshot of the simulated outdoor environment.

8 Conclusion

In this paper the simulation of wheel driven vehicles with SIMVIS3D and NEWTON has been discussed. The comparison of simulating a differential drive with

¹³ OpenCL is the open standard for parallel programming of heterogeneous systems by the Khronos group: <http://www.khronos.org/opencl>

SIMVIS3D only and by using a physics engine has outlined the benefits of the latter one: ease of use, reduction of user code, and improvement of realism due to the simulation of vehicle dynamics. This point has been further stressed by the trailer as passive vehicle that is simulated *automatically* by the physics engine after being defined in the scene description file. Finally, the Ackermann example has proven the flexibility of the whole simulation framework: standard concepts of vehicle kinematics can be realized straight forward and – using arbitrary OIV/VRML models – real world machines as a bucket excavator can be simulated in a realistic manner. Furthermore, the extension of simulation modalities via external components, e. g. walking people, spread of sound, or shapeable soil has been sketched roughly.

Future work concentrates on improving the simulation of vehicle kinematics: completely modeling the Ackermann steering linkage and power train with differential will just require a position of the steering motor and rotation velocity of the drive motor as simulation input. Then the Ackermann kinematics do not have to be calculated manually in the control system anymore. Besides, other kinematic concepts as chain drives have to be realized.

References

1. Braun, T., Wettach, J., Berns, K.: A customizable, multi-host simulation and visualization framework for robot applications. In: ICAR 2007 (2007)
2. Deines, E.: Acoustic Simulation and Visualization Algorithms. Ph.D. thesis, University of Kaiserslautern (2008)
3. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: IROS 2004 (2004)
4. Koestler, A., Braeunl, T.: Mobile robot simulation with realistic error models. In: ICARA 2004 (2004)
5. Laue, T., Spiess, K., Refer, T.: A general physical robot simulator and its application in robocup. In: RoboCup Symposium 2005 (2005)
6. Luksch, T.: Human-like Control of Dynamically Walking Bipedal Robots. Ph.D. thesis, University of Kaiserslautern (to be published, 2010)
7. Michel, O.: Webots: Professional mobile robot simulation. Int. Journal Of Advanced Robotic Systems (2004)
8. Schmidt, D., Proetzsch, M., Berns, K.: Simulation and control of an autonomous bucket excavator for landscaping tasks. In: ICRA 2010 (2010)
9. Schmitz, N., Hirth, J., Berns, K.: A simulation framework for human-robot interaction. In: ACHI 2010 (2010)
10. Wang, J., Lewis, M., Gennari, J.: A game engine based simulation of the nist urban search & rescue arenas. In: Winter Simulation Conference 2003 (2003)

Coordinating Software Components in a Component-Based Architecture for Robotics

Geoffrey Biggs, Noriaki Ando, and Tetsuo Kotoku

Intelligent Systems Research Institute

National Institute of Advanced Industrial Science and Technology (AIST)
AIST Tsukuba Central 2, Tsukuba, Ibaraki 305-8568, Japan

Abstract. Component-based software is a major design trend in robot software. It brings many benefits to system design, implementation and maintenance. One step in using component-based methods in designing the structure of a robot program is managing the components and the connections between them over time, known as coordination. In this paper we present a framework for coordinating component networks using the OpenRTM-aist software architecture, implemented using the concurrent Erlang language. The framework provides a coordination system that mimics the internal state-change notification system of OpenRTM-aist. Rather than being a fixed-structure coordinator, it allows robot developers to implement a coordinator matching the style of coordination they need. This paper shows that Erlang has potential in robotics.

1 Introduction

Component-based software design and implementation is a current trend in software engineering. Software is divided into individual components, each with a well-defined interface that specifies what functionality that component provides. Multiple software components are combined together into a complete software system [13]. Using this design methodology, robot developers can create complete robot systems from off-the-shelf software components as easily as complete electric circuits can be created from hardware components.

Component-based practices bring many benefits to software design, implementation, maintenance and reuse, including known interfaces that act as “contracts” between components, “separation of concerns” (each component only deals with its individual problem), isolation testing, and rapid development of new systems using existing commoditised software resources.

These benefits also apply to the design, implementation, maintenance and reuse of robot software. As a result, component-based software is a major trend in robotics.

An issue that all robot developers faces is the coordination of behaviours, and so in turn the coordination of the software. Coordination is important to allow the robot’s software to adapt to changes in the robot’s state as it carries out its various tasks. Before the recent rise of flexible component-based architectures,

architectures with fixed structure, often layered, were popular in robotics. In these, a higher layer manages the actions of a lower layer to provide coordination according to some generated plan. Often, the plan itself is generated by an even higher layer, although some architectures, such as CLARAty [7], intentionally do not use this approach.

In this paper, we present a coordination framework for the OpenRTM-aist component-based architecture. It is a framework rather than a complete coordinator because it provides the facilities for programmers to create their own coordination systems. Rather than a fixed coordination style, programmers are free to use whichever style suits their needs. We use the concurrent Erlang language to implement the framework in order to test its applicability in robotics.

The next section discusses coordination methods commonly used in robotics. Section 3 describes OpenRTM-aist, the architecture for which the coordination framework has been designed. The coordination framework itself is described in section 4. Discussion is given in section 5, and conclusions in section 6.

2 Coordinating Robot Software

Coordination has a long history in robot software. It has always been necessary to manage the actions of a robot in order to achieve complex goals. It is particularly common to see coordination play a major role in layered architectures. Often, a layered architecture will feature a low-level layer that consists of chunks of functionality, and at a higher level some kind of coordination system controlling the execution of these chunks to meet some goal.

Despite their strong coordination support, no layered architectures have managed to become widely used.

On the other hand, recent years have seen the popularisation of more flexible component-based software architectures for robotics. These architectures allow designers to create *component networks*. Rather than being layered, the network of components is effectively a single layer of individual programs communicating by one or more methods of transporting data. Such an architecture style is very heterogeneous and adaptable to the needs of the programmer.

Examples of these architectures include OpenRTM-aist [1], ORCA2 [3], ROS [9], ERIC'S [4] and OPRoS [2]. In each case, systems built using the architecture rely on networks of connected components with data flowing amongst them. The behaviour of the robot is represented by what data ultimately arrives at the components responsible for controlling actuators. The shaping of this data, and so determining the robot's current behaviour, is performed by the components that make up the component network between sensor components and actuator components. The reader may notice that this is similar to coordinating the actions of a lower layer in layered architectures.

Coordination in a component-based system therefore requires changing either the internal behaviour of the individual components, or changing part of or the whole component network.

None of the architectures mentioned above currently provide an automatic coordination method to alter the component network at run-time. Only two of them provide facilities for manually altering a running component network. OpenRTM-aist and OPRoS, both at least partially based on the same software specification, allow for “ports” (the points on components at which communication occurs) to be connected and disconnected by external tools at run time.

The lack of a coordinator in general component-based software is understandable. The component-based design paradigm was originally aimed at static software systems, such as business software. A component-based system, once in place, was not expected to change. This is generally the case for component-based systems today. The need to dynamically alter a component network at run-time arose as the component-based paradigm began to be applied to more dynamic systems, such as factory automation and robotics.

Previous work in robotics has produced many coordinators aimed at task management. Examples include Colbert [6] and TDL [11], both designed for specifying and controlling reactive components of a hybrid architecture. Another is the Reactive Model-based Programming Language [14], which is a synchronous system for coordination of embedded systems using a deductive reasoning engine.

3 OpenRTM-aist

OpenRTM-aist is a component-based architecture for intelligent systems, known in Japan as “Robot Technology,” or “RT.” OpenRTM-aist implements the Robot Technology Component (RTC) specification [10], which defines a common introspection interface. Introspection is an important part of OpenRTM-aist, providing the basis upon which tools build to interact with and manage systems using the architecture.

The central concept in OpenRTM-aist is the RT Component, or RTC. Each RTC has an internal state machine with known states. The component’s health can be both monitored through the introspection interface. Components begin executing when they are activated, entering the *active* state. Execution is terminated by deactivating the component, moving it to the *inactive* state. The state machine can also enter the *error* state.

OpenRTM-aist supports data-flow- and request-based communications, with interaction between components occurring at “ports.” The component network is formed by making connections between the ports of the components.

OpenRTM-aist can use manager daemons to load and manage components dynamically on remote nodes. This point is important for the dynamic management of component networks in which components may come and go as needed.

OpenRTM-aist uses CORBA [5] to implement the introspection interface. Components must register on a known name server (there can be more than one known name server in use). Typically, components are organised on the name server using naming contexts below the root context in order to provide hierarchical categorisation.

3.1 Managing RT Systems

As mentioned above, introspection interfaces are an important feature of RT Systems. It is through these interfaces that components are monitored and managed, connections are created and removed, and the RT System as a whole is managed.

This introspection interface provides completely dynamic access to an RT System at run-time. Changes, both into the interface and out of it, are reflected immediately by the receiving end. This dynamic control over the component network is what allows for external run-time coordination of the network.

OpenRTM-aist also features in its implementation a large number of internally-accessible callback functions. These functions are used by the components themselves to react to changes in the network around them and in the component itself.

For example, when a connection is made between components, a callback is triggered in each component involved. The components can use this callback for any purpose the component designer deems suitable. One example is having the component respond to a new connection on an output port by activating a part of its behaviour that produces data. Prior to a connection existing, that part of the component can remain dormant, reducing resource consumption. Another example is a callback triggered when data is received on a port, which a component can use to perform out-of-band processing on data rather than processing it within its behaviour loop.

These callbacks provide a considerably more reactive interface than the externally-accessible introspection interface. We can use them to give a component some intelligence about its behaviour based on its abstract view of the state of the network around it. Unfortunately, they cannot be accessed externally to a component. We are unable to use them for external coordination of the component network. We have therefore created an external coordination framework that mimics these callbacks.

4 Coordinating RT Components

A coordination framework has been implemented for OpenRTM-aist. It gives developers the necessary functionality to support external, fine-grained reactive and pro-active coordination of an RT System.

The coordination framework has been integrated into the “rtctree-erl” library, implemented in Erlang [2]. It consists of a set of user-definable callbacks. The following section describes rtctree-erl, while the callbacks that allow coordination are described in Section 4.2. An example of using the framework is given in Section 4.3.

4.1 rtctree-erl

This library provides the low-level framework into which the coordination callbacks are integrated. Its purpose is to provide an API for the OpenRTM-aist introspection interfaces in Erlang.

`rtctree-erl` uses a tree structure to represent the component network and related OpenRTM-aist objects, such as managers. An example of this structure is shown in Figure 11. Below the root node are all the known name servers. Name servers may have multiple naming contexts to which objects can register for organisational purposes. Developers can use naming contexts to impose an organisation on the running components.

Within the `rtctree-erl` library, each node in the tree is represented by a separate Erlang process. We take advantage of Erlang's light-weight processes to allow concurrent access to any part of the tree (a single Erlang VM can support thousands of processes on an average desktop computer). Each node acts independently of the others, communicating using Erlang's message-passing communication mechanism (Erlang processes cannot share memory, in order to provide robustness).

The processes communicate with OpenRTM-aist distributed objects using the OpenRTM-aist CORBA introspection interface. The library uses Orber [8], the Erlang CORBA implementation, to make the remote procedure calls.

Because Erlang is derived from functional programming languages, the only way to maintain data is by passing it from function to function. Each node in the RTC tree follows this pattern, maintaining a structure specific to its node type. In keeping with Erlang conventions, clients of the library do not directly use this data structure. Rather, they make remote calls to the process containing the data structure, sending a message corresponding to the operation to be performed. The calling process typically waits for the result, but it is possible to continue execution and check for a result at a later time. This is a key feature of Erlang's method passing that increases the flexibility of its RPC. The `rtctree-erl` library provides an API for each node type that hides the RPC code and provides what appears to be a traditional function-based API to clients of the library.

RT Components only exist as leaves of the tree. The node representing a component contains a cache of the component's information, retrieved through the introspection interfaces of OpenRTM-aist. This information can be updated at any time by requesting an update of all or a part of the cache, which will cause the node process to make a new remote procedure call and get the latest value. It is not updated automatically by default (although see the next section for an exception) as it does not change regularly; `rtctree-erl` assumes that the developer knows best how often to update the information.

A component's node may also contain child processes representing its ports, execution contexts and configuration sets. These are represented using their own processes to allow for concurrent access to ports as well as components. Like the component nodes, port and execution context nodes cache information retrieved through the introspection interfaces.

Using a tree structure allows additional tree-structure operations to be performed, such as iterating over every node in the tree to perform an operation on every node matching certain criteria.

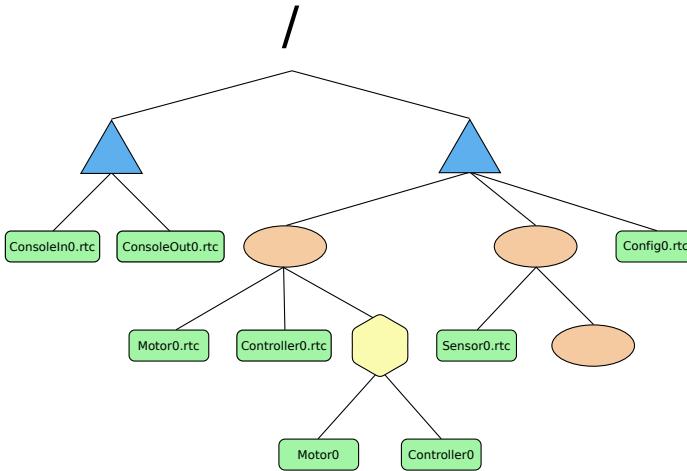


Fig. 1. An example of the tree structure used by the *rtctree-erl* library, dubbed the *RTC Tree*. The root node is at the top of the tree. The blue triangles are naming service nodes. Red ovals are naming contexts below a root context. Green boxes are RT Components. The yellow hexagon is a manager. Below it are references to the two RT Components it is managing.

4.2 External Callbacks

The *rtctree-erl* library provides easy access to the introspection interfaces of OpenRTM-aist, as though accessing an API on the objects directly. However, writing coordination code with it is not so simple because the introspection interfaces do not provide a reactive interface. A coordinator would need to manually check important conditions.

On the other hand, OpenRTM-aist provides a large set of internal callbacks for components to use, a small sample of which is shown in Table II. Having this fine-grained information available externally would make implementing powerful coordination systems easier.

Table 1. A sample of the callbacks available internally in OpenRTM-aist, and those provided by *rtctree-erl*

Internal callback	External callback Purpose (module:type)	
onActivated, onDeactivated, component:state		Notify of changes in a component's state.
onError		
onConnect	port:connect	Notify of a new connection on a port.
onDisconnect	port:disconnect	Notify of the removal of a connection from a port.
OnUpdateParamCallback	configuration: update_param	Notify of a configuration parameter being updated.

We have implemented a set of external callbacks that use the introspection interfaces to mimic the internal OpenRTM-aist callbacks. A sample of these external callbacks is given in Table 1. Not all of the internal callbacks are implemented externally; in some cases the information is not available externally in any way. For example, information relating to data being received cannot be detected externally.

The framework acts as an automated intermediary between coordinators and the rtctree-erl library’s introspection APIs. The callbacks are triggered automatically when appropriate. Developers can write their coordinators to be reactive to the changing state of the robot software system.

The callbacks are implemented in the component node, port, execution context and configuration set processes of the rtctree-erl library. The developer must implement callback functions and then register them in the process of interest, indicating the piece of state information the callback is for. For example, a developer can write a function to handle a component entering the *error* state, then register it with a component process as a callback for state changes.

The component and node processes monitor the state information relevant to the callbacks that are currently active. When this state information changes, the registered callback functions are called. (We term the process that triggered the callback the “source process.”) Callbacks are passed:

- The relevant piece of new state information,
- What the value was before the change,
- The process identifier (PID) of the component or port process that triggered the callback, and
- Any extra data relevant to the specific callback.

Callbacks are executed in a separate process from the source process. One reason for this is to prevent callbacks from blocking the source process from triggering other callbacks by taking too long to execute. However, the primary reason is to prevent deadlocks: if the source process is executing the callback, the callback cannot use the RPC-based API to manipulate the OpenRTM-aist object that triggered the callback. Instead, the source spawns a worker process to handle each callback it triggers. We take advantage of Erlang’s light-weight processes and rapid process spawning here.

The next section uses an example to illustrate the use of the framework to coordinate an event in a robot software system. Through the example, it gives further details on how the framework functions.

4.3 Example: Re-configuring a Localisation System

In this example, we are coordinating the simple RT System shown in Figure 2, a localisation system utilising a laser, gyro and odometry. Two localisation components are available, with the laser-based component in use when the robot starts. Only one is active at a time.

We monitor the system for faults in the laser. When a fault occurs, we need to switch from the laser-based localisation component to one based solely on the

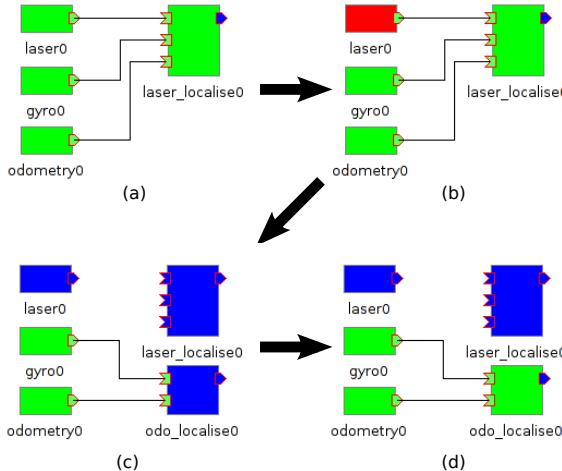


Fig. 2. The changes in the component network of the example. (a) The components operating as normal. (b) An error occurs in the laser component. (c) The connections to the failed component are removed, a new localisation component is instantiated and connected. (d) The new localisation component is activated.

gyro and odometry. We must shut down the laser-based localisation component, create the odometry/gyro localisation component, form new connections, and activate the new component. The code to do this is shown in Listing 11.

First the coordination program starts the framework (Line 6). Libraries in Erlang are implemented as “applications,” providing an API for accessing their functionality and operating concurrently to client programs. The coordination program then registers the callbacks it is interested in (Lines 8 to 9). The coordinator retrieves the node of interest from the RTC Tree using its path, then registers its callback function with this node. It provides the callback function object, the type of callback it is registering (a state-change callback, in this case), and the rate at which to update the state from the remote object.

The coordinator is now fully-activated. The starting process is free to exit at this point; the callback functions will be executed in their own processes.

The callback function is shown on lines 12 to 30. Note the rule-like pattern matching ensuring that this callback only executes when the new state is the error state, and the triggering component is the component of interest. It uses the `rtctree-erl` library APIs to handle the event, replacing the failed laser node and the now-unusable localisation component with a new localisation component. Note particularly line 20, where a new component is instantiated dynamically.

Figure 3 provides a sequence diagram showing the flow of control amongst the various processes involved in this example. Figure 2 shows the component network over time during the example.

Listing 1.1. An example callback using the framework. The callback setup is just three lines of code; adding additional callbacks would require one to two lines per callback at most. The callback itself uses the rtctree-erl library’s API to manipulate the component network in response to the laser node suffering a failure.

```

1 -module(example).
2 -export([run/0, cb/4]).
3 -include("rtctree-erl/include/nodes.hrl").

5 run() ->
6     ok = rtctree:start(),
7     ok = rtctree:add_servers(["localhost"]),
8     {ok, C} = rtctree:get_node_by_path(["/", "localhost",
9         "laser0.rtc"]),
10    component:add_cb(C, fun state_change/4, state, 1000).

11
12 state_change('ERROR_STATE', _Old, C, _Extra) ->
13     % Disconnect and shut down the old localiser and laser
14     ok = component:reset(C, 1),
15     {ok, Loc} = rtctree:get_node_by_path(["/", "localhost",
16         "laser_localise0.rtc"]),
17     ok = component:disconnect_all(Loc),
18     ok = component:deactivate(Loc, 1),
19     % Instantiate the new localiser
20     {ok, M} = rtctree:get_node_by_path(["/", "localhost",
21         "manager.mgr"]),
22     ok = manager:create_comp(M, "odo_localise"),
23     % Connect the new localiser
24     rtctree:connect_by_path(["/", "localhost",
25         "odometry0.rtc", "odo"]),
26     [{"/", "localhost", "odo_localise0.rtc", "odo"}]),
27     rtctree:connect_by_path(["/", "localhost", "gyro0.rtc",
28         "gyro"]),
29     [{"/", "localhost", "odo_localise0.rtc", "gyro"}]),
30     % Activate the new localiser
31     {ok, OdoLoc} = rtctree:get_node_by_path(["/",
32         "localhost", "odo_localise0.rtc"]),
33     component:activate(OdoLoc, 1);
34 state_change(_, _, _, _) ->
35     ok.

```

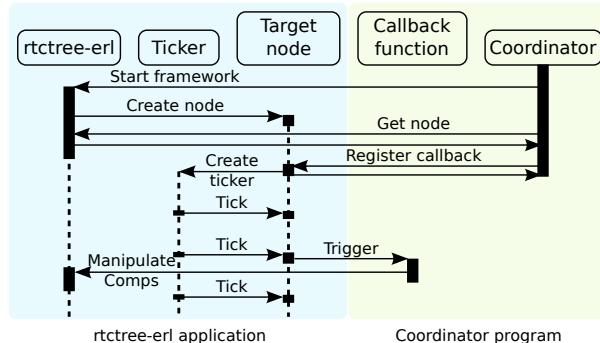


Fig. 3. The processes involved in using a single callback coordinator. Note how the processes appear and disappear as needed; when no events need management, the coordinator itself requires no resources and rtctree-erl only requires enough to regularly update the relevant status.

5 Discussion

The framework presented is designed to be flexible. The goal is to allow robot developers to implement coordinators that meet their needs. The callback system allows for continuous monitoring of any aspect of an RT System. Its speed is limited by the response of the CORBA introspection interfaces.

This callback system provides flexibility to coordinator implementers. For example, it is easy to register a callback that is called whenever the state of a component changes. Sample code showing this is given in Listing 1, which shows waiting for a component to go into the error state. Callbacks can also be registered that are called when a new connection is created on a port, when a manager creates a new component, and so on. Combining this framework with direct usage of the rtctree-erl library provides control over the network being monitored, such as re-arranging the component network when an error occurs in a component.

Because the rtctree-erl library, the coordination framework and coordinators are all implemented in the inherently-thread-safe Erlang programming language, there are no concerns with using such a large number of processes. However, this is a framework, and so it is up to the programmer to ensure their coordinator design will not lead to, for example, conflicting callbacks. While Erlang supports soft real-time, it is up to the programmer to ensure their callbacks will not execute for too long. The framework does not check if a callback is currently running if the next tick causes it to be triggered again. Hard real-time is not possible using Erlang, which may limit the applications of this system.

This coordination framework is more dynamic than previous methods. We no longer need to have every component we may potentially use executing and a part of the component network all the time. Instead, we can start and stop

components as necessary in response to changes in robot state. This preserves system resources and simplifies the component network.

This framework compares favourably to the languages mentioned in Section 2. Its primary benefit is that it uses an existing language rather than inventing a new one, simplifying the framework development and providing it with the power of a complete language. Despite this, it still has a rule-like syntax similar to custom-designed languages. In addition, both Colbert and TDL are designed for *specifying* tasks as well as coordinating them. This framework is purely for coordination of existing components. The difference with RMPL is that this framework is not synchronous. While this limits its use at lower levels for fine control, it is suitable for higher-level planning tasks.

The most important contribution of this work is showing that the Erlang syntax is a benefit when writing coordinators. Erlang's function declarations use a pattern-matching system. This makes them look like rules. We benefit from this in our coordinator framework. As the example shows, we can write the function declaration of the callback functions as a set of rules. The callback in the example is a rule for a component moving to the error state. We gain this syntax benefit without needing to create or modify a language. Erlang has relatively wide support in industry, and has been in use in industrial applications, particularly telephony, for nearly 25 years [2].

6 Conclusions

Component-based software architectures are becoming popular in robotics. They lead to more flexible robot software structures. Coordination of the component networks to adapt to changing situations and changes in robot state is still a topic of research. Many popular frameworks do not provide an externally-accessible interface for the management of components and the connections between them, making it difficult to create an external coordinator tool.

The OpenRTM-aist architecture features fully-dynamic connections and components. It does not yet have a coordination tool, and its internal state-monitoring callbacks are also more capable than its external introspection interfaces.

We have taken advantage of the introspection interfaces to create a coordination framework that provides the same fine-grained monitoring and control as the internal monitoring callbacks. The framework is implemented on top of the rtctree-erl library. It is implemented in Erlang, a concurrent programming language. It uses features of Erlang to gain concurrency support and robustness. The framework shows that Erlang has much to offer robotics in concurrency support, robustness and error-handling, and in syntax.

Acknowledgements

This research was supported by a JSPS *kakenhi* grant 21700230.

References

1. Ando, N., Suehiro, T., Kotoku, T.: A software platform for component based rt-system development: Openrtmaist. In: Carpin, S., Noda, I., Pagello, E., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2008. LNCS (LNAI), vol. 5325, pp. 87–98. Springer, Heidelberg (2008)
2. Armstrong, J.: Programming Erlang: Software for a Concurrent World. Pragmatic Bookshelf (2007)
3. Brooks, A., Kaupp, T., Makarenko, A., Williams, S., Oreback, A.: Towards component-based robotics. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2005, pp. 163–168 (August 2005)
4. Bruyninckx, H.: Open robot control software: the orocos project. In: Proceedings of IEEE International Conference on Robotics and Automation, ICRA 2001, vol. 3, pp. 2523–2528 (2001)
5. Henning, M., Vinoski, S.: Advanced CORBA Programming with C++. Addison-Wesley Professional, Reading (1999)
6. Konolige, K.: COLBERT: A language for reactive control in sapphira. In: Brewka, G., Habel, C., Nebel, B. (eds.) KI 1997. LNCS (LNAI), vol. 1303, pp. 31–52. Springer, Heidelberg (1997)
7. Nesnas, I.A.D., Wright, A., Bajracharya, M., Simmons, R., Estlin, T.: Claraty and challenges of developing interoperable robotic software. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2003, Las Vegas, Nevada, vol. 3, pp. 2428–2435 (October 2003)
8. Erlang – Orber Reference Manual (2010),
<http://www.erlang.org/doc/apps/orber/index.html>
9. ROS Wiki (2010), <http://www.ros.org>
10. The Robotic Technology Component Specification - Final Adopted Specification (2010), http://www.omg.org/technology/documents/spec_catalog.htm
11. Simmons, R., Apfelbaum, D.: A task description language for robot control. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 3, pp. 1931–1937 (1998)
12. Song, B., Jung, S., Jang, C., Kim, S.: An Introduction to Robot Component Model for OPRoS (Open Platform for Robotic Services). In: Workshop Proceedings of Intl. Conf. on Simulation, Modeling and Programming for Autonomous Robots 2008, pp. 592–603 (November 2008)
13. Szyperski, C., Gruntz, D., Murer, S.: Component Software – Beyond Object-Oriented Programming, 2nd edn. Addison-Wesley/ACM Press (2002)
14. Williams, B.C., Ingham, M.D., Chung, S.H., Elliott, P.H.: Model-based programming of intelligent embedded systems and robotic space explorers. Proceedings of the IEEE: Special Issue on Modeling and Design of Embedded Software 91(3), 212–237 (2003)

Native Robot Software Framework Inter-operation

Geoffrey Biggs, Noriaki Ando, and Tetsuo Kotoku

Intelligent Systems Research Institute
National Institute of Advanced Industrial Science and Technology (AIST)
AIST Tsukuba Central 2, Tsukuba, Ibaraki 305-8568, Japan

Abstract. Component-based software is a major recent design trend in robotics. It brings many benefits to system design, implementation, maintenance and architectural flexibility. While previous methods of structuring software led to monolithic robot software incapable of easily using multiple architectures, the component-based approach, combined with some suitable software engineering decisions, opens new possibilities for interaction amongst architectures. Rather than using a single architecture for an entire robot system, we now have the opportunity to use the best architecture for the job in each part of the robot without inefficient wrapping or translation. For example, a real-time architecture for actuator control and a planning-based architecture for intelligence. In this paper, we use the example of a native ROS transport for the OpenRTM-aist architecture to illustrate the benefits of allowing architectures to interact in this way.

1 Introduction

A component-based software system, built from the ground up using a component-based middleware such as CORBA, will only have one method of communicating between components. This has been the traditional approach to component-based software since the concept was conceived. The system is designed around the architecture to be used.

In robotics, it has recently become popular to create a component-based framework, either with or without the support of an existing middleware. The framework's goal is to reduce the time for creating a new software system and to foster greater re-use of software resources.

Unfortunately, software created using a framework is only re-usable within that framework. This is not true re-use [2]. It is common for a robot developer to find functional software they wish to use, only to discover it was written for a different framework than the one chosen by the developer.

Fortunately, the component-based approach to software development lends itself well to greater inter-operation between different frameworks. It is common for a component-based framework to abstract away the details of the transportation of data from one component to another. For example, the OpenRTM-aist framework [1] provides a one-way data flow port abstraction which, assuming

the same serialisation method is used, supports any method of moving the data between two ports. At a higher level, it also provides an abstraction for the basic concept of a port, allowing transports with any semantics to be created.

We can take advantage of this abstraction to integrate support for other frameworks directly in OpenRTM-aist. This paper describes the implementation of support for the ROS framework’s transport [7]. The implementation provides native support for communicating between OpenRTM-aist components and ROS nodes. No wrappers or protocol translators are required, ensuring no loss in efficiency for inter-operation between OpenRTM-aist and ROS.

The next section discusses previous methods of implementing inter-operation between frameworks. Section 3 describes how ROS support was added to OpenRTM-aist. Section 4 discusses the support, including what advantages it brings. Conclusions are given in Section 5.

2 Architecture Interaction

The continued development of a range of robot frameworks, with their own unique properties, has produced a continuing desire to use frameworks together. While a robot software system will likely be mainly using one framework, it may be desirable to use another at the same time in some parts of the software. Reasons for this can include:

1. The other framework includes functional software that the authors of the robot software system wish to use, without needing to rewrite or otherwise modify that software.
2. The other framework is more suited to some part of the robot under development. For example, the main framework may not support hard real-time, while another framework does. Using the other framework in the real-time control loops of the robot becomes desirable. We view this as important in future robot development as various frameworks arise with differing capabilities.

In practical experience, the first is more common. Researchers tend to develop their functional software heavily integrated into their framework of choice. When another researcher wishes to use it, they must choose either to fork the software in order to use it directly, make their framework of choice interact with the framework containing the desired functional software, or change the framework used in the rest of the system. The third option is obviously the least desirable, and may even be impossible. The first option requires a large time commitment, making the second option the most desirable. This option saves development time and is less likely to introduce new bugs into existing software.

Previous approaches to architecture interaction have included wrappers and translators, which are very similar.

In the following sections, we use the term “client” to refer to the software written by the robot developer that is executing in the primary framework. This will typically be the body of a component. The term “primary framework” indicates the framework primarily being used in the development of a robot,

while “secondary framework” refers to another framework being utilised to gain some benefit that it offers for a specific part of the robot, such as real-time capability.

2.1 Wrappers

A wrapper is the most straight-forward approach to making one framework interact with another. The goal of a wrapper is to “wrap” the desired behaviour of the secondary framework in a component for the primary framework. It creates the appearance of providing that wrapped behaviour natively in the primary framework. A wrapper could also be called a mimic component. Illustrations of this method are shown in Figures 1 and 2.

Using a wrapper to enable communication between two frameworks involves treating the secondary framework as a software resource to be used by client software of the primary framework. This client software forms the wrapper. It is responsible for servicing requests from other components of the primary framework. In order to do this, it communicates with the desired behaviour of the secondary framework as with any other software or hardware resource. The secondary framework is not a first-class entity of the complete software system.

A wrapper may also be more of a translator, attempting to mediate between the two frameworks at a level closer to the protocol than at the software functionality level.

The largest problem with using a wrapper is that it introduces inefficiencies into the robot system. Even if the data types match exactly (a rare occurrence), data must be read from one connection and written to another.

Wrappers are typically specific to the data type or behaviour of the secondary framework that they are wrapping. This means that multiple wrappers are usually necessary to support all the possible behaviour offered by the secondary framework - a somewhat daunting and unreasonable time sink for any robot developer. Maintenance is also difficult. Any changes to the secondary framework require updating the corresponding wrappers. Practical experience has shown that wrappers typically become out-of-date and unusable within a short period of time. These challenges to robot developers make wrappers less than ideal in robot development and restrict the free use of multiple frameworks.

Numerous wrappers have been created over the years. They are not typically afforded much fanfare. A set of examples can be found in the library of components for the ORCA2 framework. There are several components present that wrap interfaces of the Player framework. They provide services to other ORCA2 components, the behaviour of which is defined by the Player interfaces that they wrap. A specific example is the `LaserScanner2dPlayerClient` driver¹. This provides the behaviour of a two-dimensional laser scanner, with the actual behaviour implemented on the other side of the divide between ORCA2 and Player.

Another example of the wrapper approach is the MARIE project [5]. This project was begun specifically with the goal of allowing the multiple robot

¹ http://orca-robotics.sourceforge.net/group__hydro__driver__laserscanner2dplayerclient.html

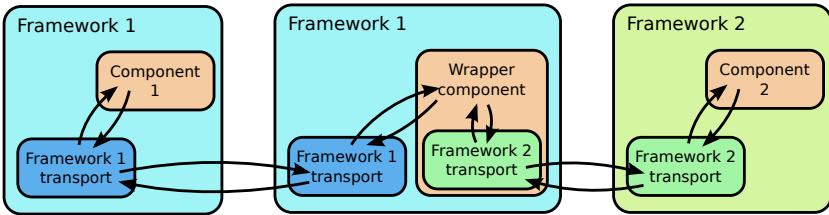


Fig. 1. Using a wrapper component is the most common approach to connecting two architectures. A separate component in the primary framework uses the secondary framework as though using any other software or hardware resource.

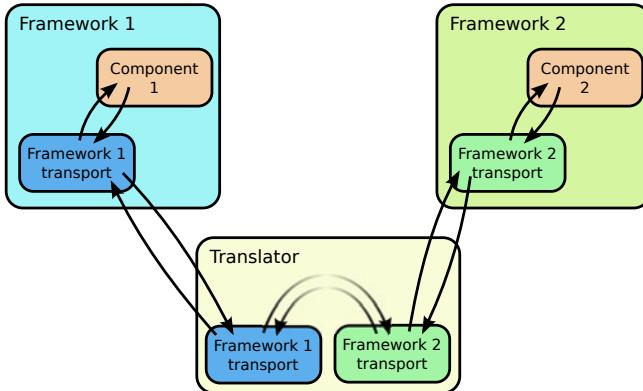


Fig. 2. A translator is a variation of a wrapper component. However, it is not a direct user of either framework, unlike a wrapper. It merely translates the protocols.

software frameworks that existed at the time to interact. It features a large collection of wrappers for various robot software frameworks, including Player and CARMEN.

2.2 Multiple Transports

A “transport” in a component-based software system is the method of communication between two separate components. The transport is responsible for ensuring data is moved from a source to a destination. This may involve serialisation of the data and transmission across a network using an underlying network protocol. Transports may provide one-way communication or a request-reply form of communication, in which case the transport may also be responsible for ensuring a reply of some kind is delivered to the requester.

A framework is not limited to a single transport. Multiple transports are often supported. The concept of multiple transports is not new. The Player framework has featured both TCP and UDP transports since 2004 [4]. Supporting multiple

transports allows the best transport for a link between two components to be chosen. In the case of Player, the reliable TCP transport can be used in parts of the robot system where all messages must be delivered. The unreliable UDP transport can be used when missing some messages will not degrade the robot's performance - gaining the performance improvement that UDP offers in those situations.

The multiple transports concept has not yet been widely exploited for allowing communication between frameworks. It does, however, offer many advantages towards achieving this goal:

- Unlike a wrapper component, transport management code exists outside the scope of the client code. The cognitive load on the developer for utilising the alternative transport is reduced.
- No translation is necessary. The data from the secondary framework can be utilised directly in the primary framework's components.
- No extra software components existing solely to translate between frameworks are necessary. The removal of an extra communication step and change in protocol removes an inefficiency present in wrappers.
- In an ideal design, the transports can be swapped transparently to the client code. This increases the re-usability of components while maintaining flexibility to communicate with other frameworks. (We discuss what is necessary to achieve this in Section 5 below.)
- Less maintenance overhead: changes in the API of the secondary framework require changes to the primary framework's use of the transport, but these should not affect client code. The wrapper approach is effectively crippled by changes in the secondary framework.

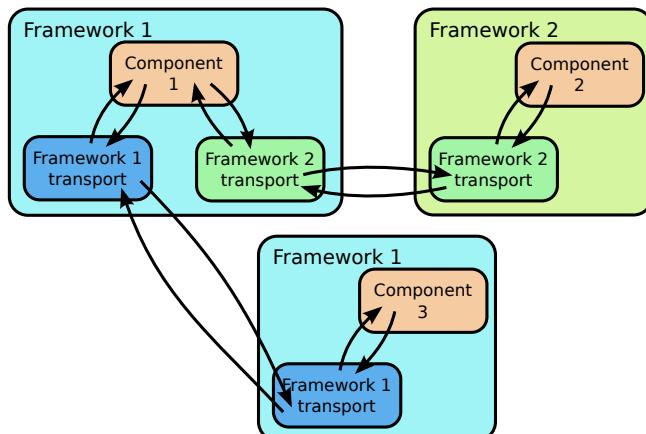


Fig. 3. Multiple transports supported within a framework allow components to directly speak the language of the peer they are communicating with. This structure applies just as well when only one framework with multiple transport types are involved.

Figure 3 illustrates how a single framework can use multiple transports to support communication with components running both the same framework and other frameworks.

The next section describes the implementation of a ROS transport for the OpenRTM-aist framework. The benefits this transport brings to both OpenRTM-aist and ROS are discussed in Section 4.

3 Native ROS Transport for OpenRTM-aist

This section describes the addition of a native transport for the ROS framework to the OpenRTM-aist framework.

3.1 OpenRTM-aist

OpenRTM-aist [1] is a component-based framework for intelligent systems. It features a component model that governs the life-cycle of each component, managers for controlling deployment of components, a powerful properties system for fine control over component configurations, and an in-depth introspection system. The introspection system complies with the Robot Technology Component standard from the Object Management Group [8], allowing OpenRTM-aist to work with any introspection tool compliant with this standard, such as the RTSystemEditor tool [9] used to manage RT Systems. The introspection system provides both inspection and control.

OpenRTM-aist has two built-in transports, both CORBA-based. The first is a basic request-reply transport. This effectively exposes a CORBA interface implementation from one component for other components to use as clients. The second transport is a one-way transport, called the data-flow transport. It uses CORBA's RPC mechanisms to effect a transfer of data from a provider to a consumer. Most importantly for the purposes of this discussion, the data-flow transport is point-to-point communication.

OpenRTM-aist relies on “ports” to support its transports. A port is defined in the RTC standard as a communication point; no further detail is given as to how that communication is carried out, making the introspection system ignorant to the transport details. (This does not stop the introspection from configuring transports, as the introspection interface can carry abstract properties to a transport for configuration purposes). Within a component implementation, a port is an object that provides a proxy for the transport connection to allow reading and/or writing of data. The abstraction of the transport implementation from the rest of the framework is what allows us to so easily implement native transports for interacting with other frameworks.

OpenRTM-aist’s component model supports hard real-time control. A component in OpenRTM-aist is known as an RT Component.

3.2 ROS

ROS, the Robot Operating System, is a relatively new component-based framework [7]. It is the opposite of OpenRTM-aist in that it provides no component

model or life-cycle, preferring instead to grant the developer as much freedom in component structure as possible.

Like OpenRTM-aist, ROS has a request-reply transport and a one-way transport. The request-reply transport exposes call-back functions in the callee to callers.

The one-way transport differs significantly from OpenRTM-aist's one-way transport. Rather than being point-to-point, it is semantically channel-based communication. Publishers place data into a channel, and any components subscribed to that channel will receive it. The channel can be persistent: if a component subscribes after the last data was written into the channel, it can still retrieve that value. This is an important capability in robotics, where irregularly-and infrequently-updating data is common. For example, a map provider.

ROS provides introspection for its communications, allowing the existing channels to be inspected from introspection tools. ROS does not directly support any form of real-time.

3.3 Native ROS Transport

As the previous sections noted, OpenRTM-aist and ROS differ in a number of ways. Most importantly from the point of view of OpenRTM-aist is that ROS provides a persistent channel-based transport, a feature with wide benefits in robotics. ROS also has a very large library of functional robot software that, were it available to OpenRTM-aist, would be very useful.

From the perspective of ROS, the hard real-time capability of OpenRTM-aist would be useful in low-level real-time control of robot hardware. Currently a robot program that requires real-time must drop out of ROS to gain it.

We have implemented a new transport for OpenRTM-aist that allows RTCs to speak ROS's language. The transport can be used both to communicate with ROS components and to communicate with other RT Components using the channel-based communications.

As mentioned earlier, OpenRTM-aist uses ports to provide access to its transports. The ROS transport provides a set of new port types: channel publication, channel subscription, service provision and service utilisation.

The output and input ports provide publisher and subscriber access, respectively, to ROS channels. The request and service ports provide access to and provision of ROS services.

These ports can be used directly in a component's code. They are written to and read from using the same API as that used for the existing data ports, in the case of the input and output ports. Due to fundamental differences in the way the OpenRTM-aist and ROS service ports operate, the service ports in the ROS transport use an API similar to that used by ROS components.

As shown in Figure 5, the ROS transport in OpenRTM-aist fits into the model shown in Figure 3.

The transport itself is implemented directly in OpenRTM-aist, making it a native transport. Unlike a wrapper, which requires that the client code of the component manage the transport, all transport management is handled within

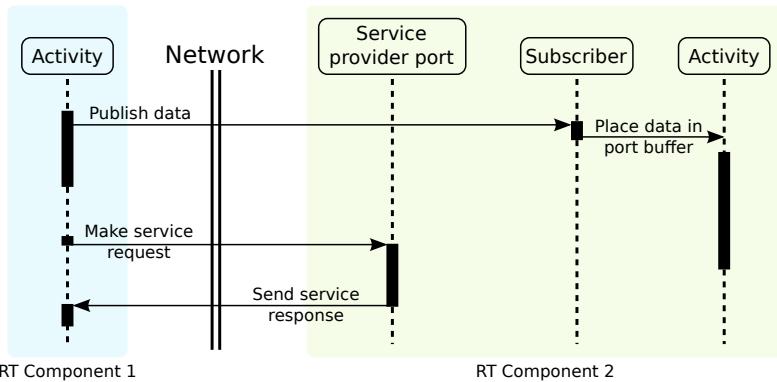


Fig. 4. The communication between processes involved in a one-way data flow and a two-way request for the ROS transport port

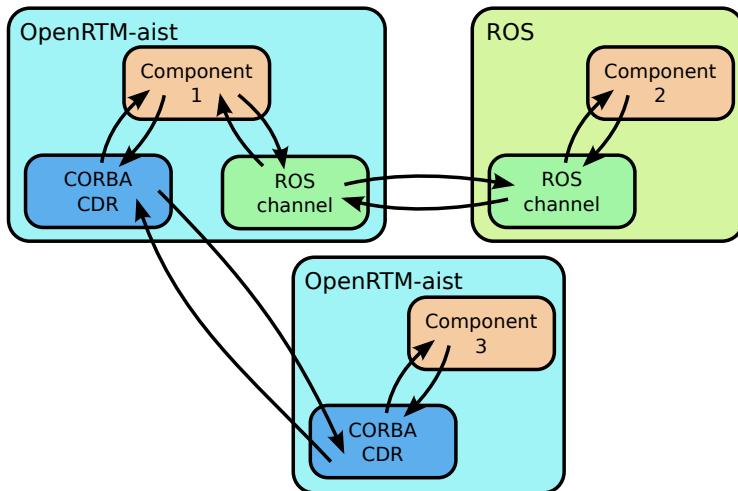


Fig. 5. The ROS transport provided by OpenRTM-aist allows a single component to communicate freely with both other OpenRTM-aist components, using the native OpenRTM-aist CORBA-based transport, and ROS components, using ROS's channel-based transport. The ROS transport can even be used between OpenRTM-aist components, if so desired.

the architecture. In OpenRTM-aist, this means utilising parallel threads of control (illustrated in Figure 4).

For each port that can receive data or requests, a parallel thread is started that manages the port. These are not complex tasks; typically it consists of providing execution time to the internal ROS systems that manage the transport. The details for each port type are given below. The actual implementation of

the transport is left to the ROS libraries; OpenRTM-aist merely utilises those libraries. This means that any improvements made to ROS will indirectly benefit OpenRTM-aist as well.

Publishing Port. The publishing report is very simple. It merely places the given piece of data into the ROS channel via the standard ROS publication functions. The ROS library takes care of serialisation and transportation from that point.

Subscribing Port. As with the original OpenRTM-aist input port, a separate variable provides access to the most recent piece of data received. The ROS subscribing port uses a callback function, as is standard with ROS subscribers. The callback is provided by the port; the user does not need to provide it themselves. The responsibility of the callback is to copy the most-recently-received data into a buffer. When the port is “read” by the component, the next value in this buffer is copied into the target variable. A separate thread is used to give execution time to `ros::spin()`, the ROS function responsible for calling callbacks when data arrives. This allows data reception to be handled out-of-band from the component execution.

Service Port. ROS services are callback-based. Each service in a ROS component exposes a single callback. The ROS transport for OpenRTM-aist works in the same way. The developer provides a callback functor object. The port uses a parallel thread to execute the ROS functions that handle receiving service requests, and ensures that the callback functor is called.

Request Port. The request port does not use a parallel thread. As with the native OpenRTM-aist request port, requests are blocking. The implementation is very simple. It calls the standard ROS function for calling a service on the ROS transport object contained within the port, blocking until a reply is received.

4 Discussion

The use of a native transport to interact with another framework brings many benefits over writing a component wrapper:

- A new wrapper must be written for each message type to be exported from the secondary framework to the primary framework. The ROS transport avoids this by allowing components interested in data from the secondary framework to access it directly. This is a more natural approach for developers.
- A wrapper is often only effective in one direction – both the ORCA2 and MARIE wrappers are effective at bringing behaviour from the secondary frameworks to the primary, but are difficult at best to use in the opposite direction. This transport allows the benefits to pass both ways. For example, ROS gains access to a framework it can use for hard real-time parts of the robot, while OpenRTM-aist gains a persistent channel-based transport.

- A wrapper only allows us to gain the behaviour contained in components. The transport approach means that we also gain the benefits of the secondary framework’s transport itself. We are not limited to using this transport to talk to the secondary framework. We can also use it between components of the primary framework as an alternative to the existing transports.
- The transport is as efficient as working directly in the secondary framework. No translation of data or reading-and-re-sending is required.

There are specific benefits both to OpenRTM-aist and to ROS of this particular transport.

- OpenRTM-aist gains a persistent channel-based transport. This has a variety of applications in robotics.
- OpenRTM-aist gains access to the huge library of functional software provided by and for ROS.
- ROS gains access to components that can execute in hard real-time.

An interesting application of native transports for framework interaction is in creating layered systems. A typical layered architecture features interaction points between the layers of some kind. With native transports, it is possible to utilise one architecture per layer, using the best architecture for the job. For example, the low-level control layer could be implemented in OpenRTM-aist, with its strong life-cycle control and hard real-time support, while a higher level for planning could utilise ROS, with its greater degree of flexibility and large library of perception and planning components. It is the author’s opinion that this method of implementing layered systems has promise in bringing greater flexibility to how such systems are implemented.

One thought that may occur is that it is tempting to design one component to act as the gateway between the two frameworks, repeating the situation we have with wrappers. This may be the case when the data types between the two frameworks do not match. Figure 3 certainly seems to imply this approach. However, it is important to remember that the transport can be used in any component, and so the more likely scenario is that in which the appropriate transport is used on a connection-by-connection basis. The ease of using either transport facilitates this.

The specific implementation presented in this paper is not perfect. It has two flaws that limit its usefulness – the lack of commonality in data type representation and incompatible introspection interfaces.

OpenRTM-aist and ROS use different and incompatible serialisation schemes in their transports. OpenRTM-aist uses CORBA CDR, with the OMG IDL language for data description, while ROS uses a home-grown scheme based on generated serialisation functions and its own IDL for data description. This means that any data types that must be sent over both transports must be specified twice, once in each IDL.

OpenRTM-aist and ROS also use different interfaces for introspection. While a ROS component network is relatively static once running, in OpenRTM-aist an introspection tool is necessary to set up the component network at run-time.

Because the OpenRTM-aist introspection tools are not aware of ROS nodes or channels, it is not possible to use them to directly connect to ROS channels at run-time. Instead, ROS channel connections must be fixed prior to run-time. This is less than optimal, from the point of view of OpenRTM-aist.

This integrated approach to inter-operation is rapidly spreading. The YARP framework [6] has recently gained experimental ROS protocol support, although it is incomplete at the present time. Differing component models do not prevent it being used; all that matters is that the data is available at each end of a connection when necessary. The local framework is responsible for providing it to components with the correct semantics after transport is complete.

4.1 Attaining the Ideal

The two disadvantages mentioned in the previous section are not insurmountable. Creating the ideal situation requires flexible serialisation systems and common introspection interfaces.

Flexible serialisation is necessary to allow a transport to choose its serialisation method. For example, if OpenRTM-aist could choose its serialisation scheme, it would be able to transmit data types specified in ROS IDL or CORBA IDL. This would remove the need to specify data types twice. ROS and OROCOS 2 [3] both feature flexible type systems that can handle any serialisation method the developer chooses to use, while OpenRTM-aist is limited to just one. This is a problem with OpenRTM-aist.

Standard introspection systems is a more wide-spread issue. OpenRTM-aist, ROS and the aforementioned OROCOS 2 all feature introspection. They also all implement it differently, with different capabilities. If the introspection of transports were at least common, it would be easier to create a tool that could manage the connections between components from multiple systems.

5 Conclusions

Previously, creating a wrapper was the most common approach to allowing interaction between robot software frameworks for use in a single robot system. This approach has several limitations. It requires considerable developer effort to create and maintain a wrapper, it introduces inefficiencies into the system, and multiple wrappers are typically required to adapt all the desired messages of the secondary framework to the primary framework.

A better approach is to separate the transport from the framework, and allow multiple transports to be used. This can then be exploited by utilising transports from other frameworks, and so allow a framework to speak the language of other frameworks and communicate with them. This approach is more flexible than wrappers, requires less effort, and is more efficient.

We have developed such a transport for OpenRTM-aist. It provides native communication with the ROS framework, allowing components for either framework to communicate seamlessly with components for the other. The new transport

brings the benefits of ROS's channel-based communication to OpenRTM-aist, and OpenRTM-aist's hard real-time support to ROS. The two frameworks can be utilised in the appropriate parts of a single robot system.

Acknowledgements

This research was partially supported by the Grant-in-Aid for the Developing Intelligent Technology for Next-Generation Robots Project (FY2007-2011) of the New Energy and Industrial Technology Development Organization of Japan.

References

1. Ando, N., Suehiro, T., Kotoku, T.: A software platform for component based rt-system development: Openrtm-aist. In: Carpin, S., Noda, I., Pagello, E., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2008. LNCS (LNAI), vol. 5325, pp. 87–98. Springer, Heidelberg (2008)
2. Biggs, G., Makarenko, A., Brooks, A., Kaupp, T., Moser, M.: GearBox: Truly reusable robot software (Poster). In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2008 (September 2008)
3. Bruyninckx, H.: Open robot control software: the OROCOS project. In: Proceedings of IEEE International Conference on Robotics and Automation, ICRA 2001, vol. 3, pp. 2523–2528 (2001)
4. Collett, T., MacDonald, B., Gerkey, B.: Player 2.0: Toward a practical robot programming framework. In: Proceedings of the Australasian Conference on Robotics and Automation. University of New South Wales, Sydney (December 5–7, 2005)
5. Cote, C., Letourneau, D., Michaud, F., Valin, J.M., Brosseau, Y., Raievsky, C., Lemay, M., Tran, V.: Code reusability tools for programming mobile robots. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2004, vol. 2, pp. 1820–1825 (2004)
6. Fitzpatrick, P., Metta, G., Natale, L.: Towards long-lived robot genes. *Robotics and Autonomous Systems* 56(1), 29–45 (2008), <http://www.sciencedirect.com/science/article/B6V16-4PT296V-1/2/4009fd97b8597d84d20457d2fc7d9db0>
7. ROS Wiki (2010), <http://www.ros.org>
8. The Robotic Technology Component Specification - Final Adopted Specification (2010), http://www.omg.org/technology/documents/spec_catalog.htm
9. RTSystemEditor (2010), <http://www.openrtm.org/OpenRTM-aist/html-en/Documents2FRTSystemEditor.html>

Run-Time Management of Component-Based Robot Software from a Command Line

Geoffrey Biggs, Noriaki Ando, and Tetsuo Kotoku

Intelligent Systems Research Institute

National Institute of Advanced Industrial Science and Technology (AIST)
AIST Tsukuba Central 2, Tsukuba, Ibaraki 305-8568, Japan

Abstract. Component-based software is a major recent design trend in robotics. It brings many benefits to system design, implementation and maintenance. The management of such systems often depends on graphical tools. These tools are powerful and provide a rapid way to layout component networks. However, they also typically require considerable resources to run. This is not ideal in robotics, where low-resource environments are common. We have created a set of command-line tools for use with the OpenRTM-aist component-based robot middleware. The tools follow the UNIX philosophy of simplicity and aggregation. These tools allow whole component-based systems to be created, managed and monitored from a command-line. They are ideal for use in environments where a graphical interface is not available. By combining tools together, more complex functionality can be easily created.

1 Introduction

Component-based software design and implementation is a current trend in software engineering. Software is divided into individual components, each with a well-defined interface that specifies what functionality that component provides. Multiple software components are combined together into a complete software system in much the same way as the well-defined hardware components of electrical circuits are combined to create a complete hardware system [12].

Component-based practices bring many benefits to software design, implementation, maintenance and reuse, including known interfaces that act as “contracts” between components, “separation of concerns” (each component only deals with its individual problem), isolation testing, and rapid development of new systems using existing commoditised software resources.

These benefits also apply to the design, implementation, maintenance and reuse of robot software. For example, the componentisation of hardware drivers and algorithms allows robot systems to be built from pre-existing, ideally off-the-shelf, software components. As a result, component-based software is a major trend in robotics, particularly service robotics. Recent examples include OpenRTM-aist [1], ORCA [2], ROS [8] and OPRoS [11].

A key part of using component-based software is interacting with the component network that makes up the software system, both when designing the

system and when monitoring and maintaining the system in its running state. There are a variety of methods available for this. Interaction may be via a specialised tool designed to monitor a specific network of components. Alternatively, a generic tool for the component architecture upon which the software system is built may be used. In some cases, no such tool may be available, with all interaction taking place through configuration and log files.

The work in this paper presents a set of tools designed for managing systems using the OpenRTM-aist component-based framework. These tools differ from the usual approach in that they apply a file-system abstraction to the components for interaction, they are command-line tools and they follow the UNIX philosophy of being small and using aggregation to add power. Such a collection of simple tools gives great flexibility and ease of use to developers using component-based robotics software, particularly when the interaction method may be constrained by unique environmental factors.

The following section describes the framework that is the focus of this work. Section 3 discusses the background for the work. Section 4 discusses the tools themselves. Discussions and conclusions are given in Sections 5 and 6.

2 OpenRTM-aist

OpenRTM-aist [1] is a component-based architecture for intelligent systems, known in Japan as “Robot Technology,” or “RT.” OpenRTM-aist implements the Robot Technology Component (RTC) specification [9] from the Object Management Group (OMG), which defines a common introspection interface. Introspection is an important part of OpenRTM-aist, providing the basis upon which tools build to interact with and manage systems using the architecture.

The central concept in OpenRTM-aist is the RT Component, or RTC. Each RTC has an internal state machine with known states. The component’s internal health can be both monitored and controlled through the introspection interface. In order for a component to begin executing, it must be activated, placing its state machine in the *Active* state. Execution can be terminated by deactivating the component, returning its state machine to the *Inactive* state. If an error occurs in the component, it moves to the *Error* state, from where it must be reset to move it back to the *Inactive* state.

OpenRTM-aist supports data-flow- and request-based communications, with interaction between components occurring at “ports.” The component network is formed by making connections between the ports of the components. CORBA is used as the transport.

OpenRTM-aist can use manager daemons to load and manage components dynamically on remote nodes.

OpenRTM-aist uses CORBA [6] to implement the introspection interface. Components must register on a known name server (there can be more than one known name server in use). Typically, components are organised on the name server using naming contexts below the root context in order to provide hierarchical categorisation.

OpenRTM-aist is part of a larger project by Japan's New Energy and Industrial Technology Development Organisation (NEDO) for creating the elemental technologies of advanced robotics. Alongside OpenRTM-aist, there are also several tools for use with the architecture under development. These include RT-SystemEditor, the main tool for creating and interacting with RT Systems, the component networks created using OpenRTM-aist.

3 Interacting with Component-Based Software Systems

There are many approaches to constructing component-based systems from individual components. One is a fixed design of components with the connections between them hard-coded into the source code. Another is the use of configuration files specifying the network of connections between components. This method is used in, for example, ROS, where an XML-format file is used to describe the component network.

Graphical tools may be provided for working with the component-based software architecture. These tools are designed to simplify the development process to a drag-and-drop level, where new software systems can be constructed from existing software components by dragging them into a system diagram and

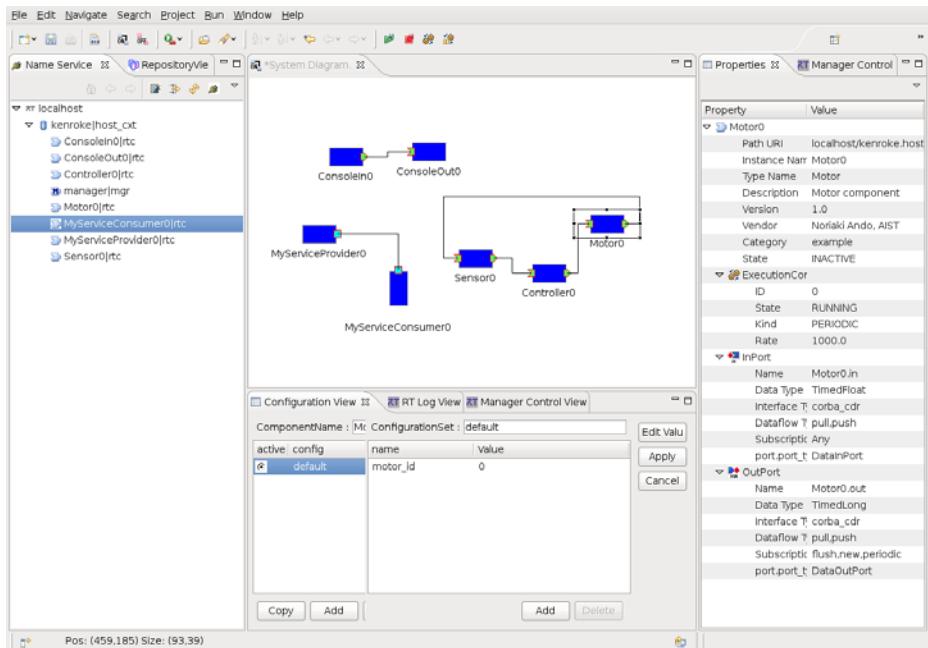


Fig. 1. The RTSystemEditor tool used with OpenRTM-aist to construct component networks

drawing connections between them. An example of such a tool for a robot-oriented component architecture is RTSYSTEMEDITOR, shown in Figure 11.

Graphical tools work well for system developers. With RTSYSTEMEDITOR, developers are able to easily experiment with new system layouts through drag-and-drop interaction. A new system is created by dragging the desired components from the naming service view, which shows known components, and then drawing connections between their ports. An entirely new, complete system, featuring two components and a single connection, can be created in seconds. It is this speed benefit that has led to so much research in graphical tools over the years, and RTSYSTEMEDITOR is not alone in providing this functionality. Other examples include EasyLab [5] and SmartSoft [10].

Unlike hard-coding and text-based configuration files, graphical tools may be useful beyond the creation step. RTSYSTEMEDITOR, for example, can be used to manage a running RT System. Individual and groups of components can be started, stopped and reset, configuration parameters of components can be altered, links created and destroyed, and the health of each component monitored. This is possible because of the extensive introspection interface offered by OpenRTM-aist. The benefit to developers is the ability to modify the running RT System and experiment with different component network layouts in actual time. A developer can even use RTSYSTEMEDITOR to monitor the state of the robot's software while it's running.

Unfortunately, the use of a graphical tool brings with it certain requirements that cannot always be met. GUIs introduce more difficulty than benefit in some situations commonly found in robotics:

- In low-resource environments, such as a robot's internal computer, the use of the graphical environment necessary to display the graphical tool, as well as the tool itself, can use up valuable resources. A robot's internal computer already has enough processing to do dealing with sensor data and planning. It often cannot handle the load of a graphical tool as well without compromising the capability of the robot.
- In order to get around the resource limitations, running the graphical tool from a remote computer may be possible. However, not all robot computers feature a network connection, making this method of connection impossible in those situations.
- Graphical tools cannot easily be scripted to perform repetitive tasks. Methods for scripting GUIs do exist, such as graphical tools with built in scripting systems and languages for manipulating GUIs [13]. However, these methods may not offer the flexibility that can be found in, for example, the command-line shell and tools available on the average UNIX system.

Inspired by the difficulty of using RTSYSTEMEDITOR on our outdoor robots, where computing resources are tight and there is no network connectivity to a remote computer, as well as the massive flexibility of command-line tools on UNIX-based operating systems, we have experimented with alternative methods of creating and managing an OpenRTM-aist component network that do not rely on a

graphical interface. We aim to create a method that allows for great flexibility in, as well as automation of, the management of the component network.

We must consider two aspects of the tool design. The first is how to reference the objects of the system. A graphical environment allows selection from a display; we must provide an approach that works in a text-based environment. The second is the method of control: pre-scripted with a programming language, or a more interactive approach, similar to what a graphical tool provides.

The following two sections describe the approach taken to each of these two issues.

3.1 The Pseudo-file System

The RTSYSTEMEDITOR tool presents known RTCs and other objects of the system in a tree structure. The top level of this tree is the root, and below that are known name servers, on which OpenRTM-aist objects register. Below the name servers are the objects themselves. They are typically sorted into naming contexts, which function as directories on the name servers.

The file system metaphor, commonly stated as “everything is a file,” is a fundamental part of the UNIX philosophy. It says that everything on the system can be treated as a file. Inspired by this, we have used it to represent the same tree structure displayed in RTSYSTEMEDITOR. The tools described in this paper present the user with a virtual file system. Files in the file system represent OpenRTM-aist objects such as RTCs and managers. Directories represent name servers and naming contexts. The user can address a component by providing its absolute path in the virtual file system, or a relative path from their current working directory.

3.2 Interacting with Components

We could use a programming language to interact with the system. Creating a software library to interact with the components would support this. We would be able to program various interactions with the system objects. However, such a library already exists in the form of the architecture itself. It provides an API, defined in CORBA IDL, used to introspect the objects. Similar libraries exist for most middleware systems. The omniORB CORBA implementation [7], for example, has an extensive API that can be used to interact with CORBA objects at a low level. omniORB’s API even already allows programmers to address objects registered on naming services using the path addressing scheme discussed in the previous section.

If such a library is already available, why doesn’t the programming approach meet our needs? It doesn’t because we desire the same level of interaction granted by graphical tools. Having to write a new program for everything we wish to do with the system is both inflexible and infeasible - developers do not have that much time.

In order to achieve the flexibility desired, we need an interactive approach. Drawing further inspiration from UNIX, we have created two sets of command-line

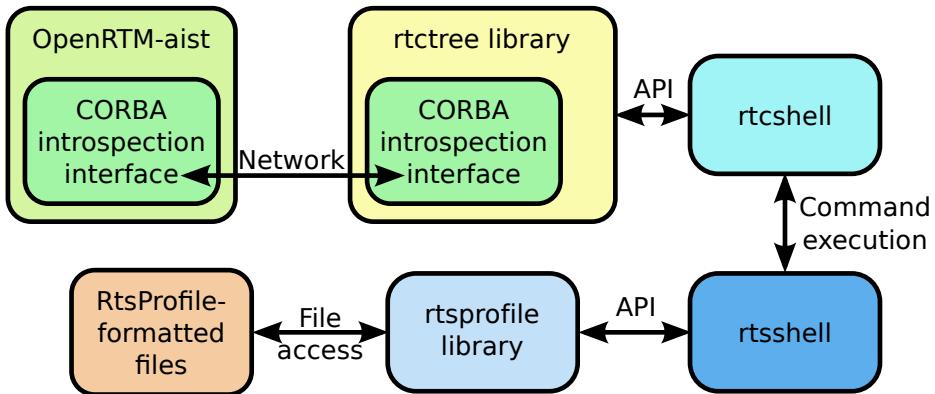


Fig. 2. The interaction between OpenRTM-aist and the various tools

tools for manipulating the objects of an RT System. We have named these “*rtcshell*,” for managing individual objects one at a time, and “*rtsshell*,” for managing entire systems at once.

4 Shell Utilities for OpenRTM-aist

The tools in both tool kits follow the UNIX philosophy of being small, doing one thing, doing it well, and using aggregation to add power. Each individual tool only performs one task, such as checking the life-cycle state of a component or making a connection, but they can be combined together and with existing UNIX commands to create more powerful tools.

The tool kits build on two libraries that were created to provide most of their functionality:

rtctree. Implements the virtual file system and is responsible for all interaction with the OpenRTM-aist introspection interfaces.

rtsprofile. Provides reading and writing of files in the RtsProfile format, an XML and YAML specification for describing RT Systems.

The interaction between the tool kits, the libraries and OpenRTM-aist is shown in Figure 2. All libraries and tools are implemented in Python.

4.1 rtctree

The rtctree library implements the virtual file system, which provides information about and addressing of all known OpenRTM-aist objects. The file system’s structure is illustrated in Figure 3. All paths branch off from a single root node, /. Below this root are all the known name servers. Name servers may have

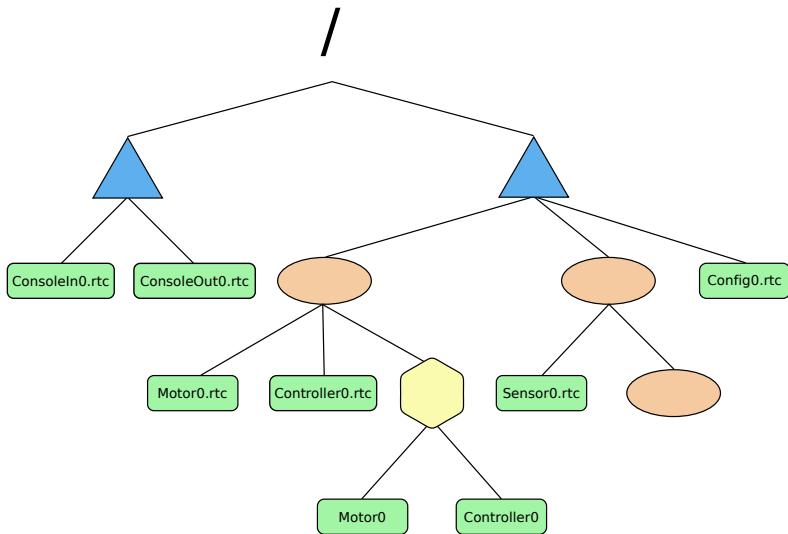


Fig. 3. The general structure of the pseudo-file system used by the *rtcshell* utilities, dubbed the *RTC Tree*. The root node is at the top of the file system tree. The blue triangles are naming service nodes. Red ovals are naming contexts below a root context. Green boxes are RT Components. The yellow hexagon is a manager. Below it are aliases to the two RT Components it is managing.

multiple naming contexts to which objects can register for organisational purposes. Within the file system, these are treated as directories.

The files of the virtual file system are RTCs and managers. The tools in *rtcshell* operate on these files. The files “contain” information about the object they represent, such as state and available ports for RTCs. This information is accessible using the various tools from *rtcshell*.

Manager objects are used to manage running component instances, so they contain references to the components they are managing. To make these easy to interact with, the virtual file system treats managers as directories containing aliases to the RTCs.

Using a tree structure allows additional tree-structure operations to be performed, such as iterating over the tree to perform an operation on every node matching certain criteria.

4.2 *rtcshell*

The *rtcshell* tool kit contains a set of shell commands for interacting with individual RTCs and managers. It uses the *rtctree* library to access the introspection interfaces of OpenRTM-aist objects. Each command in the tool kit has one specific function. These can be roughly divided into categories: navigating the virtual file system, viewing “file” contents, and manipulating files.

Navigating the virtual file system. Navigation around the virtual file system is essential for making the tools usable. We cannot expect the user to supply a full path to every command.

The virtual file system is not a part of the real file system¹. rtcshell therefore needs to provide its own commands for changing the virtual working directory. It provides equivalent to standard commands such as `cd` and `ls`. These are listed below.

rtcwd. Changes the working directory to the given path, which may be absolute or relative to the current working directory.

rtpwd. Prints the current working directory.

rtls. Lists the contents of a given path or the current working directory.

rtfind. Searches the virtual file system for objects and directories matching given search criteria. This command is implemented using the iteration function provided by `rtctree`.

The current working directory is stored in an environment variable, `RTCSH_CWD`. This variable is manipulated by `rtcwd` and used by the other commands of `rtcshell` when using paths.

Examples are shown in Listing 1.1. Lines 1 to 7 show using `rtls`. Like the standard `ls` command, `rtls` has both short and long forms. The long form briefly displays some useful information about components, such as their current state and the number of connections present. It is useful for monitoring the overall state of running components.

Listing 1.1. Examples of using the `rtcshell` tools to operate on the virtual system. Some lines have been removed for brevity.

```

1 $ ./rtls
2 Clusterer0.rtc      Hokuyo_AIST0.rtc
3   kenroke.host_ctxt/
4 $ ./rtls -l
5   Inactive  2/0  1/0  1/0  0/0  Clusterer0.rtc
6   Inactive  4/0  0/0  3/0  1/0  Hokuyo_AIST0.rtc
7   -         -    -    -    -    kenroke.host_ctxt
8 $ rtcwd kenroke.host_ctxt/
9   $ rtcat ConsoleIn0.rtc
10  ConsoleIn0.rtc  Inactive
11    Category      example
12    Description   Console input component
13    Instance name ConsoleIn0
14    Parent
15    Type name     ConsoleIn
16    Vendor        Noriaki Ando, AIST
17    Version       1.0

```

¹ We are investigating methods for making it a part of the real file system, similar to the `/sys` virtual file system in Linux.

```

18 +Execution Context 0
+DataOutPort: out
20 $ rtconf ConfigSample0.rtc set default int_param0 5
21 $ rtconf ConfigSample0.rtc set int_param1 3
22 $ rtconf ConfigSample0.rtc list -l
-default*
24   double_param0  0.11
   double_param1  9.9
26   int_param0    5
   int_param1    3
28   str_param0    foo
   str_param1    bar
30   vector_param0 0.0,1.0,2.0,3.0,4.0
31 $ rtact SequenceInComponent0.rtc
32 $ rtls -l
  Inactive  8/0  0/0  8/0  0/0  SequenceOutComponent0.rtc
34 Active    8/1  8/1  0/0  0/0  SequenceInComponent0.rtc
 [...]
36 $ rtcon ConsoleIn0.rtc:out ConsoleOut0.rtc:in
37 $ rtcat ConsoleIn0.rtc -l
38 ConsoleIn0.rtc  Inactive
 [...]
40 -DataOutPort: out
  dataport.data_type      TimedLong
42  dataport.dataflow_type push
  dataport.interface_type corba_cdr
44  dataport.subscription_type flush,new,periodic
  port.port_type          DataOutPort
46 +Connected to
  /localhost/ConsoleOut0.rtc:in

```

Viewing “file” information. The files of the virtual file system “contain” information from the objects they represent, such as the state of an RTC or the list of loaded modules for a manager. rtcshell provides several tools for viewing this information.

rtcat. This is the most basic command for viewing contents. It prints out the information provided by an object’s introspection interface, showing information such as an RTC’s current state, its available ports, and the connections provided on those ports. An example is shown in Listing 12, lines 9 to 13.

rtconf. Displays the configuration parameters of RTCs. Each RTC can contain various parameters used by its internal behaviour. See Listing 11, lines 20 to 30 for an example.

rtprint. This command displays the data being sent over a port. It is useful for checking if an RTC is sending the information the user expects. It makes use of Python’s ability to convert nearly any object to a string.

Manipulating files. The most important function of rtcshell is manipulating the objects of the virtual file system, particularly for connecting ports and changing the state of RTCs. The following commands are used for this.

rtact. Activates a component, causing it to begin executing. See Listing 11, lines 31 to 35 for an example.

rtdeact. Deactivates a component, halting its execution.

rtpreset. Resets a component, moving it back to the *Inactive* state after an error has occurred.

rtcon. Connects two ports together. Ports of a component are specified after a colon at the end of a component's path. See Listing 11, lines 36 to 47.

rtdis. Removes connections.

rtinject. Injects data into a port. This is useful to test a component's response to inputs without needing to implement another component to create test data.

rtmgr. Manipulates manager objects. Managers are daemons used to deploy components. **rtmgr** can be used to control the deployment of RTCs loaded from shared libraries.

4.3 rtsshell

rtcshell is only used for manipulating individual file system objects. It does not provide any facilities for easily manipulating entire RT Systems with a single command. rtsshell is a complimentary tool kit that provides this functionality. It works with RtsProfile files using the rtsprofile library to describe complete systems, and uses rtcshell to manipulate the individual RTCs and managers.

rtcryo. Creates a new RtsProfile file from the currently-running RT System.

rtteardown. Removes all connections in an RT System.

rtresurrect. Uses an RtsProfile file to recreate an RT System from running RTCs by restoring connections and configuration parameters.

rtstart. Starts an RT System by shifting all RTCs to the *Active* state.

rtstop. Stops an RT System by shifting all RTCs to the *Inactive* state.

All these commands are capable of checking if the necessary objects are available in the virtual file system. They can also use a partially-ordered planner to ensure that state change commands are executed in the correct order, as specified by the developer when creating the RT System.

5 Discussion

Each tool performs one task only. However, while each tool is simplistic by itself, the collection as a whole is both flexible and powerful. As is common for UNIX tools, aggregation of the tools provides additional power.

For example, combining the standard UNIX `watch` command and `rtls -l` gives a continuously-updating display of component state. A list of all output ports of

a component can be obtained by combining `rtcat` and `grep`. The shell script `for` command can be combined with `rtfind` and `rtact` to activate all components matching a given name specification. The implementation of the rtsshell tools relies on aggregating the rtcshell tools to perform their actions, such as starting all components in the system using `rtact`. We do note that this usage style leads to the RTC Tree being constructed over and over, placing additional load on the introspection interfaces of components. This can be mitigated by careful management of the rate at which tools are executed.

These tools are different from RTSYSTEMEDITOR. Many of the capabilities are the same, but each has its own advantages. rtcshell is not useful for visualising large component networks, something RTSYSTEMEDITOR's graphical interface excels at. RTSYSTEMEDITOR is very difficult to automate, while it is trivial to automate the command-line tools using shell scripting.

There is little directly-comparable work in robotics. Some other architectures have tools with some related capabilities. For example, the Task Browser component from Orocó [3] allows navigation around a running network of components, viewing the ports, properties and so on. This is more like an interactive component for debugging other components than a tool for creating and managing component-based systems.

ROS includes introspection of its communications channels, and provides shell tools for viewing information about them. One such tool is `rostopic`, which can show various information about a topic such as the data being transmitted over it. However, these tools can only monitor and send data to topics. They cannot manipulate the component network, such as creating new connections, nor can they find any information about components themselves.

YARP [4] contains command line tools for tasks such as connecting ports together and querying port locations (similar to the `rtcon` tool), through a central server.

The strength of rtcshell and rtsshell is in quickly creating small systems for experimentation, for managing both large and small RT Systems, and for automation of common tasks. No other system currently matches all of their functionality.

6 Conclusions

This paper has described a set of command-line tools for managing RT Components and component networks for the OpenRTM-aist architecture. The tools treat known components and other OpenRTM-aist objects as part of a file system. They allow the user to easily inspect and manage components in a console.

This form of interaction is well suited to the low-resource environments that are commonly found in robotics, where a resource-intensive graphical tool is not feasible. They allow greater freedom for developers. The tools can be scripted using standard shell scripting facilities, facilitating the automation of tasks.

We believe that such a set of command-line tools adds additional usability to component-based software architectures. The UNIX philosophy of tools being

small with power through aggregation has been proved over the years to lead to a highly-flexible system. Its application to robotics is important if robots are to be likewise flexible, maintainable and easy to develop.

Acknowledgements

This research was partially supported by the Grant-in-Aid for the Developing Intelligent Technology for Next-Generation Robots Project (FY2007-2011) of the New Energy and Industrial Technology Development Organization of Japan.

References

1. Ando, N., Suehiro, T., Kitagaki, K., Kotoku, T., Yoon, W.K.: RT-middleware: distributed component middleware for RT (robot technology). In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2005, pp. 3933–3938 (August 2005)
2. Brooks, A., Kaupp, T., Makarenko, A., Williams, S., Oreback, A.: Towards component-based robotics. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2005, pp. 163–168 (2005)
3. Bruyninckx, H.: Open robot control software: the orocos project. In: Proceedings of IEEE International Conference on Robotics and Automation, ICRA 2001, vol. 3, pp. 2523–2528 (2001)
4. Fitzpatrick, P., Metta, G., Natale, L.: Towards long-lived robot genes. *Robotics and Autonomous Systems* 56(1), 29–45 (2008)
5. Geisinger, M., Barner, S., Wojtczyk, M., Knoll, A.: A software architecture for model-based programming of robot systems. In: Kröger, T., Wahl, F.M. (eds.) *Advances in Robotics Research - Theory, Implementation, Application*, Braunschweig, Germany, pp. 135–146. Springer, Heidelberg (2009)
6. Henning, M., Vinoski, S.: Advanced CORBA Programming with C++. Addison-Wesley Professional, Reading (1999)
7. omniORB (2010), <http://omniorb.sourceforge.net/>
8. ROS Wiki (2010), <http://www.ros.org>
9. The Robotic Technology Component Specification - Final Adopted Specification (2010), http://www.omg.org/technology/documents/spec_catalog.htm
10. Schlegel, C., Hassler, T., Lotz, A., Steck, A.: Robotic software systems: From code-driven to model-driven designs. In: International Conference on Advanced Robotics, ICAR 2009, pp. 1–8 (June 2009)
11. Song, B., Jung, S., Jang, C., Kim, S.: An Introduction to Robot Component Model for OPRoS (Open Platform for Robotic Services). In: Workshop Proceedings of Intl. Conf. on Simulation, Modeling and Programming for Autonomous Robots 2008, pp. 592–603 (November 2008)
12. Szyperski, C., Gruntz, D., Murer, S.: Component Software – Beyond Object-Oriented Programming, 2nd edn. Addison-Wesley/ACM Press (2002)
13. Visual Basic Developer Center (2010), <http://msdn.microsoft.com/vbasic/>

Modelling Behaviour Requirements for Automatic Interpretation, Simulation and Deployment

David Billington, Vladimir Estivill-Castro, René Hexel, and Andrew Rock

Griffith University, Nathan 4111, QLD, Australia
<http://griffith.edu.au/mipal>

Abstract. In this paper we propose a high level approach to capture the behaviour of an autonomous robotic or embedded system. Using requirements engineering, we construct models of the behaviour where system activities are captured mainly by collaborating state machines while the domain knowledge is captured by a non-monotonic logic. We explain our infrastructure that enables interpretation, simulation, automatic deployment, and testing of the models, minimising the need for developers to code. The approach also minimises faults introduced in the software development cycle and ensures a large part of the software is independent of the particular robotic platform.

Keywords: requirements engineering, interpretation of models, knowledge representation.

1 Introduction

In many robotic platforms and embedded systems, the production of software is complicated by the fact that creating correct code for the final deployment environment is costly. This provides a strong incentive for simulation, as the software can be verified and/or tested and brought to required levels of quality without the significant costs and risks of using the actual platform.

We propose here that the production of software for robots should be taken one step further and follow the software engineering postulations of requirements engineering and behaviour engineering. Namely, if we capture the required behaviour into unambiguous and descriptive models, then faults are not introduced into the software in later phases of the software development cycle. Moreover, if verified models are directly translated or interpreted in simulation environments and/or the corresponding target platform, software faults will be non-existent. The simulation will enable direct validation, and the modelling will enable traceability all the way from requirements to coded behaviours. This not only improves the design of behaviours and the compliance with requirements, but it also produces far more maintainable software. This way, software models also become instruments to document the implementation and to assist in the usage or training of people interacting with such autonomous systems.

Finite state machines (and their variants) are ubiquitous in modelling the behaviour of single threaded embedded systems. They appear in one form or another to capture the states that computer-based systems incarnate and the transitions they perform. Finite state machines have been used in many software engineering methodologies, tools, and developing approaches and continue to be core to widely supported software modelling standards such as UML. Moreover, the semantics of the model can be made unambiguous, they can be represented diagrammatically, and they can also be simulated and interpreted in accordance with their semantics. We propose here to use this approach to represent the activities, transitions and states of the different modules of the software for a robot. However, we enhance the model by allowing a richer language for the labelling of transitions. We propose a non-monotonic logic that enables gradual capturing of the requirements and design of the behaviour with smooth, iterative refinement. This high-level descriptive tool allows a declarative description of domain knowledge. Moreover, the reactive nature of state-machines is complemented by the reasoning nature of the inference engine of the logic. Therefore, our modelling enables the diversity of architectures of multi-agent systems.

However, a robotic system usually demands the simultaneous control and co-ordination of several modules. We propose to enable several finite state-machine processes or threads and to model their message passing using Behavior Trees [23]. Behavior Trees have been proven to be fruitful in many complex applications of embedded system as a tool for requirements engineering.

Our models can not only be simulated, they can also be executed on board the robots and be migrated across platforms. We use a publish/subscribe architecture for what we call a *whiteboard* [3] to facilitate the communication between the multi-threaded machines and also to isolate generic software components from particular hardware platforms. Because we can enable priorities in the whiteboard, we can also adopt subsumption architectures [7]. Moreover, it is possible to simulate none, a few, or all of the modules and execute none, some, or all of the rest aboard the robot. Our whiteboard can operate on a single system as well as across a computer network. Because of this distribution capability, it is possible to execute several modules of a system for simulation and even execute some of the others on one or several robots. We can create simulations in which not all sensors and actuators are on the same robot. A situation where this may come handy is when we want to, for example, test a cruise control on a remote vehicle or robot, while using a brake pedal that is locally placed in a laboratory.

The aim of this paper is to reverse the belief among some authors that automatic code generation and simulation with state diagrams is impossible.

“Many people understand automatic code synthesis of a program to solve a problem from a statement of the problem specification. State chart-based tools cannot provide this because a state chart is just a higher-level (mostly visual) solution rather than a statement of the problem.” [18]

By using state diagrams in conjunction with an inference engine (for reasoning about transitions) for requirements engineering, we can ensure the correctness of the fully automatic, deployed system derived directly from the state diagrams.

2 Non-Monotonic Logic in Finite State Machines

Our approach uses one fundamental modelling technique, namely finite state machines (FSMs)¹. FSMs are standard models of automata. In the early 90s, state machines with a single thread of control became the instrument of choice to model the behaviour of objects. Rumbaugh's Object-Modeling Methodology (OMT) [17, Chapter 5] established state diagrams as the primary dynamic model. The so-called Shlaer-Mellor approach established state models to capture the life cycle of objects of a given class [19]. They were also the instrument of choice to model reactive systems [11]. Today, besides being a part of UML, FSMs are used with Web technologies and can be defined in XML (the W3C has a working draft specification "State Chart XML (SCXML): State Machine Notation for Control Abstraction," specifying semantics and reference implementations). Parallel FSMs and Petri nets enable the modelling of multi-threaded architectures including client-server architectures. Both formalisms are usually represented as diagrams when modelling software systems. We will add reasoning capability to the reactive nature of these modelling tools, resulting in a very expressive mechanism for robotic behaviour.

A FSM is, in fact, a formal mathematical object. It consists of a finite set of states, an input language (for events), and a transition function. The transition function indicates the new state, given an input and the current state. Other adornments include signalling some states as initial and some as final or accepting. Actions can be associated with entering, and/or leaving a state, as well as activities that are performed while in a state. This model is used in formal ways in the theory of computation for describing regular languages. It has historically been used to formally describe computer networking protocols [6]. As a software-modelling instrument, it remains the main artefact for modelling the behaviour² of each object in a class in UML and has penetration in many approaches for the description of object behaviour in object-oriented systems as well as in agent systems. This model has been extended to describe behaviours in groups of agents and also appears in the literature in Computer Supported Collaborative Work (CSCW), namely *Team Automata* [20,10] (also fmt.isti.cnr.it/~mtbeek/TA.html). State machines have also been used as the foundation for languages to model agent behaviour in RoboCup [12]. In summary, FSMs remain a core tool for modelling software behaviour.

However, a fundamental aspect of FSMs is that the transition function is just that, a mathematical function (it provides only one value of the codomain for each value in the domain). That is, given an input and a state, only one transition fires and the machine moves to only one new state. Granted that the model can be extended to a non-deterministic automaton, where given an input and a state, a set of possible states is the outcome of the transition. However, in this case, the

¹ Our approach also has been applied with the asynchronous modelling provided by Petri nets, but this will not be described here.

² Conversely, the class diagram is the artefact to model the static structure of an object-oriented system.

semantics of the behaviour has several interpretations. Therefore, as a modelling instrument in software engineering, it is typically expected that the conditions emanating from a state are mutually exclusive and exhaustive. Namely, if the symbol c_i is a Boolean expression representing the guard of the transition, then it must be the case that $\bigvee_{i=1}^n c_i = \text{true}$ (the exhaustive condition), and $c_j \wedge c_i = \text{false} \quad \forall j \neq i$ (the exclusivity condition). When a finite-state machine is used for language recognition, if the automaton has a state where $\bigvee_{i=1}^n c_i \neq \text{true}$, it is said to be input-incomplete. Input-incomplete automata can be made input-complete by adding a transition from the state to itself labelled with $\neg \bigvee_{i=1}^n c_i$. When the number n of transitions out of a state is large, this may not be what the behaviour designer attempted to capture. While, formally, our logic will achieve something similar, the priority relation of the logic can better aid the system engineer in identifying the default cases.

Thus, the designer of behaviour faces the difficult challenge of describing and ensuring the validity of exclusive and exhaustive conditions that trigger the departure from each state. We automate such validation³. We propose to use a non-monotonic logic to describe the conditions that trigger a change of state. Non-monotonic logic models are better suited for incremental elicitation of the conditions that leave a state and those that keep control in the same state [1]. Non-monotonic reasoning ensures that the behaviour has a single, exhaustive interpretation, even if the original description does not and requires common sense to avoid cumbersome, “obvious” flaws. We generalise the language that guards transitions from symbols of events to entire reasoning exercises expressed in logics. We emphasise that the modelling remains deterministic, but what rule fires is determined by logical deduction in a declarative model.

3 Modelling a Microwave Oven

We illustrate our approach with the requirements engineering of a microwave oven and its simulation in BECIE (a tool for behaviour engineering using Behavior Trees) and under Linux and Mac OS X. We also demonstrate its direct implementation on both a LEGO Mindstorm and an Aldebaran Nao.

The microwave oven has been a standard example in software engineering [13, 19, 21] and requirements engineering [14, 23, 9]. Table I shows the requirements as presented by Myers and Dromey [14, p. 27, Table 1]. Although this is in fact not exactly the same as the original by Shlaer and Mellor [19, p. 36], it illustrates the point that human specifications are ambiguous. We invite the reader to revise these requirements, and because of the familiarity today with microwave ovens, most people would find Table I a complete and sufficient description of the behaviour of such a system (please do not read further until you have reviewed the requirements again).

We take this opportunity to emphasise how humans reason in a non-monotonic fashion. In fact, most people find that R5 means *pausing* the cooking and although *opening the door stops the cooking*, we do not clear the timer. That is,

³ Validation here means that the design represents a well-defined, intended behaviour.

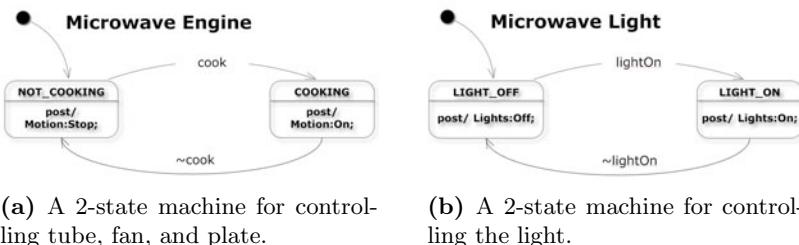
Table 1. One-Minute Microwave Oven Requirements

Req.	Description
R1	There is a single control button available for the use of the oven. If the oven is closed and you push the button, the oven will start cooking (that is, energise the power-tube) for one minute.
R2	If the button is pushed while the oven is cooking, it will cause the oven to cook for an extra minute.
R3	Pushing the button when the door is open has no effect.
R4	Whenever the oven is cooking or the door is open, the light in the oven will be on.
R5	Opening the door stops the cooking.
R6	Closing the door turns off the light. This is the normal idle state, prior to cooking when the user has placed food in the oven.
R7	If the oven times out, the light and the power-tube are turned off and then a beeper emits a warning beep to indicate that the cooking has finished.

closing the door resumes the cooking. This also implies that opening the door pauses the timer.

A distributed *whiteboard* [3] enables high level formalisation of requirements. Thus sensors, such as the microwave button, are hardware instruments that deposit a message on the whiteboard with the signature of the depositing module and a time stamp. Then, events like a door opening are detected on the whiteboard. Actions such as energising the microwave tube are communicated by simply placing a message on the whiteboard. Matters are a bit more complex, as messages on the whiteboard can expire and can have a priority, and thus actuators can be organised using “subsumption” [8].

Our methodology can be considered as building the score for an orchestra. We will have concurrent (parallel) FSMs. Each FSM controls the group of actuators that always behave the same by providing the cues for their state transitions. For example, in the microwave, energising the tube, running a fan, and spinning the plate all follow the same cues. However, the light does not and therefore would have a different FSM. We identify the states, and the transitions are labeled with a question to our inference engine. For example, the tube, the fan and the plate are at most in two possible states: a NOT_COOKING state or a COOKING state. We label the transition from NOT_COOKING to state COOKING with *cook* which is a predicate in the non-monotonic logic. It should be interpreted as the request to our deductive engine for an answer to “is it time to *cook*?” . The transition in the opposite direction is analogous. The label is \sim *cook* and again, we are

**Fig. 1.** Simple 2-state machines control most of the microwave

asking “is it time to not `cook`?” . It is not hard to apply this to the light which also has two exclusive states: `LIGHT_OFF` or `LIGHT_ON`. Again, in the transition from `LIGHT_OFF` to `LIGHT_ON` we use the label `lightOn` to represent a query to the inference engine, “is it time to turn the `lightOn`?” . In general, our labels are predicates of the non-monotonic logic and stand for the question “is it time to *LABEL*?” . The state diagrams for these machines appear in Fig. 1. We now turn our attention to the declarative module that answers these questions. The approach here is a iterative refinement. Consider the energising of the tube.

Behaviour 1: The default state is not to cook. However, if the timer indicates we have cooking time left, we should energise the microwave. Unless, of course, the door is open.

The above text directly corresponds to the logic program in Fig. 2a, depicting the code in the logic programming language DPL [16] which is an implementation of *Plausible Logic* [5][15][2]. Suffice it to say that Rule C0-C3 are the encoding of Behaviour 1. The rest of the code names the module, indicates that to make inferences with this logic program, the values of `timeLeft` and `doorOpen` are needed, and that the logic can make pronouncements with respect to `cook` and `~cook`. Fig 2b corresponds to the following behaviour.

Behaviour 2: Usually the light is off, but when the door is open or there is time left, the light is on.

Note the ease with which we can develop this logic declaration. We start with the usual common case or default (Rule L0), then we produce refinements. In this case, the two new rules (L1 and L2) take precedence over L0. In a similar way, a finite state machine is developed for the behaviour of the button that can be used to add time. This button is in the state of `ENABLED` when pressed or `DISABLED` when not pressed. We chose to ask the inference engine (when the button is `DISABLED`) “when should we `add`?” The reverse transition is a question for when is time to `~add` and move out of the `ENABLED` state. Fig. 3 captures the requirements of the button. However, a button push is ignored when the door is open. The modelling of the alarm is also similar. This appears in Fig. 4a. The alarm should start in the `BELL_OFF` but if the timer indicates there is time, the bell becomes armed

```
% MicrowaveCook.d
name{MICROWAVECOOK}.

input{timeLeft}.
input{doorOpen}.

C0: {}      => ~cook.
C1: timeLeft => cook. C1 > C0.
C2: doorOpen => ~cook. C2 > C1.

output{b cook, "cook"}.
output{b ~cook, "dontCook"}.
```

(a) DPL for 2-state machine controlling engine, tube, fan, and plate.

```
% MicrowaveLight.d
name{MICROWAELIGHT}.

input{timeLeft}.
input{doorOpen}.

L0: {}      => ~lightOn.
L1: timeLeft => lightOn. L1 > L0.
L2: doorOpen => lightOn. L2 > L0.

output{b lightOn, "lightOn"}.
output{b ~lightOn, "lightOff"}.
```

(b) DPL for 2-state machine controlling the light.

Fig. 2. Simple theories for 2-state machines

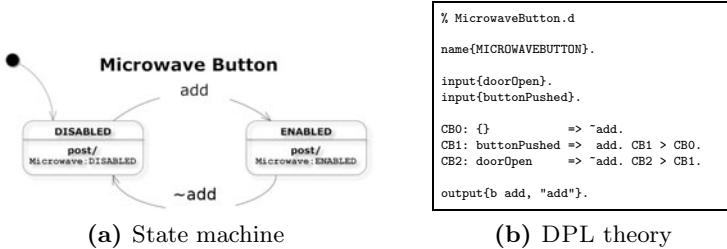
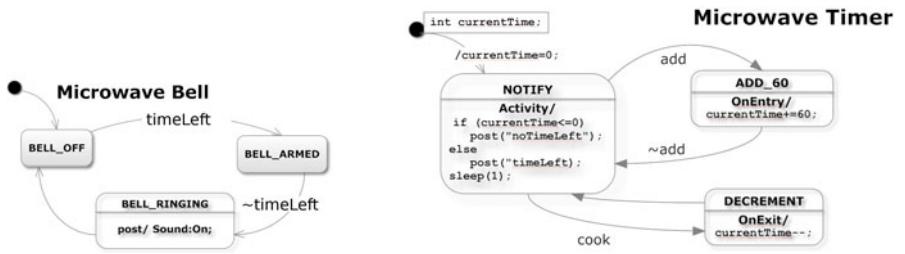


Fig. 3. The modelling of the button’s capability to add to the timer



(a) The bell’s capability to ring when the time expires.

(b) The state machine for the timer does not need a theory, but includes C++ code.

Fig. 4. FSMs for the bell and the timer use the logic theories already developed

and ready to ring. As soon as there is no time left it should move to the state **BELL_RINGING** and then back to **BELL_OFF**. Note that there is a transition with no label in Fig. 4a (such transition always fires as the null predicate evaluates to **true**). There is a simplification to this FSM by using actions in transitions. This occurs in our next case, the model of the timer (Fig. 4b).

This completes the requirements engineering process. Note that for the finite state machines we have used standard UML and OMT notations, where a transition may have *actions* that are performed *on entry* (illustrated in the timer model, where the initialisation sets the integer variable **currentTime** to zero) or *on exit* (also illustrated in the timer model Fig. 4b). Finally, we can handle *activities* that are performed while in a state (again, this appears in the model for the timer). For example, the button states (Fig. 3) have activities that post the state of the button to the whiteboard. The engine and the light (Fig. 1) post messages to the whiteboard that are read by the corresponding actuators.

We emphasise that with our architecture (and our implementations of FSM interpreters and inference engines), Figs 1, 2, 3, and 4 are all that is required for a fully functioning system! Provided that the C++ code for actions and activities is correct and matches the whiteboard API, all is translated into executable code. We simulated the full microwave in a POSIX environment (we have run the finite state machines simulating the microwave system on Ubuntu and on Mac OS X, see Fig. 5), and on a behaviour tree simulation

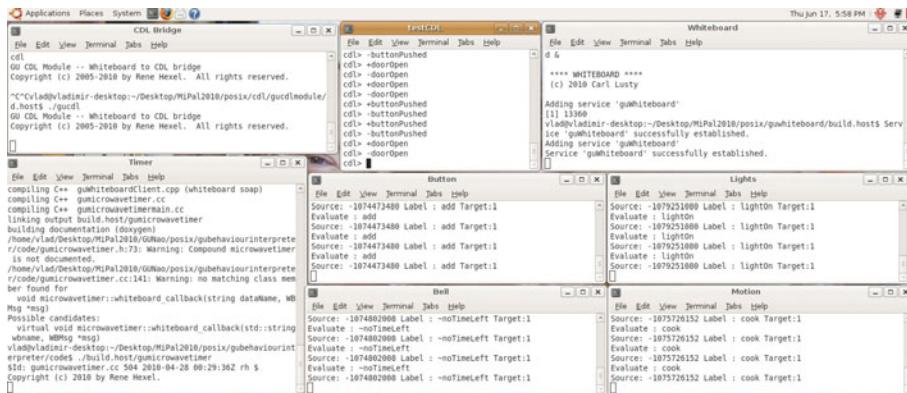


Fig. 5. Simulation of all 5 state machines in Ubuntu (top right: whiteboard; top left: CDL inference engine; top centre: button and door sensor simulation)

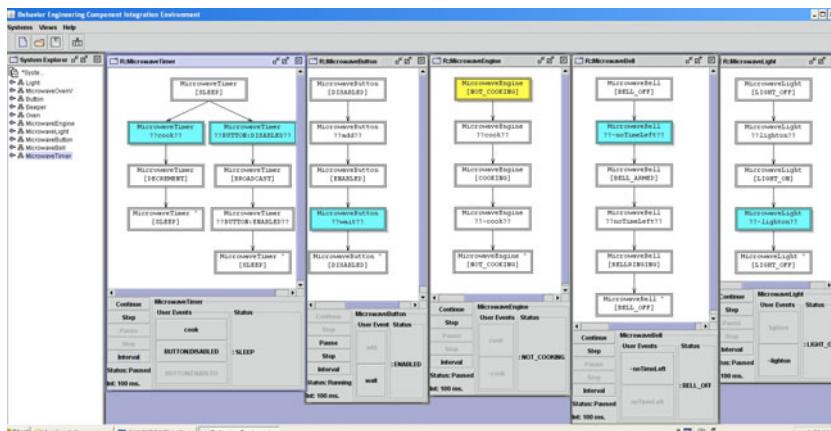


Fig. 6. A simulation snapshot of all 5 state machines using BECIE [23]

tool (see Fig. 6). Moreover, the models are automatically translated and uploaded onto a robot. We have done this for a LEGO MindStorm microwave (see <http://www.youtube.com/watch?v=iEkCHqSfMco>). To illustrate this point further, we re-wired the actuators to other actions on a Nao humanoid robot, and uploaded the behaviour onto the robot, allowing it to simulate a microwave (see <http://www.youtube.com/watch?v=Dm3SP3q9%5FVE>; the video also demonstrates distribution, as inputs are provided from an iPhone or a PC, it demonstrates the whiteboard decoupling as Nao's eyes are the light, moving the head is the engine, and making a sound is the alarm). We can also distribute the modules across a network. E.g., we can run the alarm logic on a desktop computer and have the sounds emitted from the speakers of the laptop, or even on an iPhone, while other behaviours execute on the robot or on simulators.

3.1 Modelling Inter-machine Communication

We re-emphasise that all we need to execute the microwave system is the models described earlier. Not a single line of additional code is required. Only configuration files are modified to determine which subsystems execute which modules and where on the network to locate the whiteboard. However, from the software engineering point of view, it is important to visualise and represent the inter-module communication mediated by the whiteboard. Formally, this is not necessary, but enables further understanding of the overall system. For this, we suggest to use *Behavior Trees* [23][24][22]. Here, the system with all modules can be integrated into a single behavior tree [22] (Fig. 7). We have already demonstrated the equivalence between behavior trees and state diagrams [4] and also represented state machines with DPL. However, the behavior tree approach offers a methodology to obtain visualisation of interactions (what the methodology refers to as the *component interaction network* [23]). In our illustration with the microwave system, the obtained interaction network appears in Fig. 8.

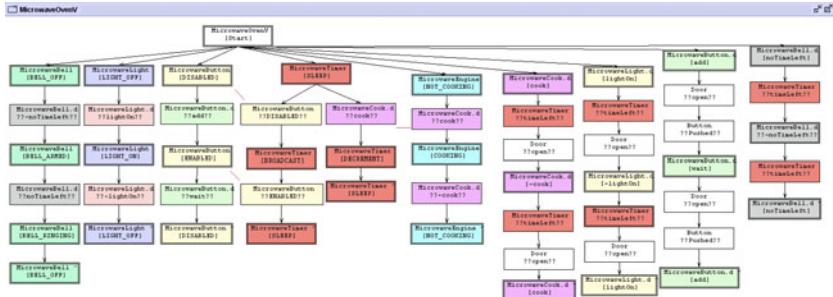


Fig. 7. Behavior tree with all components for finding the component diagram

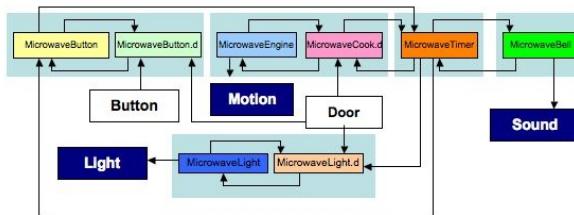


Fig. 8. The resulting component interaction network after the behavior tree analysis (white boxes: inputs/sensors; dark boxes: outputs/actuators; highlighted: the five state machines); the microwave timer plays a central coordination role

4 Modelling Agent Behaviour – A Spider

We now illustrate the design process using our approach in a more complex example of animal-like behaviour or an agent for a simple autonomous robot.

- Charlotte is a nocturnal female spider. At dusk she wakes up, spins a web, and then goes about the spidery business of waiting, eating, mating, and laying eggs. At dawn, she eats her web, then sleeps through the day (hiding away, lest she gets eaten by a bird). Eating her web ensures she is not hungry during the day, and has enough resources to spin another tomorrow evening.
- Spinning her web makes Charlotte very very hungry. She will stay hungry until she has eaten three flies or equivalent (smaller male spiders). Hungry or not, Charlotte never turns down a chance to eat something. Charlotte is receptive to mating if she has not mated yet, or has laid eggs since her last mating, and is not tired and not hungry. Mating and laying eggs make her tired, and only a good day's sleep will refresh her.
- Male spiders come along at night with the intent to mate. If Charlotte is receptive, they might get lucky. If Charlotte is hungry, “too bad”, the male gets eaten immediately. Clever males bring a fly gift for her to eat, or don't come alone, so that it is more likely that her hunger can be satisfied. Luckily, flies are preferred as food. If Charlotte is not too hungry, she will mate, and only then eat the (evolutionarily-speaking) lucky male.
- Charlotte lays eggs at the first opportunity, but only at night, when she is not hungry, and not tired.

Fig. 9. The description of Charlotte's behaviour

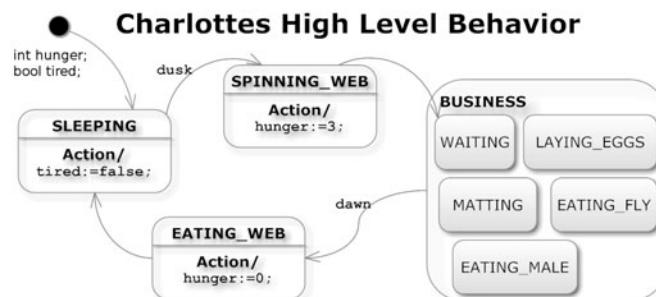


Fig. 10. High level spider behaviour with C++ actions and composite states

Fig. 9 describes the behaviour in natural language. We start with some simple finite state machines. We assume there is a light sensor that can detect day/night changes, and thus, using composite states we can obtain a high level state diagram for Charlotte (Fig. 10). Consequently, **dusk** and **dawn** are simple predicates of the logic that test the sensors' input placed on the whiteboard. However, the detailed behaviour of Charlotte's night business is more complicated, but solvable with our approach. The finite state machine (Fig. 11) is composed of sub-states. Every time a fly lands on the web, the fly count is incremented and similarly, every time a male spider appears on the web, the male count is incremented. Again, Charlotte's sensors can detect captured flies, placing a simple message on the whiteboard that set the predicate **fly landed** to **true** without need for anything else besides direct inference. The same applies when a male appears. Whenever Charlotte eats one of the flies on her web, the fly count is decremented (note that as in UML or OMT an action in a transition is placed after a slash “/” and a transition with a null predicate is always **true**). Similarly, every time she eats a male, the male count decrements. After mating, the “lucky male” also ensures that Charlotte's hunger is reduced through his sacrifice, resulting in a transition with no label from **MATING** to **EATING_MALE**.

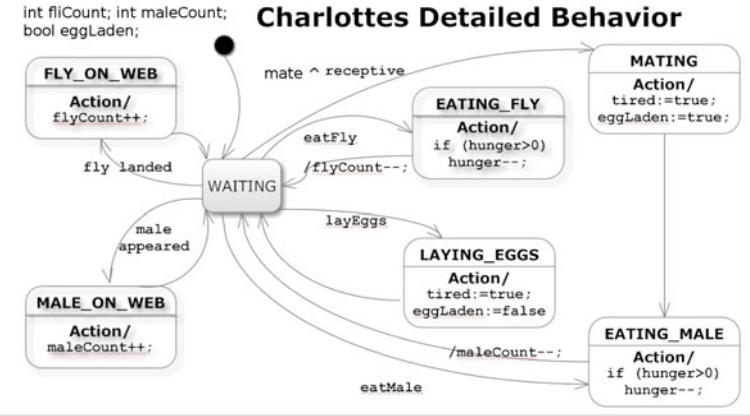


Fig. 11. Sub-Behaviours of Charlotte's business behaviour with C++ actions

```

% receptiveness to mating
Rec: () => receptive.
NotRec: "hunger > 0" => ~receptive. NotRec1 > Rec.
NotRec2: tired => ~receptive. NotRec2 > Rec.
NotRec3: eggLaden => ~receptive. NotRec3 > Rec.

% Default is to wait.
Wait: () => wait.
Wait: () => ~mate.
Wait: () => ~eatFly.
Wait: () => ~eatMale.
Wait: () => ~layEggs.
Wait: () => ~layEggs.

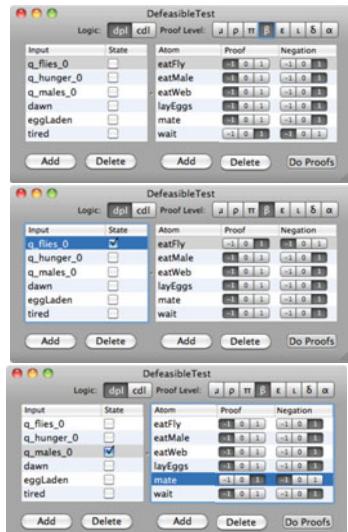
% Mating is less urgent than any other item of business.
Mate: "maleCount > 0" => mate. Mate > Wait.

% Eating beats mating if hungry.
EatMale: {"maleCount > 0", ~receptive} => eatMale. EatMale > Wait.
EatMale > Mate.

% Eating flies beats eating males or mating.
EatFly: "flyCount > 0" => eatFly. EatFly > Wait. EatFly > Mate.
EatFly > EatMale.

% Prefer to lay eggs as soon as possible.
Lay: {eggLaden, "tired, ~"hunger > 0"} => layEggs. Lay > Wait.
Lay > Mate. Lay > EatMale. Lay > EatFly.

% Overriding urgency at dawn is to eat web and hide.
EatWeb: dawn => eatWeb. EatWeb > Wait. EatWeb > Mate.
EatWeb > EatMale. EatWeb > EatFly. EatWeb > Lay.
  
```



(a) Simple theory for Charlotte.

(b) Inference Simulation

Fig. 12. Spider Business DPL

Now the logic's power really shines through predicates like `receptive` which, again, translate to the question “shall we move to MATING?” Thus, the predicate `eatFly` is also a question to the inference engine to establish whether there is a fly on the web. Similarly, `eatMale` and `layEggs` are requests for deducing if Charlotte shall enter the states `EATING_MALE` or, respectively, `LAYING_EGGS`. We do not present the entire DPL program here, just the main rules (Fig. 12a) that establish the transition out of `WAITING` and their simulation (Fig. 12b). Other states return back to `WAITING` when their activity is completed.

5 Conclusions

We have demonstrated the viability of our approach to capture the requirements of, model, simulate, and deploy a robotic system in a fully automated way. While we have no space here to describe other illustrative cases of our approach, suffice it to say that we have also modelled sophisticated embedded systems (such as the cruise control of a car) that have been described in the literature.

References

1. Antoniou, G.: The role of nonmonotonic representations in requirements engineering. *Int. J. of Software Engineering and Knowledge Engineering* 8(3), 385–399 (1998)
2. Billington, D.: The proof algorithms of plausible logic form a hierarchy. In: Zhang, S., Jarvis, R.A. (eds.) *AI 2005. LNCS (LNAI)*, vol. 3809, pp. 796–799. Springer, Heidelberg (2005)
3. Billington, D., Estivill-Castro, V., Hexel, R., Rock, A.: Architecture for hybrid robotic behavior. In: Corchado, E., Wu, X., Oja, E., Herrero, Á., Baruque, B. (eds.) *HAIS 2009. LNCS*, vol. 5572, pp. 145–156. Springer, Heidelberg (2009)
4. Billington, D., Estivill-Castro, V., Hexel, R., Rock, A.: Plausible logic facilitates engineering the behavior of autonomous robots. In: Fox, R., Golubski, W. (eds.) *IASTED Conf. on Software Engineering (SE 2010)*, Anaheim, California, USA, February 16 - 18, pp. 41–48. IASTED, ACTA Press, Innsbruck, Austria (2010)
5. Billington, D., Rock, A.: Propositional plausible logic: Introduction and implementation. *Studia Logica* 67, 243–269 (2001)
6. Bochmann, C.V., Sunshine, C.A.: Formal methods in communication protocol design. *IEEE Transaction on Communications* 28(4), 624–631 (1980)
7. Brooks, R.A.: How to build complete creatures rather than isolated cognitive simulators. In: VanLehn, K. (ed.) *Architectures for Intelligence*, pp. 225–239. Lawrence Erlbaum Associates, Hillsdale (1991)
8. Brooks, R.A.: Intelligence without reason. In: Myopoulos, R., Reiter, R. (eds.) *Proc. 12th Int. Joint Conference on Artificial Intelligence, ICJAI 1991*, San Mateo, CA, pp. 569–595. Morgan Kaufmann Publishers, Sydney (1991)
9. Dromey, R.G., Powell, D.: Early requirements defect detection. *TickIT Journal* 4Q05, 3–13 (2005)
10. Ellis, C.: Team automata for groupware systems. In: *GROUP 1997: Proc. of the Int. ACM SIGGROUP Conf. on Supporting group work*, pp. 415–424. ACM, New York (1997)
11. Harel, D., Politi, M.: *Modeling Reactive Systems with Statecharts: The STATE-MATE Approach*. McGraw-Hill, New York (1998)
12. Lötzsch, M., Bach, J., Burkhard, H.-D., Jüngel, M.: Designing agent behavior with the extensible agent behavior specification language XABSL. In: Polani, D., Brownning, B., Bonarini, A., Yoshida, K. (eds.) *RoboCup 2003. LNCS (LNAI)*, vol. 3020, pp. 114–124. Springer, Heidelberg (2004)
13. Mellor, S.J.: Embedded systems in UML. *OMG White paper label: We can generate Systems Today* (2007), <http://www.omg.org/news/whitepapers/>
14. Myers, T., Dromey, R.G.: From requirements to embedded software - formalising the key steps. In: *20th Australian Software Engineering Conf (ASWEC)*, Gold Coast, Australia, April 14–17, pp. 23–33. IEEE Computer Soc., Los Alamitos (2009)

15. Rock, A., Billington, D.: An implementation of propositional plausible logic. In: Edwards, J. (ed.) 23rd Australasian Computer Science Conf., Canberra. Australian Computer Science Communications, vol. 22(1), pp. 204–210. IEEE Computer Soc., Los Alamitos (January 2000)
16. Rock, A.: The DPL (decisive Plausible Logic) tool. Technical report (continually) (in preparation), <http://www.cit.gu.edu.au/~arock/>
17. Rumbaugh, J., Blaha, M.R., Lorenzen, W., Eddy, F., Premerlani, W.: Object-Oriented Modelling and Design. Prentice-Hall, Englewood Cliffs (1991)
18. Samek, M.: Practical UML Statecharts in C/C++: Event-Driven Programming for Embedded Systems, 2nd edn., Newnes (2008)
19. Shlaer, S., Mellor, S.J.: Object lifecycles: modeling the world in states. Yourdon Press, Englewood Cliffs (1992)
20. ter Beek, M.H., Ellis, C.A., Kleijn, J., Rozenberg, G.: Synchronizations in team automata for groupware systems. Computer Supported Cooperative Work (CSCW) 12(1), 21–69 (2003)
21. Wagner, F., Schmuki, R., Wagner, T., Wolstenholme, P.: Modeling Software with Finite State Machines: A Practical Approach. CRC Press, NY (2006)
22. Wen, L., Colvin, R., Lin, K., Seagrott, J., Yatapanage, N., Dromey, R.G.: “Integrare”, a collaborative environment for behavior-oriented design. In: Luo, Y. (ed.) CDVE 2007. LNCS, vol. 4674, pp. 122–131. Springer, Heidelberg (2007)
23. Wen, L., Dromey, R.G.: From requirements change to design change: A formal path. In: 2nd Int. Conf. on Software Engineering and Formal Methods (SEFM 2004), Beijing, China, September 28-30, pp. 104–113. IEEE Computer Soc., Los Alamitos (2004)
24. Wen, L., Kirk, D., Dromey, R.G.: A tool to visualize behavior and design evolution. In: Di Penta, M., Lanza, M. (eds.) 9th Int. Workshop on Principles of Software Evolution (IWPSE 2007), in Conjunction with the 6th ESEC/FSE Joint Meeting, Dubrovnik, Croatia, September 3-4, pp. 114–115. ACM, New York (2007)

Implementing Automated Robot Task Planning and Execution Based on Description Logic KB

Joonmyun Cho, Hyun Kim*, and Joochan Sohn

Robot/Cognitive System Research Dept.
Electronics and Telecommunications Research Institute,
Daejeon, Korea
`{jmcho, hyunkim, jcsohn}@etri.re.kr`

Abstract. For a service robot to perform a given task without human intervention, the robot must be able to make a plan, i.e., a sequence of high-level actions and execute it autonomously. Recently, a few researchers have investigated Description Logic-based task planning techniques. In this paper we discuss our implementation methods of Description Logic-based task planning and execution. Some knowledge representation scheme is investigated and a simple planning algorithm is conceived. Some techniques are conceived to bridge the gap between the symbolic world and the program world in order to execute the task plan. We also explain our experiments and the results we have got.

Keywords: task planning, autonomous robot, knowledge base, goal-driven architecture, description logics.

1 Introduction

For a service robot to perform a given task without human intervention, the robot must be able to make a plan and execute it autonomously. In an abstracted view, the task planning is the finding of an executable sequence of the behaviors, i.e., high-level actions of the robot, that leads from the initial state of the world to the final desired state, given the description of world, actions and a goal [1-4]. For example, given a task to bring milk bottle in the refrigerator, an autonomous robot must be able to determine from its several possible behaviors a sequence of behaviors like 'move to the kitchen', 'dock to the refrigerator', 'open the door of the refrigerator', 'find the milk bottle', 'pick up the milk bottle', 'close the door of the refrigerator', and 'move to the user'. And then the robot system has to execute the task plan, i.e., the behaviors in the sequence. As the robot carries out each of the behaviors in due turn, the world's state changes from the initial state finally to the desired state. In this abstraction, a behavior can be viewed as a transition between one state to another whose pre-condition must accord with the previous state and whose post-conditions must accord with the successor state [5].

* Corresponding author. address: Robot/Cognitive System Research Dept. ETRI, 138 Gajeongno, Yuseong-gu, Daejeon, 305-700, Korea; tel: +82-42-860-5626; fax: +82-42-860-1566.

The characteristics of the task planning vary according to whether or not the state of world at a time is uniquely determined by a task plan, i.e., a sequence of behaviors and the initial state [2,5]. The deterministic planning is based on such assumptions that any events which are not presupposed and thus not described in the world model do not occur and that a behavior always has only one result in the world. Those assumptions, however, do not make sense in real world. For example, for the assumption about environments a moving object or human could block the way of the robot and for the assumption about behaviors the 'open the door of the refrigerator' behavior fails from time to time.

In this paper, we, the authors, deal with the deterministic task planning with real world considerations. We think, although the deterministic planning has in theoretical sense limits to be applied to all the possible cases, in practical sense it could be useful for many cases. We could cope with the unexpected events in environments by making the behaviors more tolerable. For example, the move behavior could be implemented to be able to avoid obstacles which is not expected and thus not described in the world model. We could also overcome the conditional nature of robot behaviors through appropriate observation. For example, when the robot fails in opening the refrigerator's door we could notice that situation using a touch sensor installed in the door and make the robot retry to open or re-plan from the present world.

Existing task planning techniques based on deductive reasoning have such drawbacks that their reasoning is undecidable by using first-order logics in representing the task's goal, world model, and pre-/post-conditions of behaviors or that by using variants of propositional logics in contrast, their expressive power is limited [6]. STRIPS [7] is the typical approach among the early planning researches. In this approach any literal in first-order logic must be ground and function-free, so that there are many restrictions especially in describing the world [2].

Recently, a few researchers have investigated Description Logic-based task planning techniques. These approaches provide with considerable expressive power going beyond propositional logic, while reasoning is still decidable and theoretically needs less computing time than restricted from of first-order logics [6,8]. Description Logics (DLs) [9] are a well-known family of knowledge representation formalisms that may be viewed as fragments of first-order logic (FO). With the advances of Semantic Web technologies, we can make use of ample languages and tools for DLs. OWL DL [10] that W3C has recommended as a standard is one of the most well-known languages.

In this paper, we discuss the implementation of a Description Logic-based automated task planning. Our implementation uses a description logic knowledge base (DL KB) which has been extended to include the description of task's goal and behaviors. As the world, behaviors, and task's goal are all described in DL formalism, a DL-based task planning techniques can be implemented on it with simple algorithms and without any translation to other formalism such as of STRIPS.

2 Related Works

Milicic presented a formal definition of actions and task planning in the boundary of ALCQIO DL and showed its properties and implications [6,8]. The followings are Milicic's formal definitions of actions and task planning [8].

Definition 1. (Action, operator). Let N_X be a countably infinite sets of variables, disjoint with N_C , N_R and N_I , where N_C is a set of concept names, N_R is a set of role names, and N_I is a set of individual names. Moreover, let T be an acyclic TBox. A primitive literal for T is an ABox assertion

$$A(a), \neg A(a), r(a, b), \text{ or } \neg r(a, b)$$

with A a primitive concept name in T , r a role name, and $a, b \in N_I$. An atomic action $\alpha = (\text{pre}, \text{post})$ for T consist of

- a finite set pre of ABox assertions, the pre-conditions;
- a finite set post of post-conditions which is a primitive literal for T .

A composite action for T is a finite sequence $\alpha_1, \dots, \alpha_k$ of atomic actions for T . An operator for T is a parameterized atomic action for T , i.e., an action in which definition variables from N_X may occur in place of individual names.

Definition 2. (Executability). Let T be an acyclic TBox, $\alpha_1, \dots, \alpha_k$ a composite action for T with $\alpha_i = (\text{pre}_i, \text{post}_i)$, and A an ABox. The composite action is executable in A w.r.t. T iff the following conditions are true for all models I of A and T :

- $I \models \text{pre}_1$
- for all i with $1 \leq i < k$ and all interpretations I' with $I \Rightarrow^T_{(\alpha_1, \dots, \alpha_k)} I'$, we have $I' \models \text{pre}_{i+1}$.

Definition 3. (Planning). Planning is a tuple $\Pi = (\text{ind}, T, \text{Op}, A, \Gamma)$, where

- ind is a finite set of individual names;
- T is an acyclic TBox based on ind ;
- Op is a finite set of atomic operator for T ;
- A (initial state) is an ABox based on ind ;
- Γ (goal) is an ABox based on ind ;

A plan in Π is a composite action $\alpha = \alpha_1, \dots, \alpha_k$, such that $\alpha_i \in \text{Op}[\text{ind}]$, $i = 1 \dots k$. A plan $\alpha = \alpha_1, \dots, \alpha_k$ in Π is a solution to the planning task Π iff:

1. is executable in A w.r.t T ; and
2. for all interpretations I and I' such that $I \models A, T$ and $I \Rightarrow^T_\alpha I'$, it holds that $I' \models \Gamma$.

From the Milicic's model we can see that we need two types of reasoning in order to solve the task planning problem. The first one is the reasoning about *executability* and the second one is about *projection*. Executability is the problem of whether actions can be applied in a given situation, i.e., if pre-conditions are satisfied in the state of the world. Projection is the problem of whether applying actions achieves the desired effect, i.e., whether assertions that we want to make true really holds after executing the actions. Milicic showed the two reasoning task can be reduced to the task of standard DL reasoning that corresponds to ABox consistency checking [6,8].

3 Implementation Methods

In this section we discuss the main methods by which we implemented the automated task planning and execution based on the Milicic's formal model. The discussion is divided into tree subsections. First, we discuss the extension of the robot DL KB for automated task planning. Second, we discuss the algorithms for generating a task plan. Third, we discuss some techniques to execute the task plan.

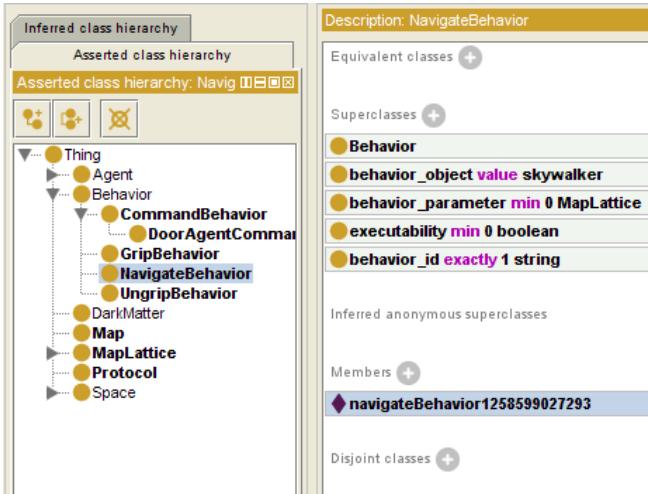
3.1 Extension of Robot Knowledge Base

For task planning, first of all, we should represent the knowledge of behaviors in the robot knowledge base. According to Milicic, a behavior (i.e., an action) is defined with pre- and post-conditions, and the pre- and post-conditions are described using parameterized ABox assertions. Milicic called the parameterized behavior an operator [8]. Consequently, the behaviors that are included in a task plan are in fact the instances of the operators. In other words, the concrete pre- and post-conditions are determined when specific individuals substitute for the parameters, and a task plan is made of those behaviors. (From now on, we use the 'operator' to mean the parameterized behavior, and the 'behavior' for the concrete behavior.)

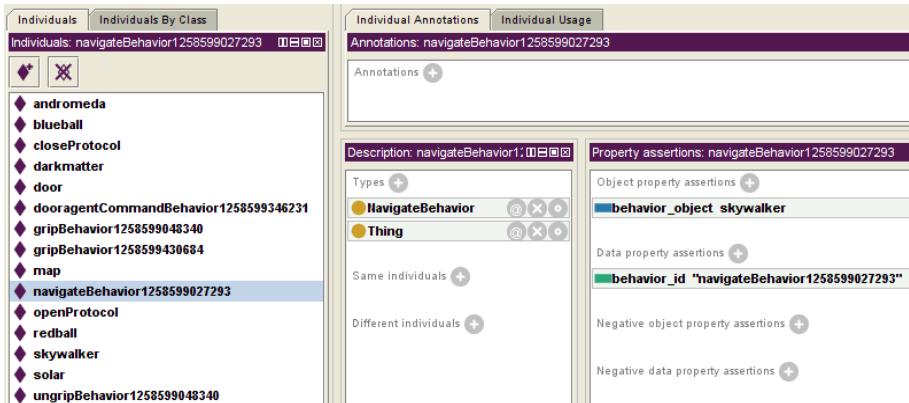
In our methods, we represent the operators like 'pick up something' as classes in our knowledge base, and the behaviors like 'pick up the milk bottle' as individuals of the operator classes. Thus, whenever an individual (e.g., milk bottle) is created in the knowledge base that can substitute for the parameter (e.g., 'something') of an operator (e.g., 'pick up something'), we create the concrete behavior (e.g., 'pick up the milk bottle') as an instance of the operator class. We also specify the possible types (e.g., Bottle class) whose instances (e.g., milk bottle) can substitute for the parameter as some object properties of the operator class. Fig. 1 shows an example of our representation of behavior knowledge.

The pre-conditions of a behavior specify the conditions at a specific time under which the behavior is applicable. For example, when the milk bottle is near the robot, the behavior 'pick up the milk bottle' is applicable, but the behavior may not be applicable at another time when the milk bottle is not near. Any pre-conditions can finally be reduced to a boolean value. In our methods, every operator class has a data property (**executability** data property in Fig. 1-(a)) so that a behavior can have a boolean value that represents its executability. Consequently, there are two requirements for representing the pre-conditions. The first one is that parameterized knowledge can be described. And the second one is that the boolean value is continuously evaluated according to the change in the world model. SWRL [11] satisfies such requirements. We represent the pre-conditions of a behavior by a SWRL rule in which the body part has a conjunctive statement about the conditions and the head part has an assertion about executability property. Fig. 2 shows the examples of our pre-condition representations.

Intuitively, the post-conditions specify the effects of a behavior to the world. The effects can be reduced to ABox assertions some of which have to be added to and some of which have to be deleted from the description of the world. There are two requirements just as in representing the pre-conditions. The first one is that we can



(a) An example of our representation of an operator. `NavigateBehavior` is the operator class. The `behavior_object` object property and the `behavior_parameter` object property represent the `NavigateBehavior` operator's parameters. The individual `navigateBehavior1258599027293` is the concrete behavior of the operator.



(b) An example of our representation of a behavior. The concrete behavior `navigateBehavior1258599027293` is created as an individual of `NavigateBehavior` operator class when the individual `skywalker` is created.

Fig. 1. Examples of our representation of an operator and a behavior

specify the effects (i.e., ABox assertions) in the parameterized form. The second one is that we can delete the negative effects from the knowledge base as well as add the positive effects to the knowledge base. By using SWRL, however, we cannot describe such post-conditions because, though the SWRL rule can add new ABox assertions to knowledge base, it cannot delete existing knowledge because of the monotonicity assumption [11]. In our methods, a simple scripting language is conceived that can

Rules:	UI
Galaxy(?g) , NavigateBehavior(?nb) , TestAgent(?ta) , belong(?ta, ?g) , gateway(?g, ?da) , location_map_lattice(?ta, ?taml) , behavior_id(?nb, ?nb_id) , state(?da, "open") -> behavior_parameter(?nb, ?taml)	@ X O
NavigateBehavior(?nb) , Skywalker(?skywalker) , behavior_object(?nb, ?skywalker) -> executability(?nb, true)	@ X O
Galaxy(?g) , NavigateBehavior(?nb) , System(?s) , constitute(?s, ?g) , gateway(?g, ?da) , principal_location_map_lattice(?s, ?ml) , behavior_id(?nb, ?nb_id) , state(?da, "open") , system_id(?s, ?sname) -> behavior_parameter(?nb, ?ml) \wedge temporal_landmark_name(?ml, ?sname)	@ X O
MapLattice(?ml) , Skywalker(?skywalker) , location_map_lattice(?skywalker, ?ml) -> location(?skywalker, ?ml)	@ X O
Galaxy(?g) , NavigateBehavior(?nb) , gateway(?g, ?da) , principal_location_map_lattice(?g, ?ml) , behavior_id(?nb, ?nb_id) , galaxy_id(?g, ?gname) , state(?da, "open") -> behavior_parameter(?nb, ?ml) \wedge temporal_landmark_name(?ml, ?gname)	@ X O
DoorAgent(?da) , DoorAgentCommandBehavior(?dacb) , behavior_object(?dacb, ?da) -> executability(?dacb, true)	@ X O
MapLattice(?lml) , NavigateBehavior(?nb) , behavior_id(?nb, ?nb_id) , granted_landmark_name(?lml, ?lml_ln) -> behavior_parameter(?nb, ?lml)	@ X O
MapLattice(?ml) , TestAgent(?ta) , location_map_lattice(?ta, ?ml) -> location(?ta, ?ml)	@ X O
Skywalker(?skywalker) , TestAgent(?ta) , belong(?ta, ?skywalker) -> company(?ta, ?skywalker)	@ X O
NavigateBehavior(?nb) , System(?s) , TestAgent(?ta) , belong(?ta, ?s) , constitute(?s, ?g) , gateway(?g, ?da) , location_map_lattice(?ta, ?taml) , behavior_id(?nb, ?nb_id) , state(?da, "open") -> behavior_parameter(?nb, ?taml)	@ X O
GripBehavior(?gb) , Skywalker(?skywalker) , Space(?space) , TestAgent(?ta) , behavior_object(?gb, ?ta) , belong(?ta, ?space) , location_map_lattice(?skywalker, ?skywalker_lml) , location_map_lattice(?ta, ?ta_lml) , sameAs(?skywalker_lml, ?ta_lml) -> executability(?gb, true)	@ X O

Fig. 2. Examples of our representation of pre-conditions using SWRL. The body part of a rule is a conjunctive statement about the conditions and the head part is an assertion about **executability** property.

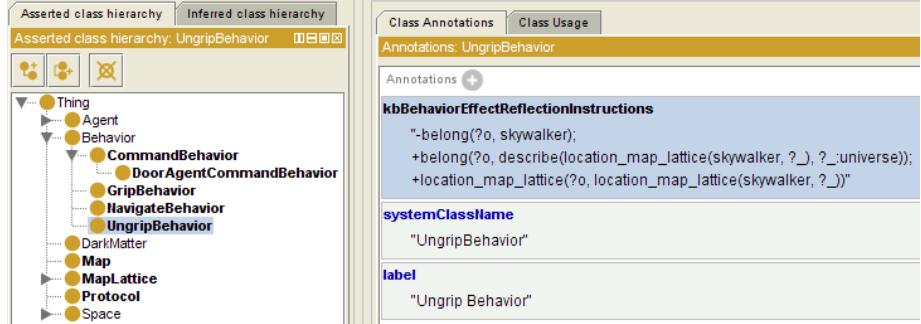


Fig. 3. An example of our representation of an operator's parameterized post-conditions. The parameterized post-conditions are represented as an annotation named **kbBehaviorEffectReflectionInstruction**. '+' and '-' means that the following ABox assertion must be added to or deleted from the knowledge base. A parameterized ABox assertion can have nested assertions. '?_' means variable for unknown individual. '?o' is the pre-defined variable that will be substituted for by the behavior's object individual. '?p' is the pre-defined variable for the behavior's supplement individual. ':' behind the variables indicates the default individual.

describe the parameterized ABox assertions just as SWRL and, furthermore, specify the negative effects as well as the positive effects. We represent the post-conditions as an annotation of the operator classes. The concrete post-conditions, i.e., the concrete ABox assertions of a specific behavior are made by substituting specific individuals for the parameters at run-time. Fig. 3 shows an example of our representation of an operator's parameterized post-conditions.

For our task planning module to interpret the representation of the parameterized post-conditions and make the concrete ABox assertions at run-time, we needed to

contrive a conceptual model of behaviors. (In contrast, the parameterized pre-conditions represented in a SWRL rule are interpreted and applied automatically by the DL reasoning engine.) In other words, the task planning module can be implemented only after we make a model that prescribes what a behavior means, how many parameters an operator can have, what the role of each parameter is, and so on. In our methods, we define a behavior as the verb in the sentences of the structure like 'subject – verb – object – adverbial complement' or 'subject – verb – object'. The subject of every behavior is always the robot itself. The object of a behavior is an individual that is the target of the behavior. And some behaviors can have an extra individual that plays the role of adverbial complement. For example, the behavior 'move to the kitchen' can be interpreted as that the robot (subject) moves (verb) itself (object) to the kitchen (adverbial complement).

In our methods, therefore, an operator must have one object parameter and can have another parameter that corresponds to a complement. With behaviors created according to the conceptual model, we can be assured that, when the task planning module interprets the parameterized post-conditions, it can always find the right individuals for the unknown variables (i.e., '?_-' variables in Fig. 3) by traversing along the relations from the already known object individual or the complement individual of the behavior to the target individual.

3.2 Task Planning Algorithms

Our algorithm for the task planning is a kind of the forward-chaining technique. It makes an executable sequence of behaviors evaluating all the possible behaviors from the initial state of the world until the successor state accords with the goal state. The process is virtually conducted as the successor states are constructed not by physically executing real behaviors but by only updating the knowledge base using the post-conditions. The algorithm can be described as following pseudo codes.

```

1 PLAN()
2     initial_plan_list = make empty list of plans;
3     solution = RESEARCH(initial_plan_list);
4     return solution;
5
6 RESEARCH(plan_list)
7     FOR size of plan_list
8         plan = get next plan from plan_list;
9
10    kb_changes_for_working_kb = plan.kb_changes;
11    working_kb = update initial_kb with kb_changes_for_working_kb;
12
13    executable_behaviors = query executable behaviors from working_kb;
14    FOR size of executable_behaviors
15        behavior = get next behavior from executable_behaviors;
16        add behavior to plan.behavior;
17
18        kb_changes_for_successor_kb
19            = make assertions for kb update from
20                post-conditions of behavior;
21        add kb_changes_for_successor_kb to plan.kb_changes;
22
23        successor_kb
24            = update working_kb with kb_changes_for_successor_kb;
      IF successor_kb entails goal
          restore successor_kb to working_kb with
              kb_changes_for_successor_kb;
          restore working_kb to initial_kb with
              kb_changes_for_working_kb;

```

```

25         RETURN plan;
26     EndIF;
27
28     restore successor_kb to working_kb with
29         kb_changes_for_successor_kb;
30     EndFOR;
31
32     restore working_kb to initial_kb with kb_changes_for_working_kb;
33 EndFOR;
34
35 RETURN RESEARCH(plan_list);

```

As the pseudo codes depict, the algorithm uses breadth-first tree search technique. The search space is composed of trees the nodes of which are applicable behaviors. There are zero or more root nodes because there can be no applicable behavior or several applicable behaviors depending on the given initial state. Since the algorithm uses breadth-first search technique, it always produces the task plan that contains the minimum numbers of behaviors if possible plans (i.e., paths) exist. In the real world it cannot be guaranteed that the minimum-number-of-behavior task plan is the optimal plan, but in the most cases it would be the best solution.

In the 11th line, the task planning module updates the knowledge base so that the knowledge base contains the current state of the world. It can be done by reflecting in order all the changes (i.e., post-conditions) of the behaviors from the root node (i.e., behavior) to the current node into the knowledge base. In the 13th line, the currently applicable behaviors are determined by querying the knowledge base about the behaviors the executability properties of which are true. In the 21th line, the module updates the knowledge base so that the knowledge base can contain the successor state attained in the case that the currently evaluating behavior is executed. The effects (i.e., ABox assertions to be added or deleted) of the currently evaluating behavior are made at run-time, in the 18th line, through interpreting the parameterized post-conditions. Then, in the 22th line, the task planning module evaluates whether the knowledge base implies the goal statement. If the goal statements are implied by the knowledge base (i.e., the successor state), the module returns the sequence of behaviors from the root node to the currently evaluating node. The evaluation of the goal implication is simply conducted by querying the goal statements from the knowledge base. Before the task planning module returns the task plan, it restores the knowledge base to the initial knowledge base in the 28th line and 31th line.

3.3 Task Plan Execution Techniques

Intuitively, the task plan execution is a process of sequential execution of the behaviors contained in the task plan. But the task plan actually contains only symbols (i.e., names) of behaviors, behavior's objects, and behavior's complements. What is really needed is the way for the task plan execution module to invoke the program objects using the names. Therefore, some extra information is needed to be represented to bridge the gap between the symbolic world and the program world.

First of all, the task plan execution module needs the names of the program classes that implement the behaviors to create the program objects through the reflection mechanism at run-time. (The name of behavior knowledge class (i.e., operator) may differ from the program class' name.) After a program object is created, the task plan execution module has to get real parameter values such as the coordinates of

refrigerator from the knowledge base. Therefore the module also needs the names of properties that represent knowledge object's attributes.

In our methods, this information is specified in the operator's annotations because, though the information itself varies from operator to operator, it is the same for all the behavior objects of the same operator. The annotation's name is also the same for all of the operators because all the operators are described according to the model of behavior as explained in Section 3.1. Fig. 4 shows an example of the specification of such information.

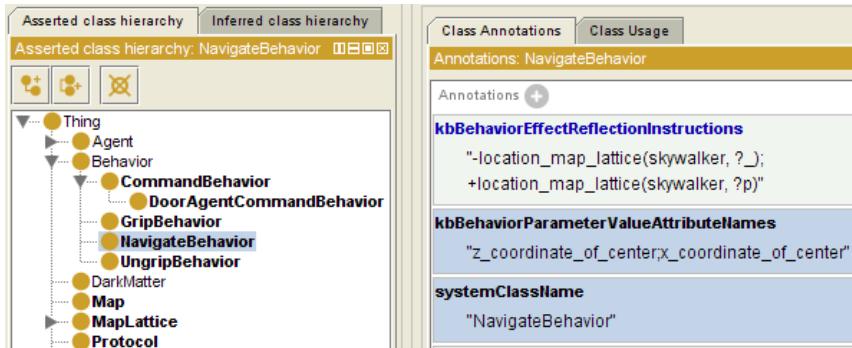


Fig. 4. An example of specification of execution-related information. The annotation `systemClassName` specifies the name of behavior system class. The annotation `kbBehaviorParameterValueAttributNames` specifies the knowledge base property names necessary when querying attribute values.

4 Prototype System and Experiments

In this section we present the results of our experiments after brief discussion of the experiment environment, i.e., our prototype system that integrates all the implementations discussed in Section 3.

4.1 Prototype System

Our prototype system is constructed using Simbad [12], the 3D robot simulator, OWL API 2.2 [13], the API library for OWL language and Pellet 2.0 [14], the DL reasoner. Fig. 5 shows the conceptual architecture of the prototype system.

Conceptually, the prototype system has three layers. The lowest layer is the Simbad environment, which can be viewed as robot hardware. Above this robot hardware layer the robot S/W framework layer is implemented. This layer is composed of three modules, the Facility module, the Behavior module, and the knowledge base module. The Facility module plays the role of mapping the robot hardware devices into the software objects. The Behavior module is implemented using the software objects in the Facility module, and plays the role of providing the high-level functions of the robot. The behaviors in the Behavior module have access to the knowledge base to refer to the changes in context (i.e., world). The knowledge base plays the role of

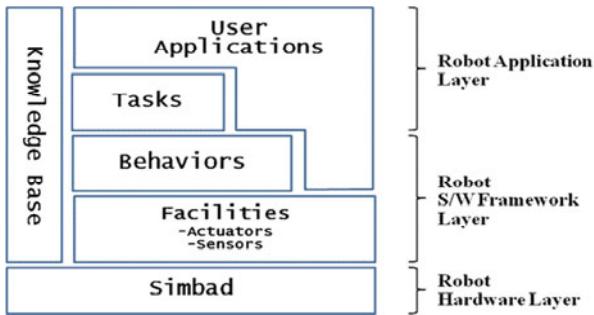


Fig. 5. Conceptual architecture of the prototype system

storing the world description and providing it to the robot applications and the behaviors. The change of the world are acquired by the sensors in the Facility module and stored in the knowledge base by the sensors. Above the software framework is the robot application layer. The robot applications are divided into two types: Tasks and the user applications. The tasks in the Task module are the applications that are performed through the automated task planning when the goals are given by the user. The Task module is implemented above the Behavior module because the task planning is conducted with the behaviors. The applications that are not performed through the task planning belong to the user application type. The user applications have their own logics for achieving the application's goal.

4.2 Experiments

We experimented with the prototype system on the scenario that a robot brings the user the ball he/she instructed. In the simulated world there are two balls, and one of the two balls, the blue ball, is located in a room with a door. For the blue ball, the task plan varies according to whether the room's door is closed or not. Fig. 6 shows in time series the robot performing the task when the door is closed.

In our experiments, the task plans are correctly generated and successfully performed by the robot even in some exceptional cases like when an unexpected object blocked the way of the robot, when the pick-up behavior failed, and when the robot has incorrect information that the door is open and eventually fails to execute the plan as it discovers that the door is actually closed. But we found some problems of performance. Although the task plan generation algorithm itself is very simple owing to the DL reasoning engine, the reasoning time rapidly increases according to the number of the behaviors, behavior's objects, and the behavior's complements. The planning for the blue-ball-bringing scenario takes about 47 minutes on IBM-compatible PC with Intel 2.0 GHz Core2 Duo CPU and 3GB RAM. At the time the knowledge base contains 7 behavior individuals, 4 behavior object individuals, and 3153 behavior complement individuals (mostly the individuals representing regions and concrete locations). And 16 SWRL rules related to the pre-conditions are specified in the KB.

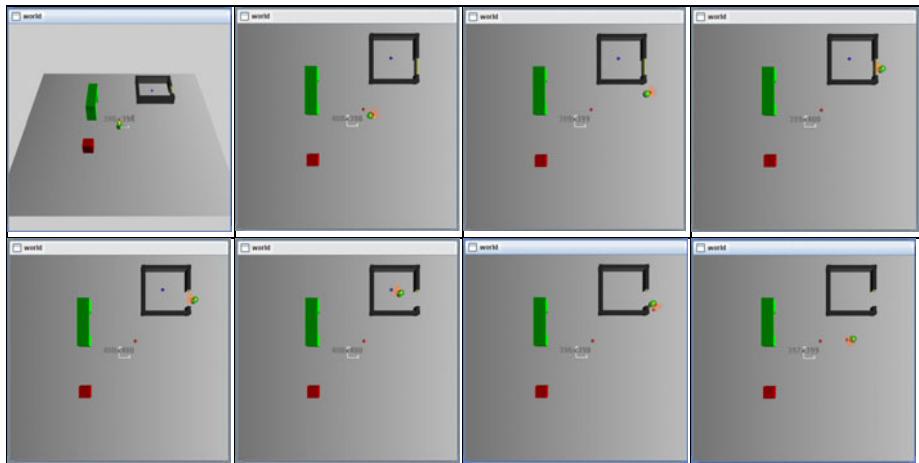


Fig. 6. A time series pictures of the robot performing the ball-bringing task when the room's door is closed

5 Conclusion and Future Works

In this paper, we have implemented the robot task planning and execution based on the DL knowledge base. For this we investigated some knowledge representation scheme and conceived simple algorithms. We also contrived some techniques to bridge the gap between the symbolic world and the program world in order to execute the task plan. We made experiments and got reasonable results in some aspects.

On the other hand, we found some problems with performance. Ideally, the backward-chaining search can improve the performance. However, representing explicitly all the causal knowledge between behaviors for backward-chaining search is not easy in any rate. Therefore we are now trying to use the knowledge of behaviors as its current form explained in this paper. Since it has some causal knowledge implicitly we might use it as the hints for the task planning system to search for the solution backward. This approach has the advantage to make the knowledge base to be simple and close to the original robot knowledge base without irrelevant extension or translation.

Acknowledgments

This research has been supported by Ministry of Knowledge Economy and Korea Research Council for Industrial Science & Technology [Grant No. 2010-ZC1140, Development of Technology of Future Robotic Computer].

References

1. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice-Hall, Englewood Cliffs (2003)
2. Stollberg, M., Rhomberg, F.: Survey on Goal-driven architecture. Technical Report: DERI-TR-2006-06-04, Digital Enterprise Research Institute, DERI (2006)

3. Galindo, C., Fernandez-Madrigal, J., Gonzalez, J., Saffiotti, A.: Robot task planning using semantic maps. *Robotics and Autonomous Systems* 56(11), 955–966 (2008)
4. Wu, X., Zeng, G., Han, F., Wang, R.: Goal representation and Reasoning based on Description Logics(DLs). In: International Conference on Intelligent Systems and Knowledge Engineering, ISKE 2007 (2007)
5. De Giacomo, G., Iocchi, L., Nardi, D., Rosati, R.: A theory and implementation of cognitive mobile robots. *J. of Logic and Computation* 9(5), 759–785 (1999)
6. Baader, F., Lutz, C., Miličić, M., Sattler, U., Wolter, F.: Integrating description logics and action formalisms: First results. In: 20th National Conference on Artificial Intelligence, AAAI 2005 (2005)
7. Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3-4), 189–208 (1971)
8. Miličić, M.: Complexity of planning in action formalisms based on description logics. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS (LNAI), vol. 4790, pp. 408–422. Springer, Heidelberg (2007)
9. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, Cambridge (2003)
10. OWL: Web Ontology Language Reference. W3C (2004),
<http://www.w3c.org/TR/owl-ref/>
11. SWRL: A Semantic Web Rule Language combining OWL and RuleML. W3C (2004),
<http://www.w3c.org/Submission/SWRL/>
12. Hugues, L., Bredeche, N., Futurs, T.I.: Simbad: an Autonomous Robot Simulation Package for Education and Research. In: Nolfi, S., et al. (eds.) SAB 2006. LNCS (LNAI), vol. 4095, pp. 831–842. Springer, Heidelberg (2006)
13. Horridge, M., Bechhofer, S.: The OWL API: A Java API for working with OWL 2 ontologies. In: OWL: Experiences and Directions, OWLED 2009 (2009)
14. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. *J. Web Semantics* 5(2), 51–53 (2007)

On the Way to High-Level Programming for Resource-Limited Embedded Systems with Golog

Alexander Ferrein¹ and Gerald Steinbauer²

¹ Robotics and Agent Research Lab, University of Cape Town, Cape Town, South Africa
alexander.ferrein@uct.ac.za

² Institute for Software Technology, Graz University of Technology, Graz, Austria
steinbauer@ist.tugraz.at

Abstract. In order to allow an autonomous robot to perform non-trivial tasks like to explore a foreign planet the robot has to have deliberative capabilities like reasoning or planning. Logic-based approaches like the programming and planing language Golog and it successors has been successfully used for such decision-making problems. A drawback of this particular programing language is that their interpreter usually are written in Prolog and run on a Prolog back-end. Such back-ends are usually not available or feasible on resource-limited robot systems. In this paper we present our ideas and first results of a re-implementation of the interpreter based on the Lua scripting language which is available on a wide range of systems including small embedded systems.

1 Introduction

In order to allow an autonomous robot to perform non-trivial tasks like to explore a foreign planet the robot has to have deliberative capabilities like reasoning or planning. There are a number of architectures in the literature like the Saphira architecture [14] or Structured Reactive Controllers [1] which enhance a robot with such advanced decision making capabilities. Other approaches, which have been investigated over the last decade, are logic-based approaches. They have been successfully applied to the control of various robot systems. One representative of this sort of decision-making approaches is the programming and planing language Golog [16]. The language is based on the Situation Calculus which allows for reasoning about actions and change. The language provides the user with some freedom for a balance between planning and imperative programming. Moreover, its syntax and semantics provides an elegant way to specify the behavior of a system.

During the years several dialects of Golog have been developed to integrate more features like dealing with uncertain knowledge, concurrency and sensing [20]. Such features are usually not provided by classical programming languages. The different Golog derivatives have in common that they are logic-based and their implementation is, so far, usually done in the logic programming language Prolog. The used Prolog back-ends like *SWI* or *Eclipse* usually have the drawback of high demands on memory and computational power. This fact and the fact that probably no Prolog system might be available at all on the target platform, prevents the application of Golog on resource-limited embedded systems. With resource-limited embedded systems we mean systems

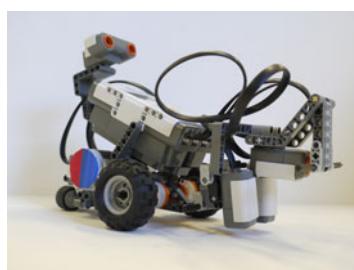
which have strongly limited resources in terms of memory and computational power. On such systems classical decision making is not able to provide decisions in time, fails to come up with decisions or the methods used are not implementable at all.

The goals of our current ongoing research is to apply such powerful decision-making approaches also to resource-limited systems. In particular we have two target robot systems in mind which we use for our research on autonomous robots. On one hand we use the humanoid robot *Nao* from Aldebaran Robotics. We use the robot for our participation in the competitions of the RoboCup Standard League [7]. *Nao* is a 21 degrees-of-freedom humanoid robot of the size of about 58 cm. It is equipped with two cameras and a AMD Geode 500 MHz CPU with 256 MB of Memory. The robot is running under embedded Linux where no Prolog interpreter is available nor feasible. The other platform is the Lego Mindstorm NXT system, which we also use for educational robotics on high-school and undergraduate level. The system is also very popular for robot competitions like RoboCupJunior or undergraduate classes in programming. The system has predefined interfaces to various sensors and actuator. The central processing block uses an ARM processor with 64 kB of RAM and 256 kB of flash-memory running at 48 MHz. There are a number of commercial and open source OS for the system. But none of them are coming with a logic back-end. The NXT robot system has also been successfully used to practically teach AI methods from standard textbooks [21][22]. But the AI methods did not run native on the robot. The related code was executed on an external computer and commands were send to the robot via blue-tooth interface.

In order to make Golog available on such platforms, we started to develop a first prototype implementation of a Golog interpreter in the scripting language Lua [5]. Lua [12] is a fast interpreted scripting language with a small memory footprint that moreover can be executed on resource-limited systems like the *Nao* robot. In the previous work we ran several standard Golog examples, e.g., an elevator controller, on our Lua interpreter as a proof of concept. In this paper, we discuss our plans to deploy IndiGolog on the Lego Mindstorm NXT platform. IndiGolog is an extension of Golog and allows on-line interpretation sensing and concurrent actions which was not the case in vanilla Golog.



(a) Humanoid robot *Nao* used in RoboCup Standard Platform League



(b) Lego Mindstorm NXT used for educational robotics.

Fig. 1. Two of our target platforms

A previous attempt to control a Lego robot with IndiGolog was done by Levesque and Pagnucco in [15]. The developed system allowed users to specify the behavior with IndiGolog, the resulting actions were transmitted to the robot via an infra-red link. Recently, the action logic C+ was deployed on a Mindstorm robot in a similar way [3].

In order to be able to use Golog-type programming languages to control autonomous robots and to directly execute it on the resource-limited system without an external Prolog back-end, we propose to port our previous idea to the two target systems. In the following sections we will explain our idea about the porting the language and the background in more details. In Section 2 we introduce Golog and our previous work on an Golog interpreter in Lua, before we show implementation details and further ideas to deploy the new interpreter on the NXT platform in Section 3. We conclude with Section 4.

2 Previous Work on Golog and Lua

In [5], we presented an implementation of the so-called vanilla Golog, this time the chosen implementation language was not Prolog, but Lua. Although only a proof of concept, it showed an alternative implementation for Golog. Especially the implementation language Lua, which was chosen in [5], is interesting, as it is available for a much larger number of systems, including embedded architectures like the Mindstorm NXT. In this section, we briefly go over the basics of the situation calculus and Golog, give a short introduction to Lua, and sketch our previous Lua implementation of Golog.

2.1 Golog

Golog is based on Reiter's variant of the Situation Calculus [17][20], a second-order language for reasoning about actions and their effects. Changes in the world are only caused by actions so that a situation is completely described by the history of actions starting in some initial situation. Properties of the world are described by fluents, which are situation-dependent predicates and functions. For each fluent the user defines a successor state axiom specifying precisely which value the fluent takes on after performing an action. These, together with precondition axioms for each action, axioms for the initial situation, foundational and unique names axioms, form a so-called basic action theory [20]. These theory models the characteristics of the environment and the capabilities of the robot. Golog emerged to an expressive language over the recent years. It has imperative control constructs such as loops, conditionals [16], and recursive procedures, but also less standard constructs like the non-deterministic choice of actions. Extensions exist for dealing with continuous change [9] and concurrency [4], allowing for exogenous and sensing actions [8] and probabilistic projections into the future [9], or decision-theoretic planning [2] which employs Markov Decision Processes (MDPs). Successful robotics application of Golog and its extensions can be found, for instance, in [6][15][19].

2.2 Lua

Lua [12] is a scripting language designed to be fast, lightweight, and embeddable into other applications. The whole binary package takes less than 200 KB of storage. When loaded, it takes only a very small amount of RAM. In an independent comparison Lua has turned out to be one of the fastest interpreted programming languages [13][23]. Besides that Lua is an elegant, easy-to-learn language [14] that should allow newcomers to start developing behaviors quickly. Another advantage of Lua is that it can interact easily with C/C++. As most basic robot software is written in C/C++, there exists an easy way to make Lua available for a particular control software. The fact that Lua is available on all systems which provide an ANSI-C compiler and a reasonable amount of memory was the main reason to choose it for our re-implementation. A re-implementation in plain-C would be also possible but would be more complex and is out of scope of our current work.

Lua is a dynamically typed language, attaching types to variable values. Eight different types are distinguished: *nil*, *boolean*, *number*, *string*, *table*, *function*, *userdata*, and *thread*. For each variable value, its type can be queried. The central data structure in Lua are tables. Table entries can be addressed by either indices, thus implementing ordinary arrays, or by string names, implementing associative arrays. Table entries can refer to other tables allowing for implementing recursive data types. For example `t["name"] = value1` stores the key-value pair (name, value1) in table t, while `t[9] = value2` stores the value2 at position 9 in array t. Special iterators allow access to associative tables and arrays. Note that both index methods can be used for the same table.

Function are first-class types in Lua and can be created at run-time, assigned to a variable, or passed as an argument, or be destroyed. Lua provides proper tail calls and closures to decrease the needed stack size for function calls. Furthermore, Lua offers a special method to modify code at run-time. With the `loadstring()` statement chunks of code (one or more instruction of Lua code is called chunk) can be executed at run-time. This comes in handy to modify code while you are running it.

2.3 The Prototype golog.lua

In [5], we presented a first prototypical implementation of a vanilla Golog interpreter in Lua. The motivation was there as it is here, to make Golog available for a larger number of platforms, as Lua in turn is available for a larger number of embedded systems. We will briefly show, how situations and programs are represented in `golog.lua`.

In Lua, everything is a table, as tables are the central and only data structure. Hence, situation terms, actions, fluents, and programs are also represented as nested tables. The following example code shows a Golog program encoded as a Lua table.

```
prog = {{ "a_1", {}}, {"a_2", {"x_1", "x_2"}},  
       {"if", {cond}, {"a_3", {}}, {"a_4", {}}}}
```

The program above consists of an 0-ary action a_1 in sequence with $a_2(x_1, x_2)$ and a conditional which, depending on the truth value of *cond*, chooses a_3 or a_4 , resp.

The program is executed with calling the interpreter function `Do1` which takes a program and a situation term, and returns the resulting situation after executing the program, or, if the program trace leads to a failure, i.e. the failure variable is true, s_2 contains the last possible action. Assuming that `cond` holds, the resulting execution trace of the `prog` will be

```
s_2 = { {"a_1", {}}, {"a_2", {"x_1", "x_2"}, {"a_3, {}}} }2
```

We use the empty table or `nil` to represent S_0 . The above situation term is to be interpreted as $do(a_3, do(a_2(x_1, x_2), do(a_1, S_0)))$. Similarly, we represent logical formulas as tables, with the connectives in prefix notation, i.e. $\{\text{and}, \phi, \{\text{or}, \psi, \theta\}\}$ represents the formula $\phi \wedge (\psi \vee \theta)$. We refer to [5] for a more concise introduction of the implementation of Golog in Lua.

3 The Next Step: indigolog.lua on the NXT

In this section, we show the next step after a vanilla Golog interpreter in Lua. We present the corner stones on the way to deploy IndiGolog on the embedded platform Mindstorm NXT. First, we introduce the library pbLua, which allows for running Lua on the NXT and provides a number of Lua functions to access the robot's sensors and actuators. Then, we outline the IndiGolog, the online Golog dialect which we have chosen for this project, and briefly discuss the differences to the fore-mentioned vanilla Golog. Finally, we show the implementation of the main loop of our new IndiGolog interpreter in Lua and introduce a target robotics domain for our approach.

3.1 pbLua

pbLua is a library from the Hempel Design Group [10] which provides a LUA interpreter for the NXT. Moreover, pbLua offers a complete API to access sensors and actuators on the NXT via Lua functions. The following examples are taken from [10].

The first example shows how to access the motor which is plugged into port 1 of the NXT:

```
-- Turn Motor 1 exactly 180 degrees at half speed
port = 1
nxt.OutputSetRegulation(port, SINGLE_MOTOR, USE_BREAK)
nxt.OutputSetSpeed(port, NO_RAMP, 50, 180)
```

The first statement sets the control option for the motor connected to port 1 (standard control for a single motor plus using the break of the motor). Then motor 1 is commanded to turn for 180° with 50% of the maximum speed not using a ramp for the acceleration/deceleration.

The second example shows Lua code that returns the values read from the light sensor until the orange button on the NXT is pressed.

¹ Note that IndiGolog, which we introduce in the next section, does not make use of `Do`, but uses a transition semantics which is defined with functions `trans` and `final`.

² Note that all program statements, actions, and fluent names must be given as strings. For reasons of readability, we omit the quotation marks throughout this paper. Note also that Lua supports to return multiple value, the situation term and the failure condition in this case.

```

-- Read light sensor until the orange button is pressed
function LightRead(port,active)
    active = active or 0
    if 0 == active then nxt.InputSetType(port,LS_WI,SV)
    else nxt.InputSetType(port,LS_WOI,SV)
5   end
    repeat
        print( nxt.TimerRead(), nxt.InputGetStatus(port) )
        until( ORANGE_BUTTON == nxt.ButtonRead() )
10 end

```

First the mode of the sensor on port 1 is configured. The different flags are: LS_WI (light sensor using own illumination), LS_WOI (light sensor using ambient illumination) and SV (provide scaled values in the range of 0 to 100%). Then read and display the value of the sensor until the orange button is pressed.

The pbLua API offers a complete interface to the sensors and actuators of the NXT and is therefore well-suited for our task.

3.2 IndiGolog

The major drawbacks of vanilla Golog [16] is that it does not allow for sensing and can only be interpreted in an offline fashion. These facts obviously limit the value of Golog for the control of autonomous robots which are deployed in a dynamic and uncertain world. In order to overcome this limitations several successors to Golog have been developed. One of them is IndiGolog (incremental deterministic Golog) [8] which allows among others for online interpretation, sensing actions and concurrent actions. We decide to use this Golog dialect for our embedded reasoning system as it provides all features necessary for intelligent robust robot control.

IndiGolog also makes use of a situation calculus basic action theory. The way how IndiGolog interprets programs, is however different. IndiGolog uses a transition semantics. That means that the input program is interpreted in a step-by-step fashion. The interpreter directly commits to actions, i.e. once a primitive action is interpreted and it is possible to execute it, it will be executed in the real world. This is the main difference to the so-called evaluation semantics of vanilla Golog, where the program is first evaluated until the end. In IndiGolog, the program is transformed from one configuration $\langle \sigma, s \rangle$ to a legal successor configuration $\langle \delta, s' \rangle$ by means of a predicate $Trans(\sigma, s, \delta, s')$. Termination conditions of the program are defined with a predicate $Final(\sigma, s)$, which states when the program σ may legally terminate. The semantics of the program statements are hence defined by predicates $Trans$ and $Final$. To give an example, we show the example of the $Trans$ and $Final$ predicates of a primitive action and a loop.

$$\begin{aligned}
Trans(\alpha, s, \delta, s') &\equiv Poss(a[s], s) \wedge \delta = nil \wedge s' = do(a, s) \\
Final(\alpha, s) &\equiv false \\
Trans(\text{while } \varphi \text{ do } \delta_1 \text{ end}, s, \delta, s') &\equiv \\
&\varphi[s] \wedge \exists \delta'. \delta = (\delta'; \text{while } \varphi \text{ do } \delta_1 \text{ end}) \wedge Trans(\delta, s, \delta', s') \\
Final(\text{while } \varphi \text{ do } \sigma \text{ end}, s) &\equiv \varphi[s] \wedge Final(\sigma, s)
\end{aligned}$$

If the primitive action is possible, i.e. its precondition axiom $Poss(a[s], s)$ holds, the action is executed. The successor situation is the one where the action was executed,

i.e. $s' = do(a, s)$, and the successor configuration is $\delta = nil$. The final configuration is obviously not reached, as the program is transformed to the *nil*-program, which in turn defines the final configuration. The loop works as follows. If the condition holds, the successor configuration consists the loop body in sequence with the while loop itself to ensure that another iteration of the loop can be executed in the next transition step. The loop reaches a legal final configuration, when the condition holds and the loop body terminates.

The second advantage of IndiGolog over vanilla Golog is that it has a direct account to sensing. The basic action theory is extended with so-called sensing actions and exogenous actions. These are actions that directly connect a robot's sensor with a fluent. Each time, the sensing action is executed, the program gets an update of the respective sensing result. Similarly, exogenous actions are very useful to model interaction between the robot and its environment.

3.3 Some Implementation Details

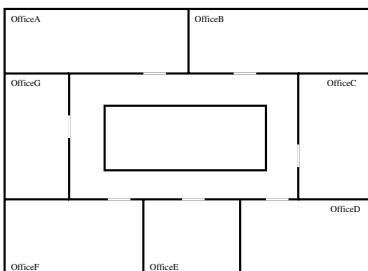
In the following, we give some implementation details of our IndiGolog implementation in Lua. Basically, we present the interpreter main loop, which shows the respective calls to functions `trans()` and `final()`, which in turn interpret the input program `p` (which we omit here). In line 8 of the function `indigo()` it is checked, whether a transition in the program `p` is possible in the situation `s`. The function `trans:check(p, s)` extracts the first program instruction of `p` and returns the transformed program (as defined in the previous section) in the member variable `trans.delta` (cf. line 21). If `trans:check()` returns `false`, it is checked if a final configuration was reached (line 13), otherwise, the program execution was unsuccessful (line 17).

```

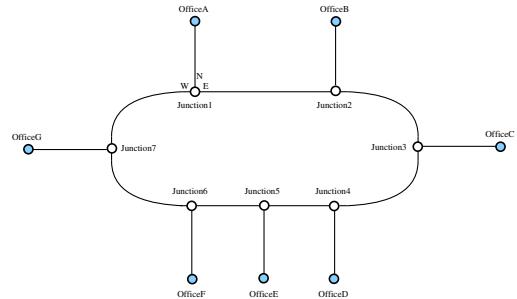
function indigo(p, s)
    trans.s = s
    local tv -- did final/trans evaluate to true?
    repeat
        -- did an exogenous action occur?
        if exog_occurs then exog_action() end
        -- is a transition possible?
        tv = trans:check(p, s)
        -- check for termination condition otherwise
        if not tv then
            if final:check(p, s) then
                print("Program terminates successfully\n")
                return
            else --
                print("Program terminates unsuccessfully\n")
                return
            end
        end
        p = trans.delta -- remainder program
        s = trans.s -- new situation term
    until false
end

```

The functions `indigolog` and `trans` were implemented as co-routines. Moreover, the search operator for non-deterministic choices had been implemented using co-routines. As in Prolog-based implementations the actions are not executed in the real world during the search. Similar as in [5], the basic action theory has



(a) Office Environment.



(b) Topological map of the example domain.

Fig. 2. Example domain for the delivery robot

to be specified. Each action is stored as an object with its own metatable, i.e. `act = action:new(name, arg1, ...)`. Then, one has to specify the pre-condition axiom in form of a function `act.Poss(s,arg1, ...)`. Similarly, a function `act.Execute(arg1, ...)` is required. This defines the interface to the real world and can be used to execute, say, drive actions on the real robot hardware. For each fluent occurring in basic action theory, one has to specify an effect axiom of the form `fluent.action(s, arg1, ...)` together with a function `fluent.initially(arg1, ...)` which defines the initial value of the respective fluent. When a fluent formula is evaluated, fluent regression is applied by subsequently calling the particular effect axioms. Assume we have a fluent f . Moreover, we have two primitive actions a_1 and a_2 which are executed in sequence starting in S_0 leading to a new situation S . In our lua-based implementation the value of the fluent f in situation S is evaluated by the recursive function call `f.a2({a1}, f.a1({}), f.initially({}))`.

Note that we assume a closed world here. More details about this can be found in [5].

3.4 The Target Domain

We plan to use the Lua-based implementation of IndiGolog to solve the following delivery robot domain which is an extension of the original example used in [15]. The robot has to deliver goods between offices. The example domain is shown in Figure 2(a).

The domain comprises several offices connected by a corridor. In order to ease the navigation for the Lego Mindstorm NXT based robot in reality there is a black line on the floor connecting the rooms. The robot can simply follow this line. The line may branch on some junctions. In order to allow the robot to detect if it has reached a room or junction there are RFID tags on the floor. Please note that branches at junctions are always orthogonal and marked with their direction, e.g., N for the northbound branch. Junctions are always located in front of an office door. Connection lines between rooms and junctions itself need not to be straight. The robot uses four standard Lego Mindstorm NXT sensors: (1) a light sensor for following the black line, (2) a light sensor to

detect the silver spots, (3) a touch sensor to detect an obstacle and (4) a touch sensor to detect if a package is currently loaded.

The robot uses a graph-based topological map to find its way between the office. The corresponding graph for the example domain is depicted in Figure 2(b). The robot receives delivery orders to bring goods from one office to another office. In order to be able to fulfill its task the robot needs some planning capabilities in order to decide which order to process next and to plan its path from the origin to the destination. Moreover, paths between junctions and office may be blocked by one or more obstacles. If a path is blocked is initially not known to the robot. Therefore, the robot has to deal with incomplete knowledge and it needs also sensing and re-planning capabilities.

IndiGolog is obviously suitable to control the robot for this task. In order to solve the task with we have to specify fluents, actions, the influence of sensing and domain-dependent facts.

For instance we need the following fluents among others:

- *at(loc)*: this fluent is true if the robot is at the corresponding junction or room *loc*
- *direction(dir)*: the robot faces towards a particular direction *dir* at a junction
- *loaded(package)*: the robot carries *package*
- *blocked(node₁, dir₁, node₂, dir₂)*: the path between the nodes *node₁* and *node₂* via the directions *dir₁* and *dir₂* is blocked

Moreover, we need domain-dependent facts:

- *connect(node₁, dir₁, node₂, dir₂)*: the nodes *node₁* and *node₂* are connected via the directions *dir₁* and *dir₂*
- *order(origin, destination, package)*: there exists a delivery order for *package* from *origin* to *destination*

In order to serve the different delivery orders we define the following control procedure:

```
control = procedure:new('control', {})
control.body = {{'while', {'some', {o,d,p}}, {'order', {o,d,p}}},
               {'serve_order', {o,d,p}}}}
```

The procedure *control* continues to chose non-deterministically an order and process it until no open orders exist anymore. We use existentially quantified statement *some*. This leads us to the procedure *serve_order*:

```
serve_order = procedure:new('serve_order', {o, d, p})
serve_order.body = { {'goto', {o}}, {'pickup', {p}},
                    {'goto', {d}}, {'deliver', {p}}}}
```

serve_order execute sequentially the actions go to origin *o*, picks up the package *p*, go to destination *d* and finally deliver the package *p*. Where the procedure *goto(g)* implements the navigation towards the goal *g*.

```
goto = procedure:new('goto', g)
goto.body = { {'while', { {'not', {'at', g}}}, {'? ', {'some', {d}, {'next_dir', d}}}, {'turn', {d}}, {'drive'}}}}
```

As long as the final goal g is not reached, expressed by the formula $\neg at(g)$ as condition, the procedure chooses the next suitable direction on the junction, turns toward the corresponding branch and moves along the branch until it reaches the next node. Please note the predicate $next_dir(d)$ is true for the next suitable direction towards the goal.

This leads us to the basic primitive actions. For instance the primitive action $turn(d)$ ensures that the robot faces towards the direction d , $d \in \{N, E, S, W\}$. The primitive actions actually control the robot's hardware via the corresponding pbLua commands. Please note that for simplicity the directions are represented internally by the set of integers $\{0, 1, 2, 3\}$.

```

turn = action:new('turn', 1) -- turn action has arity 1
function turn.Poss(s,dir)
    return true
end
5 function turn.Execute(s,dir)
    local act_dir = GetFluent('direction')
    local dir_diff = dir - act_dir
    if dir_diff < 0 then dir_diff = 4 + dir_diff end
    for i=dir_diff, i>0 do
10    repeat
        nxt.OutputSetSpeed(MOTOR_A,NO_RAMP,50)
        nxt.OutputSetSpeed(MOTOR_B,NO_RAMP,-50)
        until (nxt.InputGetStatus(SENSOR_1) < BLACK_LIMIT)
        nxt.OutputSetSpeed(MOTOR_A,NO_RAMP,0)
15        nxt.OutputSetSpeed(MOTOR_B,NO_RAMP,0)
    end
end

```

The primitive action $turn$ can be executed all time and has therefore a precondition which is always true.

The executable part of the action first calculate how many orthogonal turn it has to perform to reach the desired direction. Once it knows the number of turns it simple repeats to turn on the spot until the line sensor detects the next orthogonal branch. This fact is detected if the returned value of the light sensor falls below the threshold `BLACK_LIMIT`.

The other remaining primitive action are defined in a similar way.

4 Conclusion and Future Work

In this paper, we showed some ongoing work to make the robot programming and plan language Golog and its derivatives available for a larger number of robot platforms. Usually, Golog interpreter are implemented in Prolog, which is straight-forward regarding the fact that the specification of Golog is given in first order logic. However, Prolog might not be available for all robot platforms. Therefore in [5], a first prototypical Lua implementation of a vanilla Golog interpreter was presented. As vanilla Golog is only of limited usability for real robots, we here show the next steps to apply Golog-type languages to real robots. We started to re-implemented IndiGolog in Lua with the goal to deploy it on the platform Lego Mindstorm NXT. IndiGolog features a online execution semantics and is able to integrate sensor actions and exogenous events, which is particularly useful for real robot applications. As this is ongoing research, we are only able to show the preliminary implementation of IndiGolog on the Lua interpreter which runs the Linux-based OS of the Nao robot. This is a proof of concept and shows that

an implementation of an IndiGolog interpreter is possible without a Prolog back-end. Moreover, we were easily able to connect the interpreter to the behavior execution on the Nao as we already use a Lua-based engine [18].

Currently, we are working towards the actual native deployment of the interpreter under pbLua on the Lego Mindstorm NXT. Due to the very limited memory on the NXT, one challenge is to fit the interpreter, the domain description and the IndiGolog program onto the NXT. One possible way to do this could be to set up a cross-compile chain for the ARM CPU and just deploy the Lua byte-code on the NXT. This is due to further investigation and needs also close cooperation with the developer of the pbLua suite as such a feature is currently not available. Moreover, we still need an external PC to download the Lua code each time we start the robot. But we are convinced to have first results available in the near future. The future work also includes the full integration of sensing, the investigation of the correctness of the implementation as well as giving run-time results of the performance of the interpreter, both on the Mindstorm NXT as well as on the humanoid robot Nao.

In particular the Nao application is interesting. In previous work the Golog dialect ReadyLog [6] was used to control soccer robots of the RoboCup Middle Size League. The complexity of the RoboCup Standard Platform League is comparable to that domain which was modeled using about 100 fluents, 40 primitive actions and additional 15000 line of code for interfacing the robot hardware. How to fit such a control model into a limited system such as the Nao will be a real challenge.

Acknowledgments

A. Ferrein is currently a Feodor Lynen fellow supported by a grant of the Alexander von Humboldt Foundation. G. Steinbauer was partly funded by the Österreichische Forschungsgemeinschaft. We thank the anonymous reviewers for their helpful comments.

References

1. Beetz, M.: Structured Reactive Controllers. *Autonomous Agents and Multi-Agent Systems* 4(2), 25–55 (2001)
2. Boutilier, C., Reiter, R., Soutchanski, M., Thrun, S.: Decision-theoretic, high-level agent programming in the situation calculus. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000) and Twelfth Conference on Innovative Applications of Artificial Intelligence (IAAI 2000), pp. 355–362. AAAI Press, Menlo Park (2000)
3. Caldiran, O., Haspalamutgil, K., Ok, A., Palaz, C., Erdem, E., Patoglu, V.: Bridging the gap between high-level reasoning and low-level control. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, pp. 342–354. Springer, Heidelberg (2009)
4. De Giacomo, G., Léspérance, Y., Levesque, H.: ConGolog, A concurrent programming language based on situation calculus. *Artificial Intelligence* 121(1-2), 109–169 (2000)
5. Ferrein, A.: lua.golog: Towards a non-prolog implementation of golog for embedded systems. In: Lakemeyer, G., Levesque, H., Pirri, F. (eds.) Cognitive Robotics. Dagstuhl Seminar Proceedings, vol. 100081, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (to appear, 2010)

6. Ferrein, A., Lakemeyer, G.: Logic-based robot control in highly dynamic domains. *Robotics and Autonomous Systems, Special Issue on Semantic Knowledge in Robotics* 56(11), 980–991 (2008)
7. Ferrein, A., Steinbauer, G., McPhillips, G., Potgieter, A.: RoboCup Standard Platform League - Team Zadeat - An Intercontinental Research Effort. In: International RoboCup Symposium, Suzhou, China (2008)
8. Giacomo, G.D., Lésprance, Y., Levesque, H.J., Sardina, S.: Multi-Agent Programming: Languages, Tools and Applications. In: *Multi-Agent Programming: Languages, Tools and Applications*, pp. 31–72. Springer, Heidelberg (2009)
9. Grosskreutz, H., Lakemeyer, G.: ccgolog – A logical language dealing with continuous change. *Logic Journal of the IGPL* 11(2), 179–221 (2003)
10. Hempel, R.: pblua – scripting for the LEGO NXT (2010),
<http://www.hempeldesigngroup.com/lego/pblua/> (last visited on May 21, 2010)
11. Hirschi, A.: Traveling Light, the Lua Way. *IEEE Software* 24(5), 31–38 (2007)
12. Ierusalimschy, R., de Figueiredo, L.H., Filho, W.C.: Lua - An Extensible Extension Language. *Software: Practice and Experience* 26(6), 635–652 (1999)
13. Ierusalimschy, R., de Figueiredo, L.H., Filho, W.C.: The Evolution of Lua. In: *Proceedings of History of Programming Languages III*, pp. 2–1 – 2–26. ACM, New York (2007)
14. Konolige, K., Myers, K., Ruspini, E., Saffiotti, A.: The saphira architecture: A design for autonomy. *Journal of Experimental and Theoretical Artificial Intelligence* 9, 215–235 (1997)
15. Levesque, H.J., Pagnucco, M.: Legolog: Inexpensive experiments in cognitive robotics. In: *Proceedings of the Second International Cognitive Robotics Workshop*, Berlin, Germany (2000)
16. Levesque, H.J., Reiter, R., Lésprance, Y., Lin, F., Scherl, R.B.: GOLOG: A logic programming language for dynamic domains. *The Journal of Logic Programming* 31(1-3), 59–83 (1997),
<http://www.sciencedirect.com/science/article/B6V0J-3SNV3TD-4/2/11b8202180bb72d91149646db7f4979e>, reasoning about Action and Change
17. McCarthy, J.: Situations, Actions and Causal Laws. Tech. rep., Stanford University (1963)
18. Niemüller, T., Ferrein, A., Lakemeyer, G.: A lua-based behavior engine for controlling the humanoid robot nao. In: Baltes, J., et al. (eds.) *RoboCup 2009. LNCS*, vol. 5949, pp. 240–251. Springer, Heidelberg (2010)
19. Pham, H.: Applying DTGolog to Large-scale Domains. Master's thesis, Department of Electrical and Computer Engineering, Ryerson University, Toronto, Canada (2006)
20. Reiter, R.: *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press, Cambridge (2001)
21. Russell, S., Norvig, P.: *Artificial Intelligence: A Modern Approach*, 2nd edn. Prentice-Hall, Englewood Cliffs (2003)
22. Talaga, P., Oh, J.C.: Combining AIMA and LEGO mindstorms in an artificial intelligence course to build realworldrobots. *Journal of Computing Science in Colleges* 24(3), 56–64 (2009)
23. The Debian Project: The Computer Language Benchmarks Game,
<http://shootout.alioth.debian.org/> (retrieved January 30, 2009)

RobustHX - The Robust Middleware Library for Hexor Robots

Konrad Kułakowski and Piotr Matyjasik

Institute of Automatics,
AGH University of Science and Technology
Al. Mickiewicza 30,
30-059 Cracow, Poland
{kkulak,ptm}@agh.edu.pl

Abstract. This paper presents *RobustHX* - an extension of the *Robust* middleware library, which allows the six-legged *Hexor* robot to be programmed in *Java*. Like *Lego Mindstorms NXT* robots, *Hexor* has been adopted by many academic centers in Poland as a tool for teaching AI robotics. For this reason, the authors hope that *RobustHX* may also be found as an interesting teaching aid allowing basic robotic experiments to be carried out.

1 Introduction

In recent years, increased interest in the design of intelligent mobile robots has been visible. Although building such constructions from scratch requires knowledge of many different disciplines, such as mechanics, electronics or computer science, there are now many simple robotic hardware platforms ready to be programmed and used. One such robotic construction is *Hexor* - a six-legged didactic mobile robot [22][15] provided by the Stenzel¹ company. The low-level C based *API* allows the whole construction to be moved, stimuli coming from the environment to be detected and an installed camera to be controlled. Although the *Hexor* hardware platform can be programmed in other languages, such as *Assembler* or *Basic*, the main absentee from this group is *Java* - a very popular object-oriented high-level programming language. In order to fill this gap, the *RobustHX* extension has been developed. Following the design of the *Robust* library [11], *RobustHX* aims to provide a simple and handy API to control the robot. Thanks to the existence of *cljRobust API*, simple control algorithms for *Hexor* can also be written in *Clojure* - a new, efficient *Lisp* dialect running on a *Java Virtual Machine* [8].

In order to show where the proposed solution fits, the introductory section of the paper contains a short outline of robotic software, including remarks concerning *Java* and *Lisp* usage in AI robot programming. Section 2 briefly summarizes the *Robust* library, together with its *Clojure* extension. Section 3 contains the proposal of an *API* for *Hexor*. Section 4 presents a simple control

¹ <http://www.stenzel.com.pl>

algorithm allowing the robot to move and sense. Finally, Section 5 includes a summary and the plan for future research and library development.

1.1 Intelligent Robotic Software

When discussing the first AI Robotics constructions, we may consider HILARE - a mobile robot constructed in LAAS (*Laboratoire d'Architecture et d'Analyse des Systèmes*) [7] and *SHAKY* the robot constructed by Nilsson [19] at *Stanford Research Institute*. Since then many interesting AI robotic constructions have appeared, including 4D/RCS [2] or *Saphira* [10]. Many of them entailed the creation of valuable software, which might be (frequently free of charge) used to build customised robotic solutions. One of the best known robotic constructions, 4D/RCS [2], includes a freely available *RCS Library*. This contains a bundle of tools and documents that helps in the constructions of its own mobile unmanned solution. Another interesting software framework is *CARMEN* (*The Carnegie Mellon Navigation Toolkit*) [16]. This is an open source library of robot control software designed to provide a consistent interface as well as a basic set of primitives suitable for a wide range of robots. Although well designed and modular, the framework is perceived as unsuitable for largely indoor vehicles [5]. Another robotic middleware library is *Orca* [4]. This provides the programmer with an environment enabling the definition and development of “building-blocks”, which can later be gathered together in the form of a robotic system. Some of the blocks, such as path planning libraries or drivers (e.g. *Garmin-GPS15L* module) are ready-to-use and are being provided together with the *Orca* framework. Although *Orca* is implemented in C++, building new framework elements in Java is also possible. The object-oriented framework for mobile robot control is *MIRO* (*Middleware for RObots*) [24]. The overall architecture of the framework is formed into three layers: device layer, communication layer (strongly tied to the *CORBA* architecture) and *Class* layer containing a number of commonly-used functional modules, like path planning, localisation and mapping etc. A more extensive review of intelligent robotic software can be found at [5][13].

Although Java is not suitable for such low-level programming as C++, it is also readily chosen for the construction of robots [20][3]. An interesting example of a popular middleware library, which can be used to control robots is *JESS* (*Java Expert System Shell*). *JESS* is successfully applied as an inference engine in the task manager of *CAMUS* (*Context-Aware Middleware for URC System*) [9]. As a “traditional” language of artificial intelligence, *Lisp* is also frequently used in AI robotic projects. As an example, we may consider projects such as *Saphira* [10][18] (*CL-SAPHIRA* is a *Common Lisp API* for the *Saphira* system), *Humanoid Programming System* [23][14] or *Autonomous Control System - Remote Agent* [21]. *Lisp* is also present on the *Lego Mindstorms NXT* educational platform [1].

The *RobustHX* component is an extension of the *Robust* library [11] allowing the *Hexor* robot to be controlled. *RobustHX* tries to use as much of the original *Robust* API as possible, extending it where appropriate. Since *Robust* was originally designed for *Lego Mindstorms NXT* [6], it is possible to reuse substantial parts of the code written for *Lego* robots to control the *Hexor* platform. Thanks

to *cljRobust*, which is the *Lisp* overlay for the *Robust API*, *Hexor* can also be programmed in *Clojure* [12]. Due to all these reasons, the authors hope the *RobustHX* will be an interesting teaching aid allowing students to carry out their experiments on the *Hexor's* advanced robotic platform.

2 The Robust Library Components

The main reason for building the *Robust* library [11] was the need to provide students with an easy to use programming environment for *Mindstorms NXT*, which allows them to use the standard Java distribution run on a PC. The whole library is composed of four components, *RobustNXT*, *RobustHX*, *RobustPC*, and *cljRobust* (Fig. 1).

The first two modules are hardware platform based back-ends. The *RobustNXT* component is responsible for communication and the low-level control of *Lego Mindstorms NXT*, and *RobustHX* is responsible for the low-level control of *Hexor*. The *RobustPC* component tends to be hardware platform independent. It consists of several APIs corresponding to different functional parts of the robot. The *RobustPC* module is a *Java* front-end of the *Robust* library and is designed

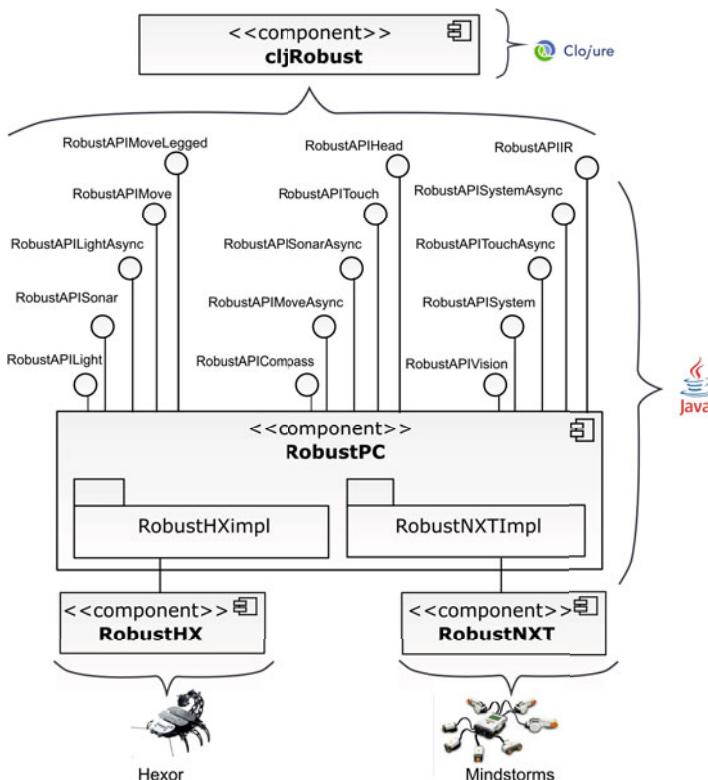


Fig. 1. General Robust Architecture

to be used under the control of a standard *JVM* (*Java Virtual Machine*). The *cljRobust* module is a *Clojure* API layer over the *Java API* [12]. It allows control applications to be developed in *Clojure* - a modern dialect of *Lisp*.

2.1 Robust on Lego Mindstorms NXT

The *Robust* library was released to the public² in the middle of 2009 with support only for *Lego Mindstorms NXT* hardware platform. Lego implementation part of *Robust* consists of two software components, *RobustNXT* and *RobustNXTImpl*. The first module is a hardware dependent application run under the control of *LeJOS* [17] - a *Java* based firmware replacement for *Lego Mindstorms NXT*. The second one is a *J2SE* based implementation of *Robust*'s APIs run on a PC. Both components communicate with each other via *BT* (*Bluetooth*).

The *RobustNXT* routines can be grouped into four modules; *Communication*, *Effectors*, *Sensors* and *Utils* (Fig. 2).

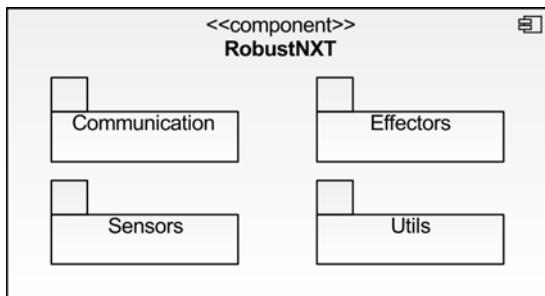


Fig. 2. Structure of *RobustNXT*

Effectors and *Sensors* are responsible for low-level control of *Mindstorms NXT* peripheries, such as servomechanisms, touch sensor, light sensor and ultrasonic sensor etc. *Communication* and *Utils* contain auxiliary routines facilitating *BT* message passing between *NXT* and the *PC* as well as remote logging. More detailed descriptions of the *NXT* related part of the *Robust* library can be found in [11].

2.2 cljRobust for Lisp Programmers

The *cljRobust* is the functional *API* for the *Robust* library. It wraps several *RobustPC* interfaces into the *Clojure* code so that writing a control application in a functional manner is also possible. The *Clojure* API opens the *Robust* library to the *Lisp* language. Thus, programmers familiar with *Lisp* are able to develop high-level control algorithms for any of the *Robust* hardware platforms. The *Lisp* language is also important in the teaching of artificial intelligence at universities. Since *cljRobust* supports *Clojure* (which is de facto the *Lisp* language) it is possible to smoothly incorporate robotic classes into *AI* course. *cljRobust* consists of several function groups corresponding to different *RobustPC* interfaces. The

² <http://sourceforge.net/projects/robust/>

most important implemented ones are: *RobustAPISystem*, *RobustAPIMove* and *RobustAPISonar*. All the implemented interfaces of the *cljRobust* were extensively discussed in [12].

3 RobustHX API

The *Robust* library was designed with *Lego Mindstorms NXT* wheel based constructions in mind. Since *Hexor* is a legged robot, it was not so easy to reuse the original *Robust* interfaces and, finally, a new API extension was introduced. The *RobustHX API*³ was implemented and tested on the *Hexor II* mobile platform.

Hexor II is an autonomous 6-legged intelligent robot device. It was developed by the *Stenzel* company as a didactic platform [22]. It has a modular construction and can easily be extended by additional components, such as a compass, a laser range-finder, and sensors, etc. The company provides the hardware and a simple development software environment. *Hexor II* is a metal-based construction. It looks and moves like a scorpion. The robot is moved by three servos: one for tilt, and two for forward and backward leg movement. Such a mechanical construction allows the *Hexor II* to perform fast insect-like movement algorithms [22]. Two other servos control a camera which is fitted on the top of the movable “scorpion tail”, providing a wide scanning area. The sonar sensor, placed together with the camera, can effectively measure the distance to objects between 3 and 300 cm from the sensor. *Hexor II* is also fitted with four infra-red sensors; three mounted at the back, and one mounted at the front, where two touch sensors (tentacles) are also mounted. The robot controller consists of two parts connected via a serial link:

- microcontroller board, which executes a low-level tasks, such as servo PWM generation, sensor scanning, etc.,
- microATX PC board, which executes higher level algorithms that implement intelligent behavior.

Due to a different mechanical construction (see Fig. 3) it was unable to implement all the APIs previously designed for wheeled devices. Thus, to cover the presented robot type, an additional interface *RobustAPIMoveLegged* was introduced. This interface API provides the following procedures:

```
moveForward(int keepAzimuth);
moveBackward(int keepAzimuth);
turnLeft();
turnRight();
moveForward(int steps, int keepAzimuth);
moveBackward(int steps, int keepAzimuth);
turnLeft(int steps);
turnRight(int steps);
stop();
```

³ Scheduled for the 3th quarter of 2010, <http://sourceforge.net/projects/robust/>

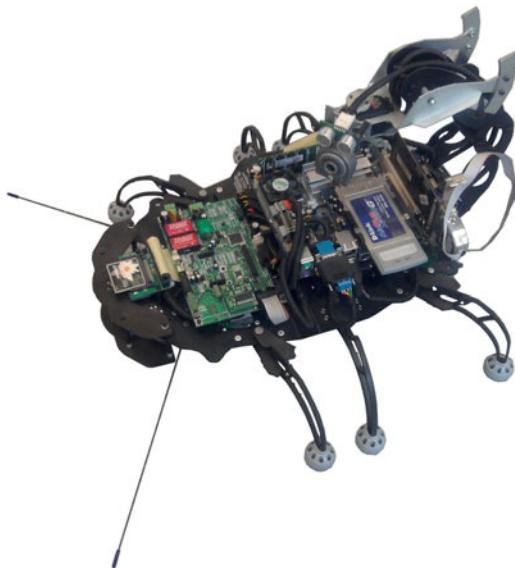


Fig. 3. Hexor II

The first four methods mentioned above, provide the functionality for instant changing of the robot's movement. The robot moves in a specified direction until another movement related command is executed. The next four API commands allow the number of steps that robot need to perform to be specified. After performing the desired amount of steps, the robot stops. Similarly, as in *RobustAPIMoveAsync* , this API is designed for queuing movement commands provided by the *RobustAPIMoveLeggedAsync* interface. The *keepAzimuth* parameter controls if the movement direction is supported by additional sensors (gyroscope, compass, etc.).

Another new interface *RobustAPIHead* was introduced to provide control over the robot's head (a tail tip), where the sonar and camera are placed. This interface allows the horizontal and vertical position of the head to be set. It consists of the following two methods:

```
horizontalHeadPosition(int hPosition);
verticalHeadPosition(int hPosition);
```

The *RobustAPIIR* provides an interface for *Hexor*'s infra-red sensors. It returns the current readings as an integer vector. Both the head and IR interfaces include asynchronous versions. The touch and compass interfaces are common for *Lego* and *Hexor* constructions, together with basic system functions, thus no additional interfaces were necessary.

4 Sample Algorithm

The following three listings present, the simple exemplary algorithm built on the basis of the previously presented APIs. This algorithm starts from applying the

sonar sensor to search for the longest possible obstacle-free path. Then, after choosing one, the robot is forced to follow the selected direction. Due to code size, some lines of code irrelevant when comprehending the algorithm are omitted from the listings.

```

1 package robust.pc.examples;
2 public class HXMoveEx {
3     private RobustAPISystem system;
4     // more private references follow
5     public void runExample() {
6         final RobustFactory factory = RobustFactory.getInstance();
7         if (!factory.isAPISupported(RobustAPISystem.class)
8             // additional required API providers checked
9             system = factory.getRobustCommandsSystem();
10            move = factory.getRobustCommandsMoveLegged();
11            systemAsync = factory.getRobustCommandsSystemAsync();
12            touchAsync = factory.getRobustCommandsTouchAsync();
13            irAsync = factory.getRobustCommandsIRAsync();
14            sonar = factory.getRobustCommandsSonar();
15            systemAsync.registerLogEvent(
16                new RobustAPISystemAsync.BodyLoggerHandler() {
17                    public void handleLogEvent(String msg) {
18                        Logger.getLogger("Body").info("BODY: " + msg);}});
19            system.startup();
20            irAsync.registerIRListener(new IRLISTENER() {
21                public void handleIREvent(int[] irStates) {
22                    move.stopNow(); obstacle = true;}});
23            touchAsync.registerTouchListener(new TouchListener() {
24                public void handleTouchEvent(int[] touchStates) {
25                    move.stopNow(); obstacle = true;}});

```

Listing 1. The move algorithm, method runExample, part I (initialization)

In general the whole algorithm consists of two functions: the *runExample* (listings: ① and ②) and the *searchDirection* (listing: ③). The *runExample* procedure is the main function of the program. It prepares all the required interfaces (lines: 6-14), sets up the logging facility (lines 15-18) and registers listeners for the infrared (lines 20-22) and touch sensors (lines 23-25). Those asynchronous callbacks interrupts movement execution if the robot enters an obstacle. The lines 28-32 contain the main program loop. Inside the loop, by execution of the *searchDirection* the program searches for the best direction to move along. The *searchDirection* method returns the number of left or right steps according to the readings of the sonar. Then, an appropriate turn is executed (lines: 30-31). Finally, in the line 32 the robot performs five steps ahead. This simple scenario is repeated 100 times or until the touch or IR sensors detect an obstacle.

```

26     int steps = 100;
27     int s;
28     while (!obstacle && steps > 0) {
29         steps--; s = searchDirection();
30         if (s > 0) move.turnRight(s);
31         else if (s < 0) move.turnLeft(Math.abs(s));
32         move.moveForward(5, 0); }
33     system.shutdown();
34     System.exit(0);
35 } // RunExample

```

Listing 2. The move algorithm, method runExample, part II (execution)

The *search Direction* method scans the robot's surroundings to find the highest sonar measurement. According to the reading, the appropriate number of left or right steps to perform is returned. The *steps* array (line: 37) contains the number of steps to execute and it is closely coupled with the *sonarPos* array (line: 38), which contains the related sonar positions. In the loop (lines: 40-45) there is a search for the maximal sonar reading and the best direction is stored in the *maxIndex* variable.

```

36     public int searchDirection() {
37         int[] steps = { -3, -2, -1, 0, 1, 2, 3 };
38         int[] sonarPos = { 0, 18, 36, 50, 68, 86, 100 };
39         int x; int maxReading = 0; int maxIndex = -1;
40         for (x = 0; x < steps.length; x++) {
41             sonar.setSonarTarget(sonarPos[x]);
42             int sr = sonar.getDistance();
43             if (sr > maxReading) {
44                 maxIndex = x;
45                 maxReading = sr; }
46         return steps[maxIndex]; } // searchDirection
47 } // HXMoveEx

```

Listing 3: The move algorithm, method searchDirection

During movements of the whole construction, the asynchronous callbacks watch for the obstacles close by. In case of entering one, the search is interrupted and the program is stopped.

Although, the presented algorithm is very simple, the robot acts as if it is executing a sophisticated search strategy. This is due to imperfections in sensor measurements, move vagueness and leg slips.

5 Summary and Future Work

In this paper, the new *RobustHX* extension of the *Robust* library has been presented. It brings several new *APIs* (e.g. *RobustAPIMoveLegged*) specific to the *Hexor* six-legged platform, and redefines existing ones e.g. *SonarAPI*. The *Hexor* robot is a more advanced construction than Lego Mindstorms NXT. In particular, it has much more computational power, so that the whole *Robust* environment, including *RobustPC* and *cljRobust* can be installed directly on the robot. This increases the reliability of the construction and makes it a truly independent, autonomous mobile robot.

Although *Robust APIs* are ready to use, a lot of aspects still need to be improved. Since, initially, the *Robust* library was designed for *Mindstorms NXT* vehicles only, some APIs should be redesigned in a more general manner. Some other robotic platforms (e.g. *Pioneer*⁴ 3-AT) also are considered as hardware implementation layers. Virtual hardware running on several different robotic simulators, such as *Webots*⁵ or *Microsoft Robotics Studio*⁶ are also taken into account. Further development of the *Robust* library aims to create a deliberative control layer on the basis of the existing *APIs*. Such an intelligent control system would integrate different kinds of knowledge, which would then be able to cooperate with different robotic constructions.

Acknowledgement

This research is supported by AGH University of Science and Technology, contract no.: 11.11.120.859.

References

1. Lego Mindstorms User Guide. The Lego Group (2006)
2. Albus, J.S., et al.: 4D/RCS: A Reference Model Architecture For Unmanned Vehicle Systems Version 2.0 (2002)
3. Bagnall, B.: Maximum LEGO NXT: Building Robots with Java Brains. Varian Press (2009)
4. Brooks, A., et al.: Towards component-based robotics. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005) (2005)
5. Broten, G., Monckton, S., Giesbrecht, J., Collier, J.: Software systems for robotics an applied research perspective. In: Advanced Robotics Systems (2006)
6. Ferrari, M., Ferrari, G., Astolfo, D.: Building Robots with LEGO Mindstorms NXT. Syngress Media (2007)
7. Giralt, G., Sobek, R., Chatila, R.: A multi-level planning and navigation system for a mobile robot; A first approach to HILARE. In: International Joint Conference on Artificial Intelligence, pp. 335–337 (1979)

⁴ <http://www.mobilerobots.com>

⁵ <http://www.cybernetics.com>

⁶ <http://msdn.microsoft.com>

8. Halloway, S.: Programming Clojure. Pragmatic Programmers (May 2009)
9. Kim, H., Cho, Y.-J., Oh, S.-R.: CAMUS: a middleware supporting context-aware services for network-based robots. In: IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO), pp. 237–242 (2005)
10. Konolige, K., Myers, K., Ruspini, E., Saffiotti, A.: The Saphira Architecture: A Design for Autonomy (July 14, 1997)
11. Kułakowski, K.: Robust - Towards the Design of an Effective Control Library for Lego Mindstorms NXT. In: Proceedings of Conference on Software Engineering Techniques CEE-SET 2009 (September 2009)
12. Kułakowski, K.: cljRobust - Clojure Programming API for Lego Mindstorms NXT. In: 4th International KES Symposium on Agents and Multi-agent Systems – Technologies and Applications (KES AMSTA) (to be appeared, 2010)
13. Long, L., Hanford, S., Janrathitikarn, O.: A review of intelligent systems software for autonomous vehicles. In: IEEE Symposium on Computational Intelligence in Security and Defense Applications, CISDA (2007)
14. Matsui, T.: EusLisp for Object-Oriented Robot Programming. Computer Software - Japan Society For Software Science and Technology Journal (January 2006)
15. Matyjasik, P., Nalepa, G.J., Zięcik, P.: Prolog-Based Real-Time Intelligent Control of the Hexor Mobile Robot. In: Hertzberg, J., Beetz, M., Englert, R. (eds.) KI 2007. LNCS (LNAI), vol. 4667, pp. 485–488. Springer, Heidelberg (2007)
16. Montemerlo, M., Roy, N., Thrun, S.: Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit. Worlds. PhD thesis, Yale Univ., Dept. of Computer Proc. of the Conf. on Intelligent Robots and Systems, IROS (2003)
17. Moral, J.A.B.: Develop LeJOS programs step by step
18. Murphy, R.: Introduction to AI robotics, p. 466. MIT Press, Cambridge (January 2000)
19. Nilsson, N.: Shakey the Robot. Tech Note 323, AI Center, SRI International (1984)
20. Preston, S.: The Definitive Guide to Building Java Robots. APress (2006)
21. Rajan, K., Bernard, D., Dorais, G., Gamble, E.: Remote Agent: An autonomous control system for the new millennium. In: ECAI 2000: 14th European Conference on Artificial Intelligence (January 2000)
22. Sajkowski, M., Stenzel, T., Grzesiak, B.: Walking robot HEXOR® II - a versatile platform for engineering education. In: 13th Power Electronics and Motion Control Conference, EPE-PEMC 2008, pp. 956–960 (2008)
23. Ueda, R., Ogura, T., Okada, K., Inabab, M.: Design and implementation of humanoid programming system powered by deformable objects simulation. In: Proceedings of the 10th International Conference on Intelligent Autonomous Systems (IAS 2010) (January 2008)
24. Utz, H., Sablatnog, S., Enderle, S., Kraetzschmar, G.: Miro - middleware for mobile robot applications. IEEE Transactions on Robotics and Automation 18(4), 493–497 (2002)

RoboComp: A Tool-Based Robotics Framework

Luis Manso, Pilar Bachiller, Pablo Bustos, Pedro Núñez,
Ramón Cintas, and Luis Calderita*

Robotics and Artificial Vision Laboratory,
University of Extremadura, Spain

{lmanso,pilarb,pbustos,pnuntru,rcintas,lv.calderita}@unex.es
<http://robolab.unex.es>

Abstract. This paper presents RoboComp, an open-source component-oriented robotics framework. Ease of use and low development effort has proven to be two of the key issues to take into account when building frameworks. Due to the crucial role of development tools in these questions, this paper deeply describes the tools that make RoboComp more than just a middleware. To provide an overview of the developer experience, some examples are given throughout the text. It is also compared to the most open-source relevant projects with similar goals, specifying its weaknesses and strengths.

Keywords: robotics framework, software usability, programming tools.

1 Introduction

Robotics software has to deal with very specific problems such as complexity, code reuse and scalability, distribution, language and platform support, or hardware independence. These problems should be addressed with appropriate software engineering techniques and be transparent to the developer when possible.

Software complexity, from the developer point of view, is an important topic because scalability decrease as complexity increases. The robotics community is already aware of this fact and has steered towards component oriented programming (COP) [1]. Despite the ostensible consensus regarding the use of COP, many different approaches have been developed in order to deal with those robotics specific issues.

We propose RoboComp, a robotics framework that focuses on ease of use and rapid development without loss of technical features. RoboComp is based on Ice[2], a lightweight industrial-quality middleware. Instead of spending large amounts of time developing an ad-hoc middleware, it was found preferable to reuse an existing one.

* This work has been supported by grant PRI09A037, from the Ministry of Economy, Trade and Innovation of the Extremaduran Government, and by grant TSI-020301-2009-27, from the Spanish Government and the FEDER funds.

The most remarkable strength of RoboComp is the set of tools it is accompanied with. These tools make programming an easy and faster experience. RoboComp also provides a wide variety of components, ranging from hardware interface or data processing to robot behavior. The web of the project gives access to an extensive on-line documentation covering all the information users may need. New components can be easily integrated with existing ones. Moreover, RoboComp components can communicate with other widely used frameworks. In particular, successful integration has been achieved with Orca2, ROS and Player.

2 Main Characteristics

Several robotics frameworks have been previously proposed to meet different requirements (e.g. Carmen[4], JDE[5], Marie[6], Miro[7], MOOS[8], OpenRTM-aist[9], Orca2[10], OROCOS[11], Player[12], ROS[13], YARP[14]). For a best understanding of what RoboComp provides, this section describes its main characteristics.

2.1 Middleware Issues

Creating a new ad-hoc middleware would have involved a considerable amount of work, not just to create the middleware but also to maintain it. We chose Ice because it meets all the requirements we experimentally identified for a robotics framework:

- Different optional communication methods (e.g. RMI, pub/sub, AMI, AMD).
- IDL-based, strongly typed interfaces.
- Good performance with low overhead.
- Efficient communication.
- Multiple language support (e.g. C++, C#, Java, Python).
- Multiple platform support (e.g. Linux, Windows, Mac OS X).

Orca2[10] is also based on Ice. However, RoboComp presents additional features that complements the middleware layer: a) It provides a well-defined component structure that ease software development; b) it includes several tools that reduce the effort needed to create and maintain software components. One of the main reasons why a new framework was created, instead of adopting Orca2, is that it would involve imposing our component structure to previous users. Nevertheless, although they are independent projects, using the same middleware facilitates the interoperation between both frameworks. Therefore, RoboComp and Orca2 users can share their components and benefit from the advantages of both projects.

Despite other frameworks have developed their own communication engine [6][12][13][14], we found preferable to adopt Ice. Besides its features and the time saving, it allows a better interoperability with other frameworks. Moreover, Ice has been used in various critical projects[3], so it can be considered very mature because of the extensive testing it has passed.

2.2 Tools and Classes

Two of the aspects in which RoboComp extends Ice are the tools and the numerous set of classes it is equipped with. The set of tools is what enhances user experience, making easier to develop and deploy complex systems. They are deeply described in section 3. The set of classes comprises different issues related to robotics and computer vision such as matrix computation, hardware access, Kalman filtering, graphical widgets, fuzzy logic or robot proprioception.

Among the different available classes, the robot proprioception class, which we call *InnerModel*, plays an important role in robotic software. It deals with robot body representation and geometric transformations between different reference frames. *InnerModel* is based on an XML description of the robot kinematics read from file. In this file, the transformations nodes (joints and links) are identified and described. *InnerModel* also provides different methods to estimate projections and frame transformations. Unlike other frameworks such as [13], RoboComp does not offer a separate component for such functionality. This decision was taken in order to reduce both, software complexity and latency. Indeed, the memory overhead of replicating the same object is negligible.

2.3 HAL Characteristics

Component oriented programming frameworks for robotics provide an effective way to keep software complexity under control. However, most of them have ignored one of the most important contributions of Player[12]: the idea of a robotics hardware abstraction layer (HAL). We think this idea is extremely important for code reuse and portability. All sensors and actuators of the same type can provide a common interface as long as they do not have very specific features. When possible, it is preferable to use the same interface: a) different users, researchers, or companies can share their components regardless of the underlying hardware, b) it reduces the impact of hardware updates, c) it prevents software deprecation [14].

Since it is possible to find hardware that does not fit the standard interface, RoboComp does not force the use of the proposed interfaces, just recommends it and emphasizes its benefits. Currently, RoboComp has defined the following HAL interfaces: Camera, DifferentialRobot, GPS, IMU, JointMotor, Joystick, Laser and Micro.

2.4 Component Structure

In order to ease development, a framework must provide a well-defined component structure. It helps users to understand which are the necessary code elements as well as the required links among these elements that lead to a correct deployment.

The skeleton of RoboComp components (figure 1) is composed of three main elements: the server interface, the *worker* and the proxies for communicating with other components. The *worker* class is the responsible of implementing the core functionality of each component. The server interface is a class derived

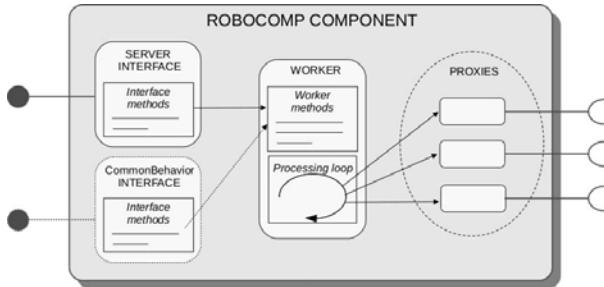


Fig. 1. General structure of a RoboComp component

from the Slice (Ice IDL) definition interface, which implements the services of the component. These services are generally handled by interacting with the *worker* class. Since delays are highly undesirable, both in the interface and in the core class, they run in different threads. Proxies are also of remarkable importance within a component. These, which are usually owned by the *worker*, are instances of auto-generated classes that provide access to other components. Components also include a configuration file where all the operational and communication parameters are specified. Using this configuration, the main program makes the necessary initializations to start the component.

Additionally, components may implement a common interface named *CommonBehavior*. This interface provides access to the parameters and status of the component and allows changing some aspects of its behavior at run time (for example, the frequency of its processing loop). This interface can be used to dynamically analyze the status of a component, providing a means of determining wrong configurations or potential execution errors.

3 Tools

A good robotics framework should not be just a bag of middleware features. Programming software for autonomous robots also involves other issues such as ease of use or readiness for an agile software development lifecycle. Thus, RoboComp is equipped with different tools that complement the middleware layer.

3.1 componentGenerator

Using a well-defined structure, the process of creating the skeleton of new components can be automated. Thus, to save the programmer from this tedious task, RoboComp includes a component generator tool. The component generator frees the programmer from middleware issues. These code pieces are automatically generated, so the programmer does not even have to read the whole component code. Our experience with new RoboComp users shows the great benefits of using this tool.

As an example of how to develop interacting RoboComp components using the *componentGenerator* tool, it is shown the source code of two components that communicate to compute the addition of two numbers. This example shows the few lines of code a RoboComp user has to write. The listings shown below correspond to the source files that should be modified by the programmer to add the desired behavior to both components. The client component reads two numbers, asks the server for the result of the addition of those numbers and prints the result. The server waits for client requests and computes the addition of numbers when it is requested to.

Listing 1.1. *worker.cpp* in the client side

```

1 #include "worker.h"
2
3 Worker::Worker(RobolabModAddTwoIntsServer::AddTwoIntsServerPrx
4     addtwointsserverprx, QObject *parent) : QObject(parent){
5     addtwointsserver = addtwointsserverprx;
6     connect(&timer, SIGNAL(timeout()), this, SLOT(compute()));
7     timer.start(BASIC_PERIOD);
8 }
9 Worker::~Worker(){}
10
11 void Worker::compute(){
12     int a, b, res;
13     cin>>a; cin>>b;
14     res = addtwointsserver->addTwoInts(a,b);
15     cout<<"a+b"<<res<<endl;
16 }
```

The listing **I.1** shows the only source file that should be modified in the client component. Remaining files of the component are not shown since their code is auto-generated and do not need to be modified. In listing **I.1**, only the highlighted lines are written by the programmer. Lines in gray are auto-generated. The line in brown (line 14) shows the sentence invoking the remote method on the server.

Listing 1.2. *AddTwoIntsServer.ice*

```

1 #ifndef ADDTWOINTSSERVER_ICE
2 #define ADDTWOINTSSERVER_ICE
3
4 module RobolabModAddTwoIntsServer{
5     interface AddTwoIntsServer{
6         int addTwoInts(int a, int b);
7     };
8 };
9
10#endif
```

Listings from **I.2** to **I.4** are the source files of the server component that have to be modified by the programmer in order to add the specific functionality. As in the listing **I.1**, lines written by the programmer are highlighted.

Listing 1.2 is the Slice definition, where the programmer has to specify the services that will be provided to other components. Listing 1.3 shows the server interface. It is the implementation of the class derived from the Slice definition that handles client requests. Besides the processing loop, the *worker* class (listing 1.4) includes the final implementation of the core of the component. As it can be observed in this example of server-component development, the programmer only has to include those lines related to service implementation without taking into account low-level issues.

Listing 1.3. *AddTwoIntsServerI.cpp*

```

1 #include "AddTwoIntsServerI.h"
2
3 AddTwoIntsServerI::AddTwoIntsServerI( Worker *_worker, QObject *parent)
4     : QObject(parent) {
5     worker = _worker;
6     mutex = worker->mutex;
7 }
8
9 AddTwoIntsServerI::~AddTwoIntsServerI(){}
10
11 int AddTwoIntsServerI::addTwoInts(int a, int b, const Ice::Current&){}
12     return worker->addTwoInts(a,b);
13 }
```

Listing 1.4. *worker.cpp* in the server side

```

1 #include "worker.h"
2
3 Worker::Worker(QObject *parent) : QObject(parent){
4     mutex = new QMutex;
5     connect(&timer, SIGNAL(timeout()), this, SLOT(compute()));
6     timer.start(BASIC_PERIOD);
7 }
8
9 Worker::~Worker() {}
10
11 void Worker::compute() {}
12
13 int Worker::addTwoInts(int a, int b) {
14     return (a + b);
15 }
```

Due to the use of a well-defined component structure, it is easier to create code generation or modification scripts. An example of this advantage is *addProxies*. This tool allows the programmer to automatically modify a previously created component including all the middleware-related code that is necessary to connect to new interfaces.

3.2 managerComp

Components are independently executed programs that interact with each other. They can be executed manually, but when a system contains more than a few components, it becomes hard to manage. In order to make this process easier, a component manager, *managerComp*, has been developed (figure 2).

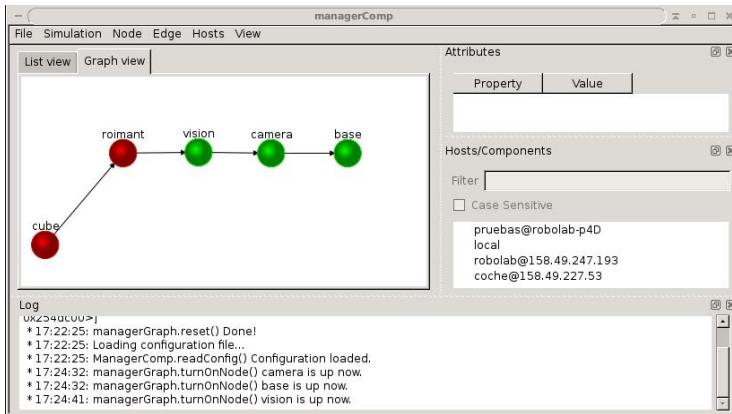


Fig. 2. Graphical interface of the *managerComp* tool

This tool allows users to graphically build and run a system in an intuitive way: **a)** components located within a host list are automatically found; **b)** components are included in the system by using a simple drag&drop operation; **c)** component dependencies are specified drawing arrows connecting nodes of the system graph (see Graph View tab of figure 2); **d)** components can be started/stopped by simply clicking the corresponding node of the graph view. This tool also facilitates system execution: when the user starts a component, all its dependencies are automatically satisfied. All these operations can be performed for components in both, local and remote hosts.

Moreover, *managerComp* can be used not only to manage local or remote components, but also to achieve certain level of introspection. As it was explained in section 2.4, components may optionally implement the common interface *CommonBehavior*. The manager tool uses this interface in order to give visual access to the parameters of the components. This is useful to detect wrong configurations or even to change operational attributes. Thanks to this feature, the manager tool can be used to help users diagnosing potential errors which would otherwise be harder to detect.

3.3 monitorComp

Developing new components may involve the application of a test process to evaluate its operation. Without a specific tool, testing a component entails the creation of another component connecting to the new one in order to check its functionality. To facilitate this test process, RoboComp includes *monitorComp*, a tool for component connection and monitoring. This tool allows the programmer either to include custom monitoring code or to use one of the templates available to test HAL components.

monitorComp provides a graphical interface that helps the programmer to carry out the component test process in an easy way. The first step is to connect to the corresponding component. The programmer has to indicate the endpoint

of the component as well as its Slice definition file. Once the connection data is introduced, the next step is the insertion of the monitoring code. The language used for writing this code is Python. It allows to minimize the size and the complexity of the testing code. Once the test code is written, *monitorComp* can run tests without any user input. All this information can also be read from file.

3.4 replayComp

The *replayComp* tool (figure 3) records the output of a set of components in order to subsequently emulate their roles. During emulation, it can run at the desired speed or manually step by step. This feature is extremely useful for debugging purposes: if the inputs are the same, the behavior of programs should also be the same. Thus, it is very helpful when trying to reproduce errors.



Fig. 3. Graphical view of *replayComp*

This tool can run in two different modes: capture and replay. In capture mode, *replayComp* can connect to different components and record their outputs. Its output file can then be transparently used in a replaying stage as input for other components (i.e. components reading data from *replayComp* will not be able to differentiate *replayComp* from real components). This way, it enables software development when no robot is physically available.

Currently, *replayComp* supports all the HAL component interfaces out of the box. Besides, the set of components that *replayComp* can record and replay can be extended by writing small Python scripts that can be included in this tool as *plugins*.

3.5 Simulator Support

A replay component is very useful for debugging software that does not need the robot to be active. However, when debugging active robot behaviors, a replay component is not enough. With a real robot, reproducing the same software behavior several times is a useful but very difficult task. Additionally, at initial phases of component development, it can be very helpful to test its operation

considering ideal conditions, instead of a noisy real environment. In order to meet these conditions, a robotics simulator seems a very adequate tool. Therefore, RoboComp includes support for Stage and Gazebo, two widely used open-source 2D and 3D robot simulators.

In order to make the use of the simulator transparent for the software, the simulator support has been included by extending the HAL components. As a result, higher-level components do not need any change to run whether on the simulator or on a real robot.

3.6 loggerComp

This tool is an optional standard output alternative designed to analyze the execution of components and their interactions. It provides a graphical interface to display the information of interest, allowing the user to filter by: type, component, time stamp, source code file and line of code. In order to use loggerComp, components only have to include the *loggerComp* proxy (which can be automatically done through the *addProxies* utility) and use the provided helper class *qLog*.

4 Framework Comparison

Throughout this paper, different robotics framework features have been discussed. Depending on the task, robots may have to use a particular platform or programming language. Different communication methods may also be very interesting features in order to fit to the different interoperational requirements and communication patterns. Finally, due to the complexity of the problems to solve, the toolset has been proved to be one of the most remarkable characteristics of a robotics framework. In order to provide a global view of the features and benefits of RoboComp, the rest of this section presents a comparison with the most relevant frameworks.

Table 1 shows the license, supported platforms and programming languages of the different frameworks. Note that both RoboComp and Orca2 support a wide variety of possibilities derived from the use of the Ice middleware.

Table 2 shows the middleware used by the different frameworks and some of its associated features. Those frameworks providing an IDL-based strongly typed interface are usually preferable: they are easier to understand and, therefore, reduce the possibility of programming errors. In RoboComp and Orca2, the Ice compilation tools are used in order to convert IDL interfaces into code of specific programming languages. Frameworks using CORBA IDL are also equipped with similar tools. The remaining frameworks do not include any IDL description facilities. Regarding the communication methods, Ice-based frameworks provide a good variety of both synchronous and asynchronous communication mechanisms. This feature allows the programmer to select the most suitable component interaction method regarding to their operation requirements.

Table 3 shows the tools provided by the reviewed frameworks. *Text* means that the framework provides a console-based tool, *gui* corresponds to a graphical user

Table 1. Middleware general aspects

Framework	License	Supported Platforms	Programming Languages
Carmen	GPL	Linux Windows	C C++ Java Python
Marie	LGPL	Linux	C C++
Miro	GPL	Linux	C C++
MOOS	GPL	Linux Mac OS X	C++ Matlab
OpenRTM	EPL	Linux Windows FreeBSD	C++ Python Java
Orca2	GPL/GPL	Linux Windows Mac OS X Android iPhone	C++ C# Python PHP Ruby Java Objective-C
OROCOS	GPL/GPL	Linux Windows	C++
RoboComp	GPL	Linux Windows Mac OS X Android iPhone	C++ C# Python PHP Ruby Java Objective-C
ROS	BSD	Linux Windows	C++ Python Octave Lisp
YARP	GPL	Linux Windows QNX	C++ Java Ruby Python Lisp

Table 2. Networking and API

Framework	Middleware	Comm. Methods	IDL
Carmen	IPC	pub/sub query/response	No IDL
Marie	ACE	data-flows (socket-based)	No IDL
Miro	ACE+TAO	sync/async client/server	CORBA IDL
MOOS	custom	pub/sub	No IDL
OpenRTM	ACE	pub/sub query/response	CORBA IDL
Orca2	Ice	RPC, AMI, AMD, pub/sub	Ice IDL
OROCOS	ACE	commands, events, methods, properties, data-ports	CORBA IDL
RoboComp	Ice	RPC, AMI, AMD, pub/sub	Ice IDL
ROS	custom	pub/sub query/response	No IDL
YARP	ACE	asynchronous data-flows	No IDL

interface, and *no* to the unavailability of a tool. Column '*code gen.*' represents the availability of a code generation tool. Despite RoboComp, OpenRTM and ROS provide this facility, only RoboComp and OpenRTM produce ready-to-use code: the ROS *roscreate-pkg* tool creates the directory tree structure of a ROS package as well as the necessary files for compilation, however it does not deal with source code generation. In addition, RoboComp provides a tool for code modification that allows adding new communication interfaces to a previously written component. Column '*manager*' represents the existence of an easy to use and specific tool to deploy and inspect the status of the deployed components (Orca2 uses IceGrid to deploy components but it is not framework specific or user friendly). Columns '*replay*', and '*log*' represent the availability of tools for component replaying and for logging messages, respectively. The '*simulator*' column lists the simulators the different frameworks provide connection to. Allowing components to work with a simulator without even changing the code

Table 3. Tools

Framework	Code Gen.	Manager	Replay	Simulator	Log	Monitor
Carmen	no	no	gui	custom 2d	gui	no
Marie	no	no	no	stage gazebo	no	no
Miro	no	no	no	no	gui	no
MOOS	no	text	gui	uMVS	gui	no
OpenRTM	gui	gui	no	stage gazebo	gui	no
Orca2	no	no	gui	stage gazebo	no	no
OROCOS	no	text	no	no	no	no
RoboComp	gui	gui	gui	stage gazebo	gui	gui
ROS	partial (text)	text	gui	stage gazebo	gui	gui
YARP	no	gui	no	icubSim	text	no

is a extremely useful feature. Thus, RoboComp provides a completely transparent use of Stage and Gazebo, two well-known robot simulators. The '*monitor*' column represents the availability of a tool to monitor the information that a component is working with, not just its status (i.e. images of a camera or the pose of a robot platform). Both ROS and RoboComp have such tool. However, *monitorComp* is more advanced than the tools ROS is equipped with: *rxbag* has a plugin system that allows users to display new types of data but it can only work with off-line data; *rxplot* works with on-line data but only with a few data types. *monitorComp* provides both features.

On the basis of this comparison, we think that RoboComp has a remarkable set of technical and user-oriented features that make it an outstanding robotics framework. However, we are aware of the big effort that would involve for new users to migrate their software from a framework to another. Thus, the on-line documentation of RoboComp describes the procedure to achieve interoperation with Orca2 and ROS, two of the most widely used frameworks in robotics.

5 Conclusions and Future Works

This paper has described RoboComp, a tool-based robotics framework, emphasizing the goals it intended to meet and the design decisions that have been made. RoboComp stands up in the comparison in several aspects such as ease of use, portability, language support or its toolset.

Due to the big efforts made by the scientific community, robotics frameworks are improving quickly. Despite RoboComp improves developer experience in comparison to other frameworks, enhancements are already being made: new features, new tools, or more introspection capabilities are under way.

References

- He, J., Li, X., Liu, Z.: Component-based Software Engineering: the Need to Link Methods and their Theories. In: Van Hung, D., Wirsing, M. (eds.) ICTAC 2005. LNCS, vol. 3722, pp. 70–95. Springer, Heidelberg (2005)

2. Henning, M., Spruiell, M.: Distributed Programming with Ice (2009)
3. ZeroC. ZeroC Customers, <http://zeroc.com/customers.html>
4. Montemerlo, M., Roy, N., Thrun, S.: Perspectives on Standardization in Mobile Robot Programming: The Carnegie Mellon Navigation (CARMEN) Toolkit. In: Proc. of International Conference on Intelligent Robots and Systems (2003)
5. Cañas, J.M., Ruíz-Ayúcar, J., Agüero, C., Martín, F.: Jde-neoc: component oriented software architecture for robotics. *Journal of Physical Agents* 1(1), 1–6 (2007)
6. Côté, C., Brosseau, Y., Létourneau, D., Raïevsky, C., Michaud, F.: Using MARIE for Mobile Robot Component Development and Integration. In: Software Engineering for Experimental Robotics, Springer, pp. 211–230. Springer, Heidelberg (2007)
7. Utz, H., Mayer, G., Kaufmann, U., Kraetzschmar, G.: VIP: The Video Image Processing Framework Based on the MIRO Middleware. In: Software Engineering for Experimental Robotics, pp. 325–344. Springer, Heidelberg (2007)
8. Newman, P.: MOOS - Mission Orientated Operating Suite. Massachusetts Institute of Technology, Dept. of Ocean Engineering (2006)
9. National Institute of Advanced Industrial Science and Technology (AIST). RT-Middleware: OpenRTM-aist (2010), <http://www.openrtm.org/>
10. Brooks, A., Kaupp, T., Makarenko, A., Williams, S., Orebäck, A.: Orca: A Component Model and Repository. In: Software Engineering for Experimental Robotics, pp. 231–251. Springer, Heidelberg (2007)
11. Bruyninckx, H.: Open Robot Control Software: the OROCOS project. In: Proc. of International Conference on Intelligent Robots and Systems, pp. 2523–2528 (2001)
12. Gerkey, B., Collet, T., MacDonald, B.: Player 2.0: Toward a Practical Robot Programming Framework. In: Proc. of the Australasian Conf. on Robotics and Automation (2005)
13. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.: ROS: an open-source Robot Operating System. In: ICRA Workshop on Open Source Software (2009)
14. Fitzpatrick, P., Metta, G., Natale, L.: Towards Long-Lived Robot Genes. *Journal of Robotics and Autonomous Systems* 56(1), 29–45 (2008)

Improving a Robotics Framework with Real-Time and High-Performance Features

Jesús Martínez*, Adrián Romero-Garcés, Luis Manso, and Pablo Bustos

Computer Science Department, University of Málaga, Spain

jmcruz@lcc.uma.es

Dept. Tecnología de los Computadores y las Comunicaciones,

University of Extremadura, Spain

adrigt@unex.es, lmanso@unex.es, pbustos@unex.es

Abstract. Middleware has a key role in modern and object-oriented robotics frameworks, which aim at developing reusable, scalable and maintainable systems using different platforms and programming languages. However, complex robotics software falls into the category of distributed real-time systems with stringent requirements in terms of throughput, latency and jitter. This paper introduces and analyzes a methodology to improve an existing robotics framework with real-time and high-performance features using a recently adopted standard: the Data Distribution Service (DDS).

Keywords: robotics, middleware, data distribution service, performance, real-time.

1 Introduction

Software Engineering for robotics is focused on designing and implementing robot control architectures to be reusable, scalable and maintainable, so that software modules and algorithms for robots can be adapted to new platforms and requirements, not only reducing the cost and time-to-market of a complete robotics system but also improving its overall performance. In this application domain, the complex tasks performed by a robot are divided into distributed processes, which communicate using a middleware. This key software layer connects networked applications in a platform and language-independent manner. The middleware also hides low-level communication details from developers.

One of the most important milestones in the robotics field has been the creation of libraries and object-oriented frameworks for building robotics software. Although some of these frameworks are now mature and widely used in the robotics community for research purposes, sometimes it is unclear whether they will allow developers to deploy complete systems in embedded platforms which

* This project has been partially supported by grants TSI-020301-2009-27 from the Spanish Government and the FEDER funds and by P07-TIC-003184 from the Junta de Andalucía funds.

meet the stringent real-time requirements that a running robot needs. Therefore, rigorous analysis must be done to guarantee deterministic behaviors that do not miss any critical deadline, and it is not surprising that the use of third-party software, such as middlewares, may affect the predictability and performance of a system if they can not be adjusted properly.

Unfortunately, some middlewares are not best suited to implement distributed and real-time embedded (DRE) robotic systems. However, widely adopted frameworks may not replace middleware easily, unless they are redesigned from scratch. This paper presents our experiences to adapt a robotics framework to be DRE compliant. In our study we have selected the novel *Data Distribution Service for Real-Time Systems* (DDS) standard [10] from the Object Management Group (OMG), which addresses the anonymous, decoupled, and asynchronous communication among a data sender, called publisher, and its subscribers. This standard addresses the needs for real-time and quality of service (QoS) of distributed applications, and it is being adopted in mission- and business-critical applications, such as air traffic control, telemetry or financial trading systems.

The main contributions of the paper are *i*) the proposal and demonstration that a DDS-based middleware improves significantly a component-based robotics framework (RoboComp [15]) in terms of throughput, latency and jitter, and *ii*) a methodology to incorporate this middleware to the framework with minimum changes from a developer's perspective, which also ensures backward compatibility with previously deployed modules. Our study shows interesting results that help to conclude that the use of DDS will be important for the robotics community in the years to come.

The paper is organized as follows. Section 2 gives an overview of the state of the art in middleware for robotics. Section 3 introduces our methodology to include DDS in RoboComp, a component-based robotics framework. Finally, we give some concluding remarks.

2 State of the Art in Middleware for Robotics

Design patterns [4] and frameworks have been used within the robotics community in many proposals [5][7][2][3][1][15]. In general, these open source approaches provide a catalogue of services (from low-level to high-level tasks) which allow developers to reuse existing code in order to create complex software for a robot. They also rely on some kind of middleware to allow the deployment and communication of services within a distributed platform, where resource-intensive software modules are executed in different network nodes.

The middlewares proposed by Player [5] and Carmen [7] are the most basic ones, and require some management of low-level communication details from developers in order to implement new distributed services, for instance to define new message types in a platform-independent way using a specific C API (Carmen) or using the eXternal Data Representation (XDR) notation [16] and its C compiler (Player).

Other frameworks hide these complexities from developers by using platform and language-independent standard middlewares, which follow the distributed

object computing paradigm, such as the CORBA [1] standard by the Object Management Group (OMG) or the Internet Communications Engine (Ice) by ZeroC [19] (an industrial-quality middleware used in many critical projects). The two middlewares make use of an Interface Definition Language (IDL) to define communication interfaces for distributed objects, which will be available through the so-called Object Request Broker (ORB). This is the approach followed by Orocó [2] and Miro [3] (CORBA-based), or by Orca [4] and RoboComp (Ice-based).

All of the above-mentioned frameworks may be configured to use a traditional client/server communication model (one-to-one) for remote procedure calls (RPC) or remote method invocations (RMI), although Carmen and Orca also use a publish/subscribe model (one-to-many). The latter is usually implemented using a central process (the broker) that delivers published messages to the processes that were previously subscribed to them. Although this mechanism decouples the way in which robotic services communicate, the use of a central broker has a direct impact in the overall performance of the system, reducing also its fault tolerance.

As software for robotics falls into the category of DRE systems, Orocó and Miro include mechanisms to ensure predictability and fully deterministic behavior. Orocó makes use of the so-called Real-Time Toolkit, a set of C++ primitives to implement (lock-free) data exchanges and event-driven services in hard real-time. Miro relies on TAO [17], an open source implementation of the real-time CORBA standard [8] in C++. TAO provides predictable end-to-end quality of service in a modular and flexible design. It has an ORB that supports real-time concurrency and real-time event services [6] for CORBA.

Although CORBA (or Ice-based) applications are sometimes referred to as components, they are not exactly equivalent. Components are autonomous and loosely coupled pieces of software which allow developers to create more modular and extensible software. Therefore, the Object Management Group has standardized the CORBA Component Model (CCM) [9]. CCM aims at creating component-based distributed systems which communicate using well-defined CORBA interfaces. A component is described using the Component Implementation Definition Language (CIDL), an extended version of the CORBA IDL. CCM components are composed of attributes, facets (provided interfaces) receptacles (dependencies on external interfaces) and event sources and sinks. Components run within containers, which provide their runtime environment and are also responsible for local and remote communications (using an ORB). CCM is now a mature specification for component-based developments and it is already available in several open source middlewares such as the Component Integrated ACE ORB (CIAO) [18], which supports the TAO real-time ORB, making it possible to maintain predictable behavior. Nevertheless, The CORBA Component Model (and its associated specifications) are not widely used. In spite of that, the OMG aims at their future adoption within the robotics community by standardizing its novel Robotic Technology Component Specification [12], which focuses on the structural and behavioral features required by robotics software as a supplement to a general component model.

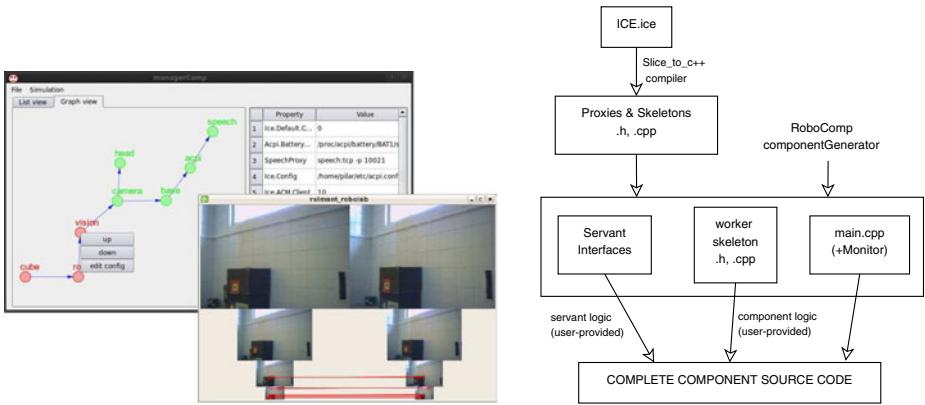


Fig. 1. RoboComp management tools (left) and its component generation process (right)

Meanwhile, some of the robotics frameworks described above have defined their own component model, such as Orococos and RoboComp. The components in Orococos are not language-independent and must be implemented directly in C++. However, RoboComp takes advantage of the Ice IDL capabilities to define and implement component interfaces for several programming languages. The following section discusses some of the main features of the RoboComp component model and its powerful management tools.

3 Improving Robotics Software with DDS

This section describes the improvements we have made to the RoboComp framework to be DRE-compliant. Therefore, we have recommended and justified the use of the DDS standard (and one of its open source implementations) as the most appropriate real-time middleware for RoboComp.

3.1 The RoboComp Framework

RoboComp is a general purpose, open-source and component-based robotics framework. It provides ease of use and rapid development of robust software by providing a wide set of management tools that help robot software developers in most of their everyday tasks. RoboComp is also designed to be used as a hardware abstraction layer (HAL) for sensors and actuators (like Player) by standardizing some of its component interfaces.

As mentioned in the previous section, RoboComp relies on the Ice middleware, which has a wide language and platform support. Ice allows developers to design distributed applications with a high-degree of fault-tolerance and security, using remote exceptions and secure connections, respectively. It supports RMI (in both synchronous and asynchronous mode) and publish/subscribe communications (using a centralized broker called IceStorm).

The point where RoboComp departs from Orca, or other Ice-based frameworks, is in its component model and in its configuration model. RoboComp components are composed of four modules. The first module is the main procedure, which acts as a ready-to-use runtime environment and as the component container. The second module includes the Servant classes that inherit from the skeletons generated by the Ice IDL compiler. They must implement the behavior associated with the interface operations. The third module is the *Monitor* class, a thread in charge of the initialization procedures which checks that the component is ready for a safe execution. The fourth module is the *Worker* class, which is the component main class. It encapsulates its behavior including the communication with other components using Proxies (the RMI stubs generated by the IDL compiler).

Fortunately, RoboComp provides developers with an automatic component generator tool (*componentGenerator*), a command line application which generates code from IDL files. Users only need to specify the interface names of the other components that will be accessed during the execution. The complete process is depicted in fig. ② (right).

RoboComp also includes powerful management tools. The *component management tool* (fig. ②-left in background) is a visual application which displays the status of the whole system as a graph of interacting components. Each node in the graph is depicted with a color that represents the activity/inactivity of the component. Upon demand, it is also able to show the configuration properties of each component, and to start or to stop them while respecting their dependencies. The *monitoring tool* (fig. ②-left in foreground) allows users to write and perform unitary tests to RoboComp software components. This tool allows programmers either to include their own monitoring code or to use one of the templates available to test HAL components, making it possible to display the outputs of the tested component. In order to analyze the behavior of a component or a group of interacting components, users can benefit from the logging facilities of the *loggerComp tool*, which includes filters to display logs with different criteria, such as date, priority or sender. Finally, the output of RoboComp components can be recorded (and subsequently replayed) by the *replayComp tool*. Thereafter, components can connect to the replayComp application and obtain previously recorded data. This tool is really useful for off-line debugging, that is, when there is no robot hardware available. Besides, RoboComp provides support for some well-known simulators, such as Stage and Gazebo (from Player).

Despite the powerful component and configuration models provided by RoboComp, its wide adoption could be compromised if it does not fit well the stringent real-time and performance requirements of a DRE system. Fig. ② shows some results that confirm this statement. They correspond to some experiments that measure round-trip latency and jitter in a distributed scenario. This scenario consist of two nodes in a local area network (gigabit ethernet), which are executing two components: a sender and a receiver. The results are obtained for different configurations which use the underlying Ice communication protocols.

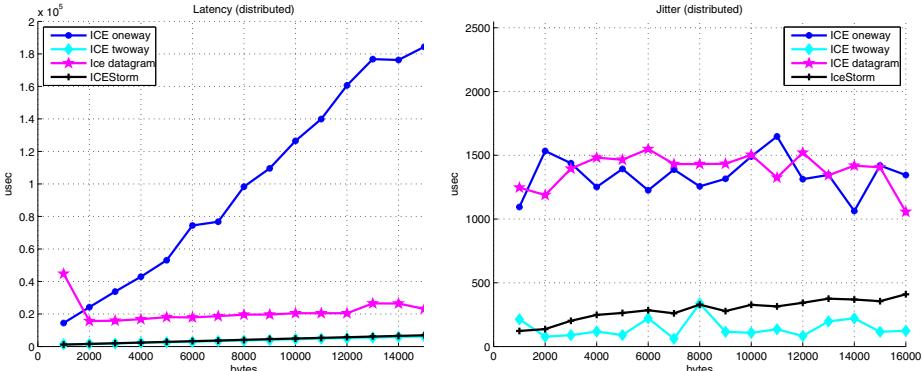


Fig. 2. Latency and jitter results in RoboComp for different transport configurations

So called *oneway* and *twoway* configurations use the TCP transport protocol, whereas the *datagram* configuration uses UDP. A comprehensive analysis on the advantages and disadvantages of these transport protocols is beyond the scope of this article, although it is surprising that the results are better for the *twoway* mode, and it is also worth noting that TCP is not the best choice to implement real-time distributed services (figs. 4 and 5 will give a closer look at these values).

The main conclusion is that the combination of the Ice proprietary RMI protocol with TCP gives results that are worse than what it should be allowed for very precise real-time configurations (especially for jitter whose values in the experiment range between 77 and 337 microseconds). Thus, it becomes clear that RoboComp needs an alternative middleware with real-time and high-performance features.

3.2 The Data Distribution Service

The OMG published the Data Distribution Service for Real-time Systems standard in 2004. This specification focus on describing a middleware based on the publish/subscribe model for distributing data with high-performance on real-time environments, where systems must be predictable and deterministic. Since its inception, the DDS has been used in defense and aerospace applications, radar processes, naval combat management or air traffic control systems, among others.

The DDS standard is composed of the Data-Centric Publish and Subscribe layer (commonly referred to as DCPS) and by the DDS Interoperability Wire Protocol (DDSI v2.1) [4]. The former defines the DDS architecture, participants and standard API, along with some profiles which enhance its use [10]. The latter defines a new protocol which ensures interoperability across DDS implementations from different vendors.

The publish/subscribe model implemented in DDS does not use a central broker. Publishers and subscribers access to the so-called global space data to

exchange information, which avoids a single point of failure. Data in DDS are defined as *topics*. Topics are described as type-safe data structures, which also contain a key (to identify different topic instances) and an associated quality of service. These QoS policies specify resource limits for delivery, liveness or reliability, among other features. Moreover, publishers and subscribers can also have QoS associated, which must be compatible before a communication takes place. As in other OMG specifications, the definition of topics is done using an IDL (in this case, a subset of the CORBA IDL), which can be compiled to primitives and data structures for specific programming languages.

After the increasing success of the DDS standard, the OMG is trying to incorporate its main benefits to the Corba Component Model. Therefore, it is now defining the so-called DDS4CCM specification [13] (now in beta 2), where CCM will benefit from DDS features using special artifacts called *connectors*. The connectors will overcome the intrinsic limitations of the CORBA IDL interfaces, which originally were intended to work with a one-to-one communication model.

Regarding the availability of DDS implementations, there are a few commercial products but also some open source ones, such as OpenSplice DDS from Prismtech (GPLv3 license) and OpenDDS from OCI (BSD-like license). The former middleware is the most complete one in its free version (referred to as *Community Edition*), which has support for the complete DCPS and the Interoperability Wire Protocol in C, C++, C# and Java. Therefore, we have selected OpenSplice DDS in C++ to improve the RoboComp framework.

3.3 Mapping RPC/RMI Concepts to DDS

In order to hide the DDS API and its communication model from developers, we propose a general methodology which maps existing RPC/RMI communication semantics to their equivalent publisher/subscriber operations. First of all, we

Table 1. Mapping RMI operations to DDS topics

RMI interface	DDS request data type	DDS response data type
[return_type] operation();	data.operation.request { long id; };	//only if return_type!=void data.operation.response { long id; return_type value; };
[return_type] operation(type param);	data.operation.request { long id; type param; };	//only if return_type!=void data.operation.response { long id; return_type value; };
[return_type] operation(out type param);	data.operation.request { long id; };	data.operation.response { long id; [return_type value;] type param; };
[return_type] operation(type iparam, out type oparam);	data.operation.request { long id; type iparam; };	data.operation.response { long id; [return_type value;] type oparam; };

have to emulate the RMI message exchange. A client request will be equivalent to the publication of a topic instance, which is identified with a key (for instance, a random identifier or a sequence number). The server (part of the component runtime environment) subscribes to this kind of topic and *i*) executes the reception asynchronously, *ii*) invokes the appropriate method of the component *Worker* instance with the input provided by the received data and *iii*) prepares a response topic whether needed (for instance, if the equivalent RMI operation returns a non-void type or if it contains output parameters) and then publishes it using also the same key that was found in the request. The client will wait for the response, which is perfectly identified by its matching key value.

Table [II](#) enumerates our proposed mappings. For the sake of clarity we have used a simplified notation that does not match exactly with the Ice and OpenSplice IDL grammar, and the symbols enclosed in brackets are optional. The first column represents the four main types of method signatures (called *operations*), which are part of the interfaces defined in the IDL file. The second and third columns represent their equivalent DDS topics which implement the emulated request/response protocol. As mentioned above, the response is not always needed and, therefore, topics in the third column (and their associated publish/subscribe operations) will not be used. In order to match a request with its corresponding response, topics will share the same key value. This key must be provided first by clients as part of the topic structure (the `long id` field in table [II](#)).

These guidelines have been used to design a new Ice IDL to OpenSplice DDS IDL compiler in order to obtain the RMI-equivalent DDS topics and stuff. Fortunately, Ice has a modular compiler that implements the Visitor pattern [III](#) to navigate through the abstract syntax tree available with the IDL file. Our compiler makes extensive use of it. Until now, it supports only a subset of the Ice IDL language (interfaces without inheritance). This decision has been motivated by the usual interface descriptions found in the existing repository of RoboComp components. However, a more complete compiler to DDS is under development.

3.4 Implementation and Discussion

Fig. [II](#) (right) has shown the development process of a RoboComp component. Our proposed extension now includes DDS capabilities and is depicted in fig. [B](#), where dark boxes represent the files and modules that are modified. First of all, we have extended the capabilities of the Ice IDL compiler in order to generate a DDS IDL file that follows the mapping procedures described in table [II](#). Therefore, we obtain the appropriate definition of DDS topics which will be the input to the OpenSplice IDL compiler to C++. The resulting files implement the DDS publishers, subscribers and other data structures and procedures that will work with our defined topics using the DDS API.

Our next modification affects the RoboComp code generation script, which now needs to generate source code to implement the subscription and publication of the request/response topics using DDS. From a developer's point of view, the new code is still backward compatible with existing RoboComp components, that

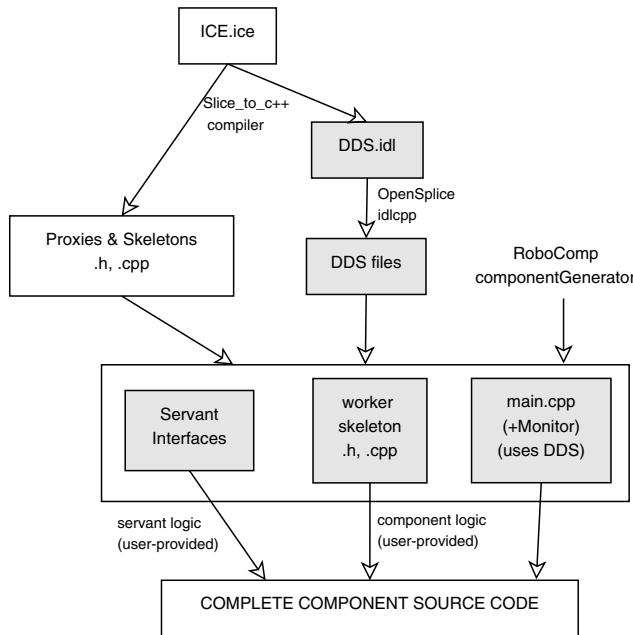


Fig. 3. The improved component generation process with DDS

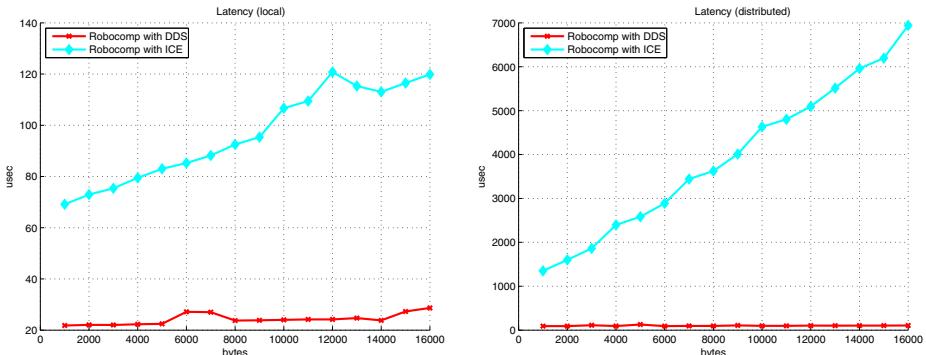


Fig. 4. Latency measurements for RMI and DDS communications in RoboComp

is, the Worker class can still use the traditional way of invoking remote operations as client using any Ice Proxy. However, these Proxies have been extended to include an extra method for every existing operation. These new methods share the signature of the original ones, although their names incorporate the suffix *_rt* (real-time), which means that they will use our DDS communication mechanism instead. Regarding the execution environment of the RoboComp module, the main procedure activates the servant objects, but also includes the automatic subscription to every request topic available with the DDS IDL.

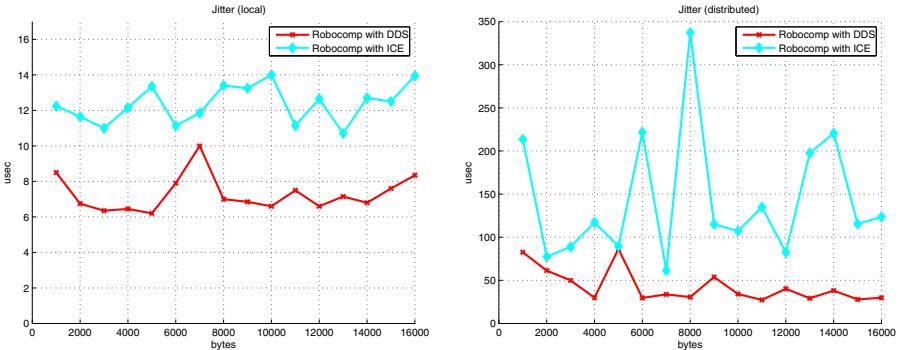


Fig. 5. Jitter measurements for RMI and DDS communications in RoboComp

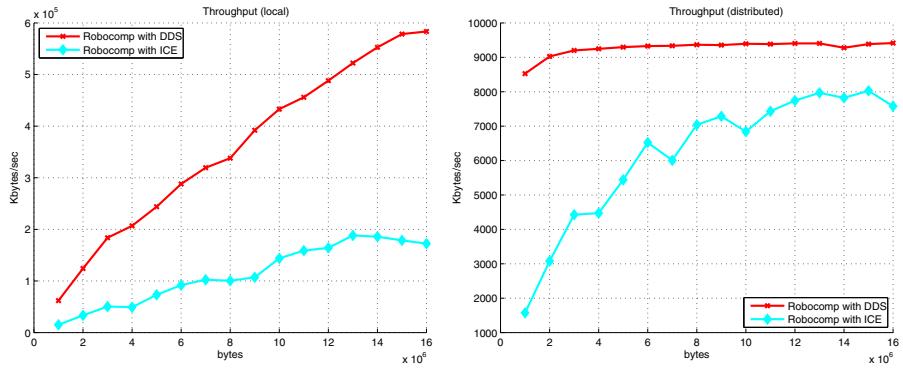


Fig. 6. Throughput measurements for RMI and DDS communications in RoboComp

Figs. 4, 5 and 6 show our experiment results after comparing communications using the original Ice-based RoboComp RMI mechanism and using the new DDS one for two scenarios: local (figures on the left) and distributed (figures on the right). The experiments consisted of a client component using the middleware facilities for sending messages (with different sizes) to a server component. This server implemented an echo service that sent every message back to the client. The middlewares were used with equivalent configurations: Ice with TCP and two-way mode, and DDS with reliable and ordered delivery.

It is worth noting that figures 4 and 5 show how latency and jitter are bounded and stable in the DDS case. In fact, latency is reduced to be less than 29 microseconds for local communications and less than 128 in the distributed scenario (and an average of 105 microseconds). These are impressive results with respect to their corresponding numbers for RMI. The results in the local scenario are also justified by the use of shared memory as the interprocess communication method in OpenSplice DDS. Therefore, we can conclude that every RoboComp component that uses DDS now can be DRE-compliant. The same good results

are obtained for throughput in fig. 6, which also means that DDS uses the protocol with the lowest overhead in networked scenarios.

4 Conclusions and Future Work

This paper has described our experiences on improving an existing robotics framework with real-time and high-performance features. Our main conclusions are that robotic software modules can benefit from the recent Data Distribution Service standard and from the emerging open source toolkits that implement it, such as OpenSplice DDS or OpenDDS. After the improvements, we have demonstrated how DDS middlewares allow the communication among robotic components with low latency and jitter, but also with a throughput that outperforms the one obtained with the previous protocol for RMI.

We have also developed a methodology that is little intrusive with respect to the existing IDL compilers and RoboComp code generators and that guarantees backward compatibility with previous deployed components. It is worth noting that our RMI to DDS mapping guide is valid for any Ice-based framework but can be used also as a reference for other IDL language mappings.

Another great advantage of using DDS is that it now adds quality of service features to the robotics framework, it being possible to design and deploy real-time applications for the robot. We plan to work intensively on exploring and adjusting these new DDS QoS features in order to understand which configurations are better suited in the context of the robotics software.

References

1. Brooks, A., Kaupp, T., Makarenko, A., Williams, S., Orebäck, A.: Orca: A component model and repository. In: Brugali, D. (ed.) *Software Engineering for Experimental Robotics*. Springer Tracts in Advanced Robotics. Springer, Heidelberg (April 2007)
2. Bruyninckx, H., Soetens, P., Koninckx, B.: The real-time motion control core of the Orocos project. In: IEEE International Conference on Robotics and Automation, pp. 2766–2771 (2003)
3. Enderle, S., Utz, H., Sablatnög, S., Simon, S., Kraetzschmar, G., Palm, G.: Miro: Middleware for Autonomous Mobile Robots. In: Telematics Applications in Automation and Robotics (2001)
4. Gamma, E., Helm, H., Johnson, R., Vlissides, J.: *Design Patterns*. Addison-Wesley Pub Co., Reading (1995)
5. Gerkey, B.P., Vaughan, R.T., Howard, A.: The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In: Proceedings of the 11th International Conference on Advanced Robotics, pp. 317–323 (2003)
6. Harrison, T., Levine, D.L., Schmidt, D.C.: The design and performance of a real-time corba event service. In: Proceedings of OOPSLA 1997, Atlanta, GA, pp. 184–199. ACM, New York (1997)
7. Montemerlo, M., Roy, N., Thrun, S.: Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (CARMEN) toolkit. In: Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS), pp. 2436–2441 (2003)

8. Object Management Group: Real Time CORBA (2005)
9. Object Management Group: CORBA Component Model (CCM), version 4.0 (2006)
10. Object Management Group: Data Distribution Service for Real-time Systems (DDS), version 1.2 (2007)
11. Object Management Group: Common Object Request Broker Architecture, CORBA/IOP (2008)
12. Object Management Group: Robotic Technology Component (RTC), version 1.0 (2008)
13. Object Management Group: DDS for Lightweight CCM (DDS4CCM), in process version 1.0 beta 2 (2009), <http://www.omg.org/spec/dds4ccm/1.0/Beta2>
14. Object Management Group: The Real-time Publish-Subscribe Wire Protocol DDS Interoperability Wire Protocol specification, version 2.1 (2009)
15. Bustos, P., Bachiller, P., Manso, L.: RoboComp Project (2010),
<http://sourceforge.net/apps/mediawiki/robocomp>
16. Srivivasan, R.: Request for Comments 1832: XDR: External Data Representation Standard (1995)
17. Schmidt, D.C., Gokhale, A., Harrison, T.H., Levine, D., Cleeland, C.: Tao: a high-performance endsystem architecture for real-time corba (1997)
18. Wang, N., Schmidt, D., Gokhale, A., Rodrigues, C., Natarajan, B., Loyall, J., Schantz, R., Gill, C.: Qos-enabled middleware. In: Mahmoud, Q. (ed.) Middleware for Communications (2003)
19. ZeroC: Internet Communications Engine (2010), <http://www.zeroc.com/>

Implementation of Distributed Production System for Heterogeneous Multiprocessor Robotic Systems

Yosuke Matsusaka and Isao Hara

National Institute of Advanced Industrial Science and Technology (AIST),
1-1-1 Umezono, Tsukuba, Ibaraki, Japan

Abstract. We have developed an RTDPS (Robotic Technology Distributed Production System) for use as a framework for realizing optimized performance of an integrated robot. The RTDPS is a set of software built on OpenRTM-aist (an implementation of OMG RT-middleware specification), which consists of a script compiler, node components, and deployment utility. By distributing the application logics among the processing elements, the RTDPS can prevent the centralized control of integrated systems, which causes system bottlenecks. By performing experiments involving two typical applications, we show that the network traffic can be reduced by 73% while balancing the processing load among the processing elements.

1 Introduction

Recently, there are increasing demand for robotic intelligence.

There are projects that have been launched for developing software components for robotic applications [1] [2]. OpenRTM-aist is a software developed by AIST, which implements RT-middleware architecture standardized at OMG [3]. In OpenRTM-aist, a component is implemented as a node. An integrated system is constructed using nodes connected with links. Ideally, the system integrator can develop any kind of applications easily by selecting the required components from existing components and linking them together.

However, in practice, there are problems in the deployment of the integrated system. Because each of the components consumes processing power and because each of the links could consume network bandwidth, the allocation of the nodes to the processing elements significantly influences the overall performance of a total robotic system. Owing to these problems, the system developer has to design not only the links between the components but also the actual allocation of the nodes to the processing elements by considering the overall performance of the system. This adjustment process requires considerable effort, especially for robots that use many components and many processors.

In practical development, the system integrator has to write application logics to realize a desired behavior of the robot in addition to designing the network for the components. If the developer implements the application logics as a single

component, the component will be a supervisor component and will be connected to all the other components in the system. The supervisor component tends to be a bottleneck in the system because all the inputs and outputs depend on the component (described in the next section).

To solve this problem, we propose a framework in which the application logics are divided into parts and distributed among the processing elements. We use a scripting environment based on a production system model to write the application logics. First, we compile the script into a network of nodes. Subsequently, we use an algorithm that automatically determines the optimal allocations of nodes. By using this process, the developer can obtain an optimal distribution of the application logics to improve the performance of the system with less effort.

“Rete algorithm” is an algorithm developed by Forgy [5] in 1974. This algorithm converts a script in production system form to a network. Some studies have attempted to run the Rete network in parallel to realize high performance [6][7][8][9][10]. However, most of the studies consider dense connection multiprocessor systems with symmetric structures. For these systems, the algorithm can be simplified to distribute the matching processes to the processing elements in an equal manner (e.g., [7]). However, these algorithms cannot be applied to robotic systems directly because most of the current robotic systems use sparse connection heterogeneous multiprocessor architecture (will be described in the next section).

Objective of our research is by extending an optimization algorithm for node allocation in addition to conventional Rete algorithm, we realize optimal performance even on the heterogeneous multiprocessor networks. Our ultimate aim is to realize high performance robotic intelligence that has numerous functionalities integrated together.

In Section 2, we briefly overview existing integration architectures. In Section 3, we describe the conventional production system model and the Rete algorithm. In Section 4, we introduce the formalization of our proposed algorithm. In Section 5, we show our optimization algorithm for node allocation and buffer insertion. In Sections 6 and 7, we explain the implementation and present an evaluation of the algorithm.

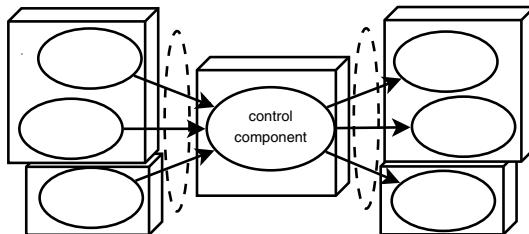
2 Integration Architectures for Intelligent Robots

In this section, we discuss the classification of the integration architectures for intelligent robots from two viewpoints: network topology and control architecture. Our proposed architecture uses distributed control in a sparse network topology. We briefly review past studies from both viewpoints.

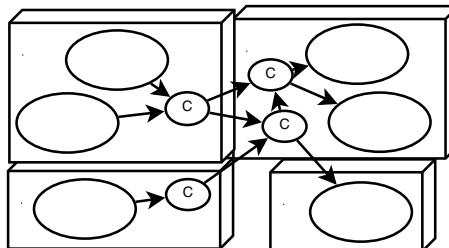
From early times, the robotic system has comprised a multiprocessor system. A multiprocessor robot system has the following advantages: **Show high performance:** Some robotic systems require high processing power, which is difficult to provide with a single processor system. **Use specific processing architecture with specific algorithm:** Some processing algorithms (e.g., image processing) can run more efficiently with specific architecture (e.g., GPU).

Increment a processing element and the function simultaneously: Multiprocessor robot systems can increment a new function with the processing power required to run the function. This reduce the integration effort largely.

Network topologies of multiprocessor robot systems can be roughly classified into two types: dense connection multiprocessor robot systems and sparse connection multiprocessor robot systems. The former was developed to realize high-performance robots. Yamasaki et al. [11] developed pluggable processor architecture to realize a multifunctional mobile robot system. Matsui et al. [12] have developed an office service robot (Jijo2) and a lisp implementation that has an interprocess messaging function [13]. Later, the sparse connection multiprocessor robot system was introduced. It was developed to realize high processing power on a small robot platform. It is widely used in general robot systems. Inaba et al. proposed the concept of remote brain robotics and implemented distributed video and control processing applications [14]. Matsusaka et al. [15] have implemented an architecture in which a centralized publish-subscribe server is used to share information among multiprocessor robot systems. OpenRTM-aist [3] and ROS [4] are component architectures that use the publish-subscribe model on the import and outport and facilitate supervised control of the connections. By using the above described architectures, each function of the robot can be easily distributed to a processing element. However, after developing robotic applications, we need not only the required functions but also application logics to realize the integrated behavior of the robot.



A) Example of allocation of processing elements in the centralized control architecture. Dotted circles indicate the total load for network inputs and outputs.



B) Allocation of processing elements in the distributed control architecture. The circle with character "C" denotes the application logics that are divided into parts.

Fig. 1. Centralized control architecture and distributed control architecture. The boxes indicate processing elements. Circles denote functional components.

For this purpose, most of the robotic architectures are adopting centralized control. For example, ORCA is an architecture developed by Toshiba [16]. It uses HORB components and python script to design a centralized controller that implements application logics. However, because all the inputs and the outputs are connected to the centralized controller component (see Fig. II-A), the controller component tends to become a bottleneck. For overcoming this problem, we propose a framework in which the application logics are divided into parts and distributed among the processing elements (see Fig. II-B).

3 Production System Model and Its Formalization

3.1 Terms

A production system consists of many rules, and a “Rule” consists of many conditions and actions. The term “condition” stands for an equation in which two or more values or symbols are compared, while “action” refers to an equation in which a value is assigned to a symbol. The term “script” stands for a text file that contains one or more rules.

The term “pattern matching” implies a process that compares conditions in rules to the current values of the symbols and finds the rules that match all the conditions. “Conflict resolution” refers to a process that selects one rule from the matched rules on the basis of a given priority. The term “fire” implies activating the actions defined in the selected rule. The term “cycle” stands for a sequence of pattern matching, conflict resolution, and fire.

Fig. 2 shows an example of the script using production system model. In this script, rules are defined for the following actions: to track a visual object once it is detected (imagetrack), to track an audio event when there is no visual object (auditiontrack), and to remain actionless if neither the object nor the event is detected (notrack).

3.2 Rete Network

The Rete algorithm is an algorithm developed by Forgy [5]. It compiles the script in production system form to a network that consists of nodes and links (called “Rete network”).

```

rule: name = "imagetrack," priority = 1.0
  condition: vision.detected != 0      # condition 1-1
  action: motorcontrol.direction = vision.direction
rule: name = "auditiontrack," priority = 0.9
  condition: vision.detected == 0      # condition 2-1
  condition: audition.detected != 0    # condition 2-2
  action: motorcontrol.direction = audition.direction
rule: name = "notrack," priority = 0.8
  condition: vision.detected == 0      # condition 3-1
  condition: audition.detected == 0    # condition 3-2
  action: motorcontrol.direction = 0

```

Fig. 2. Example of a script using a production system model. The script implements multimodal tracking by using audio and video information sources.

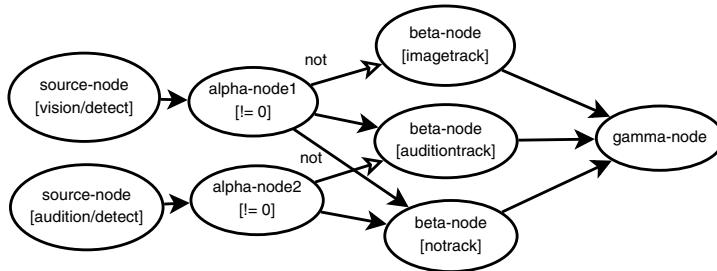


Fig. 3. Rete network that implements the functions described in the script shown in Fig. 2

Here, the term “node” stands for a component of the system that has an ability to compute simple formulas. The term “link” refers to a connection to the information transfer channel, which connects two nodes.

In the Rete algorithm, “alpha node,” “beta node,” and “gamma (conflict resolution) node” are used. The functions of the nodes are as follows: **Alpha node:** It evaluates a condition. **Beta node:** Node which has function to merge the results between two or more nodes. **Gamma (conflict resolution) node:** It evaluates the results of two or more nodes on the basis of a given priority.

3.3 Rete Algorithm

The script can be converted to a Rete network by using an algorithm that performs the following steps.

Step 1: Unify same conditions in the rules. This process can be carried out by using the following pseudo-code.

```

cond = {}
for r in rules:
    for c in r.conditions:
        if c not in cond:
            cond[c] = newcond(c)
            cond[inv(c)] = inv(cond[c])
            createalphanode(c)
        
```

Step 2: Create links between alpha and beta nodes. This process can be performed by using the following pseudo-code.

```

cmat = zeros(n, n) # initialize connection matrix
for r in rules:
    b = createbetanode(r)
    for c in r.conditions:
        cmat[b, cond[c]] = 1 # connect nodes
    
```

Step 3: Create links between beta and gamma nodes. This process can be carried out by using the pseudo-code given below.

```

g = creategammanode()
for r in rules:
    b = getbetanode(r)
    cmat[g, b] = 1 # connect nodes
    g.setpriority(b, r.priority)
    g.setactions(b, r.actions)
    
```

Fig. 3 shows an example of the Rete network. The network is converted from the script shown in Fig. 2. Alpha-node 1 in Fig. 3 corresponds to the conditions 1-1, 2-1, and 3-1 in Fig. 2. Similarly, alpha-node 2 corresponds to the conditions 2-2 and 3-2. Each beta node corresponds to a specific rule.

4 Heterogeneous Distributed Production System

The Rete algorithm was originally developed to reduce the computation cost of the pattern-matching process in production systems with massive rules [5]. However, in this paper, we show that an extension of the algorithm is also effective in improving system performance when used for optimizing the allocation of nodes in sparse connection heterogeneous multiprocessor systems.

4.1 Terms

We define the term “processing element” as a processor that is used to run each component. “Node location” stands for the processing element whose node is running. The processing element can contain any number of nodes, provided the total computing cost of the nodes does not exceed the “processing capacity” of the element.

The term “physical network” stands for a physical network connection between the processing elements. Note that the physical network is different from the link in the Rete network. The physical network can contain any number of links, provided the total network cost of the links does not exceed the “network capacity” of the network.

“Packet” stands for information transmitted between links in the Rete network, and “update frequency” stands for the number of packet transmissions in the period.

4.2 Allocation of Nodes and Its Effects on System Performance

The alpha node has a function to compare a symbol with a value. Because the node only transmits a packet when the result of comparison changes, the update frequency will be reduced between the input and output links.

The network cost is defined as the product of packet size M and update frequency F :

$$T_i = F_i M_i \quad (i \in N) \quad (1)$$

Here N denotes the set of nodes.

The total cost of each physical network S_n^c is proportional to the product of the number of links passing through the physical network and the information cost.

$$S_n^c = \sum_{i \in N} T_i C_{i,n} + \sum_{i \in N} T_n C_{n,i} \quad (f(i) = n) \quad (2)$$

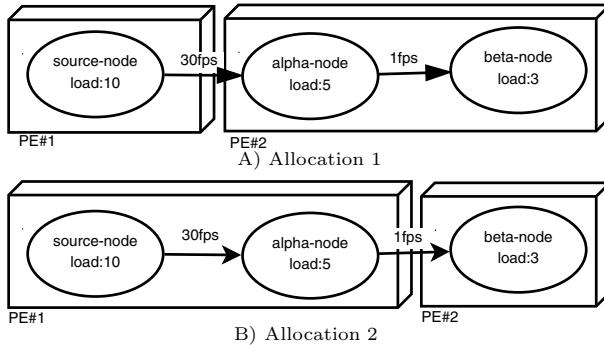


Fig. 4. Effects of allocation of nodes on the system performance

Table 1. Performance of the system shown in Fig. 4

Configuration	Processing load		Network load (bps)
	PE#1	PE#2	
Allocation 1	10	8	240
Allocation 2	15	3	8

Here, P denotes the set of processing elements; f , a mapping function that maps the node to the processing element (later explained in Section 5.1); and C , the connection matrix (nodes i and j are connected to each other when $C_{i,j} = 1$).

The processing load of each processing element is proportional to the number of nodes.

$$S_n^p = \sum_{i \in N} R_i \quad (f(i) = n) \quad (3)$$

Here, R represents the processing cost for each node.

We now explain the relation between the allocation of nodes and system performance through two examples.

Fig. 4-A shows alpha-node locate same processing element to the source-node. Fig. 4-B shows alpha-node locate different processing element to the source-node.

All Rete networks have the same function, but the system performance differs if there are differences in the allocation of the nodes. Table 1 shows the system performance in each example. It is apparent that the network load and processing load have a trade-off relationship.

4.3 Immutability of Each Nodes

Here we define two terms: “non-immutable node” implies a node that can work in a specific processing element and “immutable node” represents a node that can work in any processing element.

Alpha nodes, beta nodes, and gamma nodes are immutable nodes. The other nodes (source and output nodes) are the non-immutable nodes.

The performance of the system can be improved by optimizing the allocation of the immutable nodes. In the next section, we describe the algorithm that we have developed to automate the optimization.

5 Algorithm for Finding Optimal Allocation of Nodes

5.1 Cost Function

Let N be a set of nodes; P , the set of processing elements; C , a connection matrix of the Rete network; T , information cost (see Section 4.2); R , the processing cost for each node (see Section 4.2); L_p , processing power capacity; and L_c , physical network capacity. Here, the cost function of the entire system S is defined as follows:

$$S = \sum_{i \in N} \sum_{j \in N} T_i W_{f(i), f(j)} C_{i,j} + \frac{\sum_{i \in P} (w_p L(l_p, S_i^p) + w_c L(l_c, S_i^c))}{P} \quad (4)$$

Here, T stands for information cost (see Section 4.2). W is a $P \times P$ matrix and denotes the network cost; W_{ij} represents the network cost from processing element $i \in P$ to processing element $j \in P$. f is a mapping function that defines the current allocation of nodes $f : N \rightarrow P$. L is a penalty function that limits the processing load and network load so that they do not exceed the capacity of each processing element (w_p and w_c are the weights given to the penalty functions).

$$L(l, f(x)) = \max[0, f(x) - l]^2 \quad (5)$$

5.2 Cost Function with Buffer Node Insertion

Here, we introduce a new node type, “buffer node.” A buffer node accepts an input packet from an upstream node and outputs it to the downstream nodes. The network performance can be improved by inserting a buffer node between the physical network connections.

Fig. 5 shows examples of systems with and without a buffer node. We see that the amount of network transfer has reduced by inserting the buffer node.

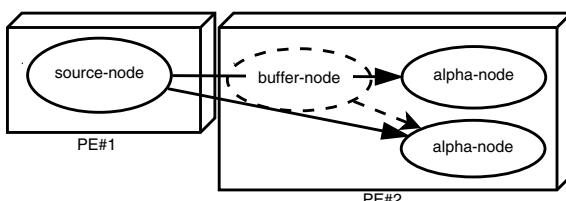


Fig. 5. Effects of insertion of buffer node. The number of links between processing elements 1 and 2 can be reduced by inserting a buffer node.

In order to perform optimization by inserting buffer nodes, we modify the connection matrix using the following pseudo-code:

```

nodecrossing = {}
for n1 in nodes:
    for n2 in nodes:
        if cmat[n1, n2]:
            nodecrossing[n1.outport, n1.loc, n2.loc]++
cmat2 = zeros(n, n)
for n1 in nodes:
    for n2 in nodes:
        if nodecrossing[n1.outport, n1.loc, n2.loc] == 1:
            cmat2[n1, n2] = 1

```

5.3 Optimization Algorithm

Given the weight function in (4), the optimal allocation of nodes can be determined by modifying the mapping function f to minimize the weight function.

The minimization of the weight function can be accomplished by several methods. In this paper, we use a genetic algorithm (GA) [17]. We consider the weight function as the fitness function and the N dimensional vector \mathbf{g} as the gene ($\mathbf{g}_i \in P$). We also consider a uniform distribution for the mutation and assume the mutation rate to be 0.2, with random initialization.

6 Implementation

6.1 Parameter Definition File and Default Parameter Generator

The parameter definition file is an XML file that contains the host names of all processing elements, and the values of F , M , W , l_p , and l_c (see Section 4.2 and 5.1 for details).

In order to reduce the labor involved in defining each parameter, we have developed a utility script that generates the default value of each parameter. By using this utility script, developers only have to modify the appropriate specific values for their application.

6.2 Network Compiler

The network compiler is implemented in Python. The script in production system form is the input to the network compiler. The script is compiled as a Rete network by using the algorithm presented in Section 3. Then, node allocation optimization is performed using the algorithm given in Section 5. Optimization using the genetic algorithm is implemented using PyEvolve¹. Compilation of the network is done off-line (the compilation time will be discussed in Section 7. The network compiler outputs the compiled network in XML form (termed “network definition file” hereafter).

¹ <http://pyevolve.sourceforge.net/>

6.3 System Configurator

The system configurator deploys the network definition file to the real system. First, it loads and configures the node components that run on the processing elements. Next, it creates links between the nodes on the basis of the network definition file. Finally, it activates all the components (decreasing order of preference is gamma node, beta node, and alpha node). After the deployment, the system runs autonomously without having to be configured.

7 Evaluation

7.1 Tasks

In order to evaluate the effectiveness of the system, we implemented the following two test cases and compared the performance of system not using the RTDPS and with that using the RTDPS.

Multi-modal tracking: An object is tracked from video and audio information. Runs with three processing elements (vision, audio, and robot control). The script is shown in Fig. 2

HumanAID: Tasks are performed by a HRP-2 robot (e.g., picking objects and controlling electric equipments) in living space. Runs with three processing elements (speech recognition, robot control, and localization). The total script consists of 8 rules with 13 conditions.

7.2 Evaluation Metrics

We evaluated the performance of the systems using following metrics: **Total processing load:** The sum of the processing costs of all the nodes. **Total network load:** The sum of the network costs of all the physical networks. **Processing/network load of each processing element:** In order to determine this parameter, the network and the processing loads in the system are averaged, and we calculate maximum and minimum values of the processing/network loads for each processing element. **Network optimization time:** The time taken to construct the network using the optimization algorithm (on a 2.4-GHz Intel Core 2 Duo machine).

7.3 Result

Fig. 6 shows a network constructed by using the RTDPS. Fig. 7 shows a network constructed by using the RTDPS along with a buffer-node insertion algorithm. For the HumanAID application, we show the results on the web page 8. In the original network, the entire control process is located at PE#3. In contrast, the process is distributed in the RTDPS network. In the RTDPS network, a new buffer-node is inserted in PE#2 and some nodes are allocated to balance the load of each processing element.

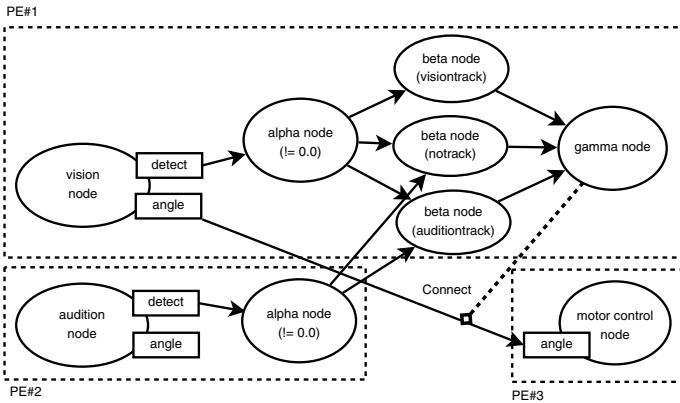


Fig. 6. Network generated for multimodal tracking application by using the RTDPS

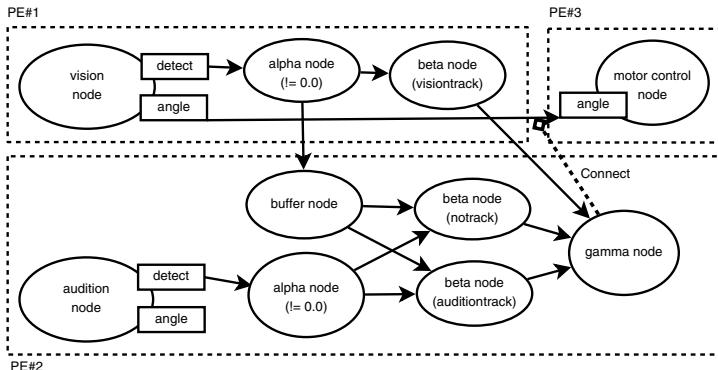


Fig. 7. Network generated for multimodal tracking application by using the RTDPS along with a buffer-insertion algorithm

Table 2 shows the results of the optimization. It is apparent that in the multimodal tracking application, the total network load is reduced by 73% by introducing the RTDPS. This reduction results from the introducing the alpha node in each of the processing elements, which run the recognition processes. This node not only serves to distribute the processing load among the different processing elements but also reduces the network traffic, as discussed in Section 4.2. In the HumanAID application, because the application is based on event-driven architecture that implements dynamic task switching, the influence of the RTDPS is limited. The RTDPS is effective to balance the processing load to each processing elements, and the total network load is greater than that of the original network.

² <http://staff.aist.go.jp/yosuke.matsusaka/RTDPS/results.html>

Table 2. Results of the optimization.

	Total load Proc.	Each PE Net.	(max,min) Proc.	Time Net.	(s)
MMT-Original	50	960	40, 10	960, 480	-
MMT-RTDPS	50	257	25, 10	257, 17	4.63
MMT-RTDPS+B	51	257	23, 10	257, 17	4.75
HAID-Original	37.1	1.6	17.1, 10.0	1.6, 0.8	-
HAID-RTDPS	37.1	7.2	14.1, 11.1	5.6, 3.2	8.32
HAID-RTDPS+B	37.2	5.6	15.2, 10.8	4.8, 2.4	8.31

* Network load is in bps. Processing load is calculated from specific parameters.

8 Summary and Future Work

In this paper, we have presented a formalization of a heterogeneous distributed production system and developed a software RTDPS (Robotic Technology Distributed Production System) for use as a framework for realizing optimized performance of an integrated robot. The RTDPS is built on OpenRTM-aist, which consists of a script compiler, node components, and deployment utility. We have shown that the traffic of the network can be reduced by 73% while balancing the processing load among the processors, by distributing the control logics among the processing elements.

In this study we only considered the optimization problem with static parameters. However, in some robotic systems capable of dynamic task switching (e.g., the HumanAID application), the values of the parameters may change dynamically depending on the situation, and the current optimization algorithm has shown limited effectiveness in improving the performance of these systems. To overcome this limitation, we are currently implementing a function to monitor the activity of each node remotely, and we are developing an algorithm to realize dynamic optimization based on on-line statistics.

References

1. Brooks, A., Kaupp, T., Makarenko, A., Orebäck, A., Williams, S.: Towards Component-Based Robotics. In: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 163–168 (2005)
2. Sato, T., Matsuhiro, N., Oyama, E.: Common Platform Technology for Next-Generation Robots. In: Workshop on Standard and Common Platform for Robotics, International Conference on Simulation, Modeling and Programming for Autonomous Robots (2008)
3. Ando, N., Suehiro, T., Kitagaki, K., Kotoku, T., Yoon, W.: RT-Middleware: Distributed Component Middleware for RT (Robot Technology). In: Proc. International Conference on Intelligent Robots and Systems, pp. 3555–3560 (2005)
4. ROS - Robot Open Source – Willow Garage,
<http://www.willowgarage.com/pages/software/ros-platform>
5. Forgy, C.: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. Artificial Intelligence 19, 17–37 (1982)

6. Gupta, A.: Parallelism in Production Systems, Research Notes in Artificial Intelligence (1987)
7. Gupta, A., Forgy, C., Kalp, D., Newell, A., Tambe, M.: Parallel OPS5 on the Encore Multimax. In: Proc. International Conference on Parallel Processing, pp. 271–280 (1988)
8. Laird, J., Rosenbloom, P.: The Evolution of the Soar Cognitive Architecture. In: Steier, D., Mitchell, T. (eds.) Mind Matters: A Tribute To Allen Newell. Carnegie Mellon Symposia on Cognition Series, pp. 1–50 (1994)
9. Ishida, T.: Parallel, Distributed and Multi-Agent Production Systems: A Research Foundation for Distributed Artificial Intelligence. In: Proc. International Conference on Multi-Agent Systems, pp. 416–422 (1995)
10. Weiss, G. (ed.): Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press, Cambridge (1999)
11. Taira, T., Yamasaki, N.: Functionally Distributed Control Architecture for Autonomous Mobile Robots. Journal of Robotics and Mechatronics 16(2), 217–224 (2004)
12. Asoh, H., Hara, I., Matsui, T.: A Structured Dynamic Multi-Agent Architecture for Controlling Mobile Office-Conversant Robot. In: Proc. IEEE International Conference on Robotics and Automation, pp. 1552–1557 (1998)
13. Matsui, T.: Multithread Object-Oriented Language EusLisp for Parallel and Asynchronous Programming in Robotics. In: Workshop on Concurrent Object-based Systems, IEEE 6th Symposium on Parallel and Distributed Processing (1994)
14. Inaba, M.: Remote-Brained Robotics: Interfacing AI with Real World Behaviors. In: Proc. International Symposium on Robotics Research (1993)
15. Matsusaka, Y., Oku, K., Kobayashi, T.: Design and Implementation of Data Sharing Architecture for Multifunctional Robot Development. Systems and Computers in Japan 35(8), 54–65 (2004)
16. Orca, <http://orca-robotics.sourceforge.net/>
17. Fogel, D.: Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. IEEE Press, Los Alamitos (2006)

Robot Programming by Demonstration

Krishna Kumar Narayanan, Luis Felipe Posada, Frank Hoffmann,
and Torsten Bertram

Institute of Control Theory and Systems Engineering, Technische Universität Dortmund,
44227, Dortmund, Germany,
krishna.narayanan@tu-dortmund.de

Abstract. The manual design of vision based robotic behaviors remains a substantial challenge due to the complexity of visual information. It takes an ample effort to draw meaningful relationships and constraints between the acquired image perception and the geometry of environment both empirically and programmatically. This contribution proposes an alternative framework for learning autonomous visual navigation behavior from demonstration examples by integrating 3D range and an omnidirectional camera. A programming by demonstration approach is utilized to learn the demonstrated trajectories as a mapping between visual features computed on the omnidirectional image onto a corresponding robot motion. Exhaustive tests are performed to identify the discriminant features in order to mimic the teacher demonstrations. The relationship between perception and action is learned from demonstrations by means of locally weighted regression and artificial neural networks. The experimental results on the mobile robot indicate that the acquired visual behavior is robust and is able to generalize and optimize its performance to environments not presented during training.

1 Introduction

The decreasing cost of computer vision systems in conjunction with an increase in performance render them more attractive as the primary sensor in mobile robot navigation [58][12]. However the design of robust reactive visual robot behaviors remains a significant challenge if the goal is autonomous pure vision based navigation as 2D images lack depth information for 3D scene reconstruction, exhibit substantial variation in geometry, texture and visual appearance of indoor environments. These problems aggravate the conventional behavior based approach to identify prototypical scenarios and to design context specific albeit general reactive behaviors for them. One feasible approach is to reconstruct the geometry of the local environment from single or multiple views. The vision system more or less mimics the conventional functionality of proximity sensors, such that existing navigation behaviors can be again utilized. However, such an approach is prone to ignore information in the image that might be useful to improve the robots navigational capacities and skills. A vision system in contrast to a proximity sensor is able to distinguish between objects such as obstacles, walls or doors based on their visual appearance. This observation motivates an approach in which relevant visual features are automatically generated and identified. The main contribution of this

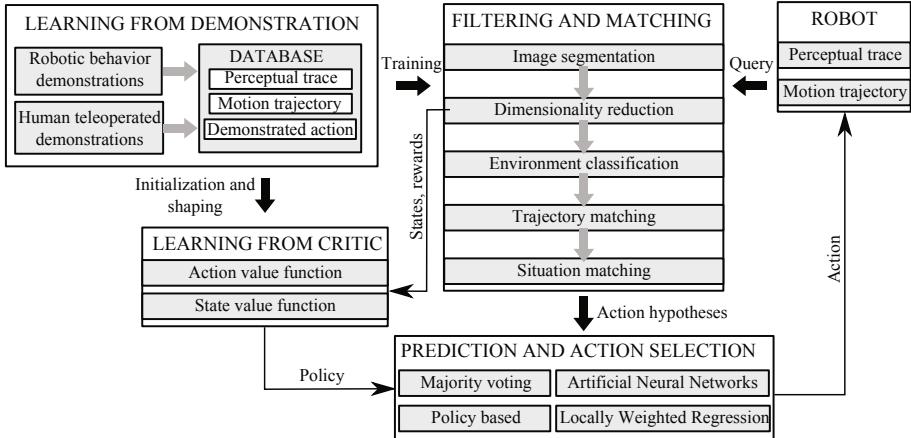


Fig. 1. Robot navigation learning framework

paper is the presentation of a new framework for visual robot navigation of indoor environments. The approach intuitively generates a policy from mimicking a sonar based navigation behavior and transfers it onto a more complex vision based behavior. The complete robot navigation framework (Fig. 1) is a three staged approach towards acquiring autonomous visual behavior through imitation of other robotic behaviors, explicit demonstration of a particular skill by a human teleoperating the robot and finally learning with a critic on the basis of the robots experiences. Robotic training examples are either generated from proximity sensor based behaviors or human teleoperation. With the former, action selection relies on proximity sensors in which the sequence of actions taken by the robot is recorded in conjunction with a trace of visual perceptions. Learning from robot demonstrations allows the transfer of existing behaviors onto a novel sensor modality without the need of explicitly reprogramming the behaviors [13]. However, in this type of learning the competence of the visual behavior does not exceed its proximity sensor based counterpart. For example, a sonar based demonstration is too ambiguous to acquire a robust vision based door traversal behavior. In these cases robot demonstrations under human teleoperation provide a means to enhance the performance of vision based navigation by exploiting the more complex and detailed visual information. In this contribution, we explore the ability to acquire autonomous robot behavior from a proximity sensor based behaviors. The key challenge is the generalization of perceptions and actions in demonstrated scenarios onto novel unseen situations. The diversity of indoor environments as well as the ambiguity and complexity of visual perceptions severely complicates the identification of appropriate measures of similarity. Any attempt to directly match images across different scenarios is bound to fail in natural unstructured environments. We propose a three stage scheme for matching the current situation with demonstrated scenarios. The first level is concerned with the classification of the environmental context into a limited set of prototypical classes, such as corridor, open space, cluttered environment, junction, door or dead end. In the case of omnidirectional vision the classification relies on the distribution and the shape of segmented

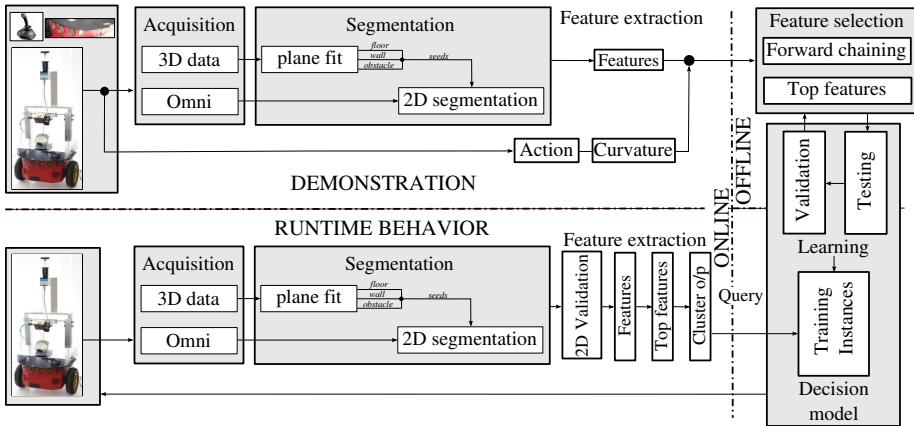


Fig. 2. System architecture: demonstration, learning and run-time behavior

free space in the image. The second level is concerned with the identification of similar scenarios, such as passing an isolated obstacle to the left. A scenario is characterized on the one hand by the trace of visual perceptions as well as the recent robot trajectory. Matching scenarios thus relies on the similarity of perceptual as well as motion trajectories. In order to cope with the complexity of visual perceptions, the information contained in the omnidirectional image is first reduced by segmentation into free space and obstacles [20]. The information complexity is further reduced by characterization of the free space by a small subset of geometric features. This dimensionality reduction of the segmented image is either obtained by means of a principal component analysis or feature selection through a wrapper approach [10]. In summary, scenarios are considered as similar if their perceptual trace projected onto a lower dimensional subspace and the robots planar trajectory match according to a trajectory specific distance metric based on the longest common subsequence [25].

The demonstrations are thus reduced to a small subset that match the current context in terms of environment, scenario and robot motion. Within each demonstration the state most similar to the current perception is identified and its associated recorded action constitutes a hypothesis about the robots action decision. Action selection computes an overall compromise decision on the basis of the multiple hypotheses in a way that is similar to command fusion in behavior based robotics. The aggregation either relies on majority voting or instance based weighted regression of demonstrated actions. Learning from demonstration endows the robot with a repertoire of robust visual behaviors suitable for a large variety of different scenarios. These skills provide the basis for further learning based on the robots own experience thus adapting its behaviors to the specific geometric structure and visual appearance of its environment. Reinforcement learning constitutes a means to learn an optimal policy on the basis of trial and error. Schaal proposes a framework for learning from demonstration in the context of reinforcement learning without the need to generate explicit reward or state space models [21]. The recorded demonstrations are used to initialize and shape the state or action value function, thus avoiding the need to learn from scratch. In general demonstrations

contain positive examples of successful scenarios, however in order to facilitate learning it is useful to provide a few negative counterexamples, such as deliberate collisions with obstacles as well.

This contribution is a first step towards the realization of the proposed integrated framework for robot learning from demonstrations as well as experience. It is concerned with the transfer of a sonar based obstacle avoidance and corridor following behavior onto a similar vision based policy. It addresses some of the key challenges outlined above, namely the matching of visual perceptions, the complexity reduction of visual information and the generalization of actions from demonstrations onto novel scenarios. Here, we present an integration of 3D range and 2D catadioptric omnidirectional camera to learn a robust indoor navigation behavior from demonstration examples (Fig. 2). The training examples are generated from a robust sonar based obstacle avoidance and corridor following behavior, during which 3D-2D images are captured together with the executed robot action. This approach allows it to transfer a sonar based behavior onto an equivalent vision based behavior. The captured images of the robots neighborhood are segmented into floor and obstacles. The robots local free space is described by a set of geometric features extracted from the segmented image. The mapping between robot action and geometric features is learned by means of locally weighted regression (LWR) [22][4] and artificial neural networks (ANN) [4]. Feature selection is performed by means of a wrapper approach (Forward chaining) in which the subset of features that best generalizes on unseen validation data is extracted.

Computer vision systems for imitation receive increasing attention in robotics such as mobile navigation [6][8][23][5] and object manipulation [7]. In [13], depth cues are obtained from monocular images and genetic programming is used to find obstacle avoidance algorithms and in [24], the authors propose color histograms to model the appearance of the floor obtained from a trapezoidal reference area using a single monocular camera. However, the system presents difficulties when the appearance of the floor changes abruptly. This problem is overcome using a 3D camera as a groundtruth to guide and train 2D segmentation features. One popular alternative to visual learning systems is to reconstruct the geometry of the local environment from single or multiple views [1] or using 3D laser scanners [9][7]. However, such an approach is prone to ignore information in the image that might be useful to improve the robots navigational capacities and skills. A vision system is able to distinguish between objects such as obstacles, walls or doors based on their visual appearance. This observation motivates our approach in which relevant visual features are automatically generated and identified. The robustness of the learned models are validated on unseen examples. The generated training data are cross-validated between similar but different scenarios and completely different scenarios for their generalization performance and correspondingly tested on the real system.

2 Segmentation

Here we present a brief description of the floor segmentation algorithm used. The Pioneer 3DX mobile robot is equipped with a 3D Range camera with 204×204 pixel resolution across a $40^\circ \times 40^\circ$ field of view (FOV) and an omnidirectional camera. The omnidirectional sensor consists of a CCD camera with a hyperbolic mirror with a

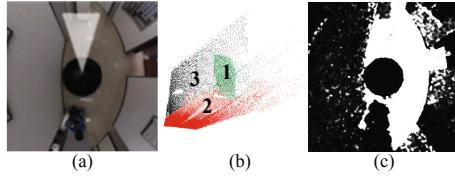


Fig. 3. 3D segmentation. (a) Omnidirectional image. Shaded region corresponds to 3D camera’s field of view, (b) RANSAC plane segmentation and (c) 2-D Histogram backprojection segmentation.

vertical field of view of 75° directed towards the bottom in order to capture the floor. The key idea is to use the reliable depth information of the 3D camera in the frontal view to initialize the floor, wall and obstacle models for the appearance based segmentation. The underlying assumption that floors in an indoor office environment are planar motivates the use of RANSAC algorithm to fit and estimate planar surfaces from the 3D point cloud. Authors of [16] make use of a combination of RANSAC and Iterative Closest Point (ICP) registration to extract planes offline and label them, whereas [19] propose plane fitting as an incremental region growing followed by polygonalization step to exploit the neighborhood relation for proper surface models. Since, we already possess the knowledge of the tilt and installation height of the 3D camera and that a geometric model is of secondary importance, an ICP registration deems itself unnecessary. By applying only RANSAC iteratively in that, the inliers of the best model are removed from the dataset and the next plane is estimated with the remaining points. The orientation of the surface normal and the area of the extracted plane are used to classify it as either floor, wall or obstacle robustly and fast (0.1s). Pixels that belong to a large horizontal plane are considered as floor, vertical planes as walls and small planes or remaining isolated non planar patches as obstacles. Figure 3(b) shows such a classification where three such planes are identified and their inliers visualized. A minimum of 200 inliers and a normal separation threshold of 0.05 m between a 3D point and the plane is set as a prerequisite to identify a plane and qualify an inlier. The labeled regions originating from the 3D segmentation are projected as ground truth segmentation onto the omnidirectional view through homography obtained from the intrinsic camera parameters. Histogram backprojection finds areas in the image whose appearance matches with a reference model. This technique has been successfully employed in image retrieval, in order to recognize objects reliably in cluttered scenes even under occlusion. We adopt this technique in order to recognize the floor area according to the histogram reference floor model obtained from the ground truth 3D segmentation. We refrain from going into the details of the algorithm, and recommend the readers to refer to the works of [20, 26] for further information. Figure 3(c) shows one instance of the segmentation with the 3D camera FOV represented by the shaded region (Fig. 3(a)).

3 Visual Features

In our case, the training data is generated by a robust wandering behavior emerging from the behavior fusion of obstacle avoidance and corridor following based on sonar

information that provides the training examples in terms of recorded perception-action pairs. With an identity mapping between the teacher and recorded execution [1], the velocity outputs of the demonstrated behavior are transcoded to traversed curvatures by computing the ratio between rotation and translational velocity. By representing the robot execution in terms of curvature, behavior learning is decoupled from the dynamics of the robot. A total of 8876 instances are generated in six distinct scenarios and environments, two corridor scenarios (2.5m wide) and four scenarios in an open room and foyer environment with isolated obstacles emphasizing obstacle avoidance. Training examples in each scenario are generated by recording perceptions and actions over a period of 10-15 minutes while the robot autonomously wanders in the environment controlled by the sonar based behavior. To increase the diversity of the robot poses and perceptions within the environment, the behavior and the recording is temporarily abandoned and a random change in heading direction is executed at regular time intervals. Each environment differs in the texture and appearance of the floor as well as the walls and obstacles as shown in Fig. 4. The omnidirectional image is segmented into free space and non free space by means of histogram backprojection. The backprojection segmented image presents itself as a similarity map P with a floor similarity values of a pixel whose location in the omnidirectional image is described in cylindrical coordinates defined by radial distance r and angular orientation θ . The similarity map $P(r, \theta)$ is filtered in that all pixels with a similarity lower than a threshold of $\delta = 0.7$ are removed. The similarity map is discretized into segments defined by the angular range $(\theta_{min}, \theta_{max})$ and radial intervals (R_{min}, R_{max}) . The partition into segments is uniform

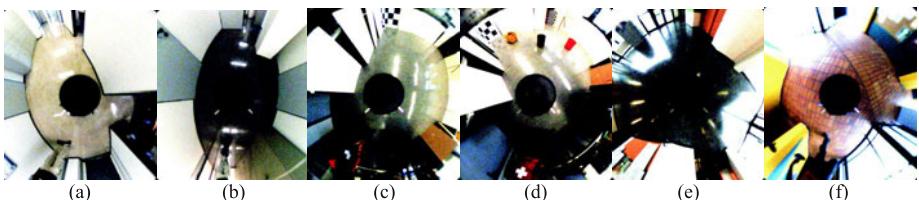


Fig. 4. Training scenarios. (a) Corridor 1, (b) Corridor 2, (c) Open room 1, (d) Open room 2, (e) Foyer 1 and (f) Foyer 2

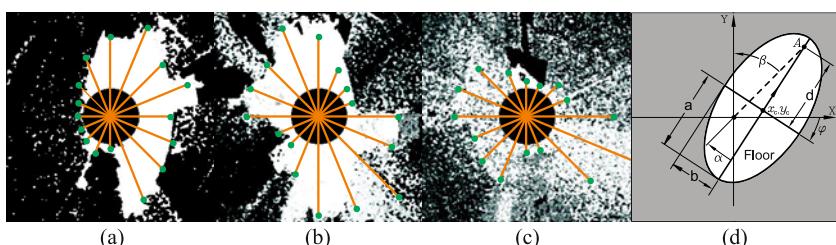


Fig. 5. Finding free floor: Scan lines represent the distance to the closest obstacle in the considered sector. (a) Good segmentation, (b) moderate segmentation and (c) poor segmentation. Extracted features: (d) Aggregated moment features.

Table 1. Top correlating features and selected features

Feature#	Feature	Regression coefficient	Correlating feature	Selected feature
17	Previous turn direction	0.71	x	x
3	X-coordinate of the elliptical centroid	0.40	x	-
15	Angle between major axis and current robot heading(β)	0.35	x	x
16	Ratio of the elliptical axis	-	-	x

with 20 bins in radial and 16 bins in angular direction. A segment is labeled as occupied in case the fraction of non-floor pixels exceeds a threshold τ . In each sector the distance to the obstacle r_{min} is given by the radius of the first non floor segment in radial direction. Figure 5 shows the mapping between segmentation and distance to obstacle for three situations with good, moderate and poor segmentation results. The inverse of the closest distance $1/r_{min}$ and the angle of the closest obstacle θ_{min} capture both the proximity of the robot to the obstacles and the direction of the corresponding obstacle. The obstacle avoidance behavior in essence turns the robot away from the direction of close by obstacles. In case of no nearby obstacles the navigation behavior attempts to center and align the robot within the corridor walls. The orientation of the corridor and the lateral displacement are indirectly inferred from the centroid and second order moments of the free space. From these moments the distribution of free space is approximated by an ellipse with a semi major axis a , semi minor axis b and orientation angle or the tilt φ as shown in Fig. 5d). Considering a look ahead point A located on the major axis at a distance d (2m) from the centroid, the angles α reflecting the lateral offset and β capturing the orientation error are computed. Furthermore to ensure continuity of avoiding motions in corners or dead end situations and to promote continuation of turns, two more features namely a flag that indicates the previous turning direction (-1, 1 and 0) and the nearest safe traversable free floor direction to the front of the robot that extends beyond a distance D are also registered. Hence the feature set is composed of eighteen moment and aggregated features in addition to the nearest obstacle features.

4 Feature Selection

The correlation between the features and the recorded curvature of the robot path are analyzed by means of linear regression analysis and the three most correlated features are listed in Table 1. This analysis provides an insight into the relative importance of the individual features but allows no conclusion on the optimal combination of features. Hence, a wrapper approach [10] with forward chaining is employed to identify the subset of features that achieves the best generalization. We employ two different learning algorithms for validating the features; instance based learning model namely locally weighted regression (LWR) and artificial neural networks (ANN) trained with Levenberg-Marquardt backpropagation (five neurons). LWR [22] is a memory based method that once queried with input features predicts the output by locally interpolating among the training samples that are most similar to the query according to a

distance metric. LWR is a nonlinear function approximation model in higher dimensional spaces. At the core of the approximation lies piecewise local linear models in a computed region of validity defined by a weighting kernel. Given a query point x_q , we employ a Gaussian kernel centered at x_q and perform regression. With forward chaining candidate features are incrementally included to the input representation, until the root mean squared error ($rmse$) of the trained model on unseen test data no longer decreases with additional features. Diagonally weighted Euclidean metric is used to compute the distance between the query point and data point. With an eager learning method such as ANN, the target function is approximated globally during training, thus requiring less memory and runtime for prediction than lazy learning methods. Initial feature selection is performed on the moment features only. Table 1 also lists the three features that result from forward chaining with LWR. It is worth noting here that the centroid feature which previously exhibited strong individual correlation with the recorded action falls out and does not figure among the features selected by forward chaining. Hence, in order to confirm the confounding factor of the centroid feature, different combinations of prospective features namely 3, 15, 16 and 17 are tested with LWR and ANN for generalization.

The training instances are validated using *hold-out cross validation* with 60% of the data used for training, 20% for validation and the remaining 20% for testing. The training, testing and the normalized root mean squared error ($nrmse$) between the prediction and the true curvature for both LWR and ANN are listed in Table 2. Normalized root mean squared error interprets the residual variance of the predicted output by dividing the prediction with the standard deviation (σ) of the target. From both the tables, one can observe that the combinations {15, 17}, {15, 16, 17} and {3, 17} generalize better than the remaining ones. The $nrmse$ of these three subsets also show a considerable improvement in the residual variance of the prediction by almost 25 – 30%. Nevertheless the difference among the top three feature sets are negligible and directly reveals that feature 16 has little or no influence on the generalization error. Therefore, we restrict the subsequent analysis to the feature sets {3, 17} and {15, 17}. In the following we consider combinations of the feature sets with the closest obstacle features ($1/r, \theta$) information obtained through three different levels of angular partition. The frontal half plane is partitioned into either two, four or eight uniform segments, resulting in either two, four or eight feature pairs of closest obstacle distance and orientation within each sector. The effect of alternative sector partitions towards the generalization of unseen data are performed and analyzed in combination with the preselected moment features. The generalization capability of the selected feature sets are validated across the entire data set, inter-scenario and intra-scenario generalizations. The inter-scenario validation tests the ability to generalize from training in one corridor (CO1) onto data taken in a second unseen corridor (CO2). The corridors share a similar geometry but different texture and appearance of the floor and walls. The intra-scenario analysis investigates as to whether the learner is able to generalize from training in the two corridor scenarios (CO) onto the fundamentally different open room scenario (OR). Since the final features for training are in fact generated using exhaustive tests on generalization error, the combination of scenario specific image features for the entire data set would further establish the final robustness of the approach. Table 3 shows the validation results with

Table 2. Feature combination and validation using LWR and ANN

Features	Training rmse [°/cm]		Testing rmse [°/cm]		nrmse	
	LWR	ANN	LWR	ANN	LWR	ANN
3 15 16 17	0.020	0.027	0.045	0.038	0.83	0.84
3 15	0.028	0.041	0.056	0.043	1.04	0.95
3 15 16	0.029	0.042	0.055	0.041	1.01	0.91
3 15 17	0.026	0.029	0.047	0.034	0.87	0.76
15 16	0.034	0.040	0.053	0.040	0.98	0.97
15 17	0.026	0.030	0.038	0.033	0.71	0.74
15 16 17	0.025	0.031	0.038	0.034	0.71	0.76
3 17	0.024	0.034	0.039	0.027	0.72	0.60

feature set {15, 17} and {3, 17} using LWR and ANN. In both cases, the ANN exhibits better performance in generalization than LWR and feature set {15, 17} approximates better than its counterpart {3, 17}. For both, the generalization error between the corridors (CO1 and CO2) is robust. This is attributed to the similarity of the training and testing instances albeit different corridors. By considering the nearest obstacle information together with the selected features, ANN tends to generalize better than LWR. With two and four sectors, the generalization exhibits similar robustness compared to tests performed without explicit obstacle features, nevertheless the error increases substantially for eight sectors. The reason for inferior performance of LWR with sector features is attributed to the lack of diverse training examples compared to the dimension of the feature space giving way to potential overfitting of the data. The validation error between corridors and open room is relatively larger with LWR mainly attributing to the lack of similar examples, nevertheless with ANN a better generalization is realized.

5 Experimental Results

The experiments are performed on a Pioneer 3DX mobile robot with the moment features {15, 17} as it exhibits better generalization and obstacle features information generated from four frontal segment partitioning (Table 3) governed by the curvature predicted by the trained ANN. The robot is allowed to wander along the corridor for 45 minutes with frequent restarts in randomized initial headings. Narrow passages and corridor dead end situations are resolved by half turns towards the free corridor. However, in the event of insufficient lighting conditions and in case of high similarity between the floor texture and obstacle color, the free space segmentation degrades causing inconsistent behavioral responses such as ongoing turns and collisions. Two typical successful scenarios are depicted in Fig. 6. Scenario (a) is an instance of a prototypical obstacle avoidance scenario. The robot is positioned in front of a pillar along side the left wall of the corridor. The depicted trajectory in the figure shows the successful circumnavigation of the obstacle and subsequent centering along the corridor. As a mode of comparison, the demonstrated sonar behavior is also performed exactly from the same starting position. As one can see from the figure, the learned behavior initiates the turn earlier towards the right in order to avoid the pillar to the left, whereas the sonar behavior goes

Table 3. Feature sets 15, 17 and 3, 17. Locally weighted regression (LWR) and Artificial Neural Networks (ANN).

Feature set	Training	Testing	# sector	Training rmse[°/cm]		Testing rmse[°/cm]	
				LWR	ANN	LWR	ANN
15, 17	CO1	CO2	-	0.0250	0.0318	0.0375	0.0285
15, 17	CO1	CO2	2	0.0243	0.0303	0.0378	0.0295
15, 17	CO1	CO2	4	0.0243	0.0266	0.0379	0.0335
15, 17	CO1	CO2	8	0.0229	0.0288	0.0369	0.0264
15, 17	CO	OR	-	0.0251	0.0315	0.0446	0.0409
15, 17	CO	OR	2	0.0244	0.0308	0.0531	0.0440
15, 17	CO	OR	4	0.0238	0.0272	0.0742	0.0436
15, 17	CO	OR	8	0.0233	0.0233	0.0604	0.0440
15, 17	Full	Full	-	0.0288	0.0416	0.0449	0.0449
15, 17	Full	Full	2	0.0286	0.0321	0.0463	0.0398
15, 17	Full	Full	4	0.0283	0.0313	0.0434	0.0352
15, 17	Full	Full	8	0.0270	0.0313	0.0493	0.0302
3, 17	CO1	CO2	-	0.0237	0.0320	0.0406	0.0322
3, 17	CO1	CO2	2	0.0240	0.0326	0.0455	0.0266
3, 17	CO1	CO2	4	0.0226	0.0254	0.0401	0.0350
3, 17	CO1	CO2	8	0.0211	0.0257	0.0397	0.0224
3, 17	CO	OR	-	0.0237	0.0349	0.0550	0.0427
3, 17	CO	OR	2	0.0234	0.0284	0.0618	0.0435
3, 17	CO	OR	4	0.0223	0.0294	0.0656	0.0441
3, 17	CO	OR	8	0.0222	0.0236	0.0638	0.0442
3, 17	Full	Full	-	0.0294	0.0326	0.0521	0.0355
3, 17	Full	Full	2	0.0285	0.0332	0.0527	0.0344
3, 17	Full	Full	4	0.0265	0.0350	0.0530	0.0315
3, 17	Full	Full	8	0.0265	0.0278	0.0543	0.0345

straight till it encounters the pillar and then initiates a turn. Figure 6(b) shows a scenario in which the robots initial heading is misaligned with the corridor facing a wall. The scenario is a good task to test the behaviors ability to gauge both the learned corridor centering and obstacle avoidance properly. The demonstrated behavior as programmed

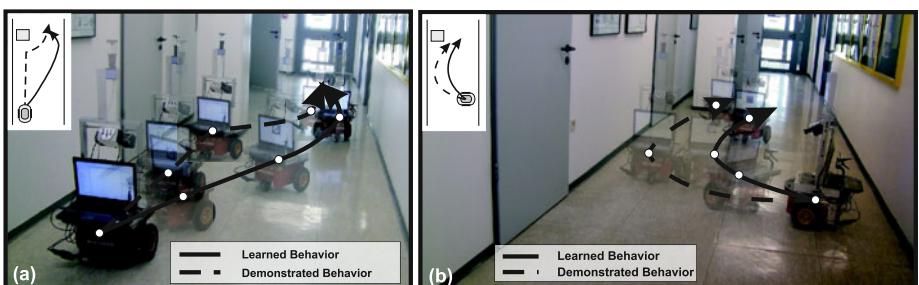


Fig. 6. (a) Obstacle avoidance situation. Demonstration behavior vs. learned behavior, (b) Misalignment in the corridor. Demonstration behavior vs. learned behavior

first approaches the the opposite wall and starts to turn relatively late. The learned behavior reacts earlier and achieves alignment of the robots heading with the corridor direction faster thereby surprisingly optimizing the demonstrated behavior through the diversity of the training examples. The experiments demonstrate that the selected visual features are sufficient to mimic the demonstration in a robust and effective manner.

6 Conclusion

This paper presented a novel framework for programming visual reactive robotic behaviors from demonstration. Our approach tackles the problem to generate and extract aggregated visual features that allow to learn a meaningful relationship between perception and action by demonstration. The method is useful to transfer existing, well established mobile robot behaviors that rely on a different sensor modality onto a vision based behavior. The approach provides a possible solution to the problem of finding a suitable record mapping in the context of vision based learning from demonstration, such that the visual features used for learning capture those aspects relevant for the decisions of the demonstrator. Exhaustive tests are performed to confirm the robustness of the approach and validated using LWR and ANN. From the validation, we concur that ANN generalizes marginally better than LWR. Further, the experimental results demonstrate the ability of the learned behaviors to generalize in different situations. The trajectory of the learned behavior is also compared with the demonstrated non-visual behavior to establish the ability of the learned behavior to replicate and at times optimize from demonstrated examples. In the future, the next steps of the general robotic navigation framework will be investigated with human demonstration examples possessing a larger degree of ambiguity and exhibiting scenario awareness and ultimately rely on the appearance of the environment rather than the geometry.

References

1. Argall, B., Chernova, S., Veloso, M., Browning, B.: A survey of robot learning from demonstration. *Robotics and Autonomous Systems* 57(5), 469–483 (2009)
2. Atkeson, C., Moore, A., Schaal, S.: Locally weighted learning. *AI Review* 11, 11–73 (1997)
3. Billard, A., Calinon, S., Dillmann, R., Schaal, S.: Robot programming by demonstration. In: *Springer Handbook of Robotics*, pp. 1371–1394 (2008)
4. Bishop, C.: *Neural Networks for Pattern Recognition*, 1st edn. Oxford University Press, USA (1996)
5. Bonin-Font, F., Ortiz, A., Oliver, G.: Visual navigation for mobile robots: A survey. *J. Intell. Robotics Syst.* 53(3), 263–296 (2008)
6. Dillmann, R., Rogalla, O., Ehrenmann, M., Zöllner, R., Bordegoni, M.: Learning robot behaviour and skills based on human demonstration and advice: the machine learning paradigm. In: 9th Intl. Symp. of Robotics Research (ISRR), pp. 229–238 (1999)
7. Ekwall, S., Krägic, D.: Robot learning from demonstration: a task-level planning approach. *Intl. J. Advanced Robotic Syst.* (2008)
8. Gaspar, J., Winters, N., Santos-Victor, J.: Vision-based navigation and environmental representations with an omnidirectional camera. *IEEE Transactions on Robotics and Automation* 16, 890–898 (2000)

9. Hähnel, D., Burgard, W., Thrun, S.: Learning compact 3d models of indoor and outdoor environments with a mobile robot. *Robotics and Autonomous Systems* 44(1), 15–27 (2003)
10. Kohavi, R., John, G.: Wrappers for feature subset selection. *Artificial Intelligence* 97(1-2), 273–324 (1997)
11. Lee, D., Hebert, M., Kanade, T.: Geometric reasoning for single image structure recovery. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR (2009)
12. Lenser, S., Veloso, M.: Visual sonar: fast obstacle avoidance using monocular vision. In: Proc. of IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS), vol. 1, pp. 886–891 (2003)
13. Martin, C.: The Simulated Evolution of Robot Perception. Ph.D. thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA (2001)
14. Mitchell, T.: Machine Learning. McGraw-Hill, New York (1997)
15. Narayanan, K., Posada, L., Hoffmann, F., Bertram, T.: Imitation learning for visual robotic behaviors. In: Proceedings of 19th Workshop Computational Intelligence, pp. 221–236 (2009)
16. Nüchter, A., Hertzberg, J.: Towards semantic maps for mobile robots. *Robot. Auton. Syst.* 56(11), 915–926 (2008)
17. Nüchter, A., Lingemann, K., Hertzberg, J., Surmann, H.: 6d slam—3d mapping outdoor environments: Research articles. *J. Field Robot.* 24(8-9), 699–722 (2007)
18. Peula, J., Urdiales, C., Herrero, I., Sánchez-Tato, I., Sandoval, F.: Pure reactive behavior learning using case based reasoning for a vision based 4-legged robot. *Robotics and Autonomous System* 57(6-7), 688–699 (2009)
19. Poppinga, J., Vaskevicius, N., Birk, A., Pathak, K.: Fast plane detection and polygonalization in noisy 3d range images. In: Proc. of IEEE/RSJ Intl. Conference on Intelligent Robots and Systems (IROS), pp. 3378–3383 (2008)
20. Posada, L., Narayanan, K., Hoffmann, F., Bertram, T.: Floor segmentation of omnidirectional images for mobile robot visual navigation. In: Proc. of IEEE/RSJ Intl. Conference on Intelligent Robots and Systems, IROS (2010)
21. Schaal, S.: Learning from demonstration. In: Advances in Neural Information Processing Systems, vol. 9, pp. 1040–1046. MIT Press, Cambridge (1997)
22. Schaal, S., Atkeson, C., Vijayakumar, S.: Scalable techniques from nonparametric statistics for real time robot learning. *Appl. Intell.* 17(1), 49–60 (2002)
23. Sofman, B., Lin, E., Bagnell, J., Vandapel, N., Stentz, A.: Improving robot navigation through self-supervised online learning. In: Proceedings of Robotics: Science and Systems, Philadelphia, USA (2006)
24. Ulrich, I., Nourbakhsh, I.: Appearance-based obstacle detection with monocular color vision. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, pp. 866–871. AAAI Press / The MIT Press (2000)
25. Vlachos, M., Kollios, G., Gunopulos, D.: Elastic translation invariant matching of trajectories. *Machine Learning* 58(2-3), 301–334 (2005)
26. Yoo, T., Oh, I.: A fast algorithm for tracking human faces based on chromatic histograms. *Pattern Recognition Letters* 20(10), 967–978 (1999)

Design Principles of the Component-Based Robot Software Framework Fawkes

Tim Niemueller¹, Alexander Ferrein², Daniel Beck¹, and Gerhard Lakemeyer¹

¹ Knowledge-based Systems Group

RWTH Aachen University, Aachen, Germany

{niemueller,beck,gerhard}@kbsg.rwth-aachen.de

² Robotics and Agents Research Lab

University of Cape Town, Cape Town, South Africa

alexander.ferrein@uct.ac.za

Abstract. The idea of component-based software engineering was proposed more than 40 years ago, yet only few robotics software frameworks follow these ideas. The main problem with robotics software usually is that it runs on a particular platform and transferring source code to another platform is crucial. In this paper, we present our software framework Fawkes which follows the component-based software design paradigm by featuring a clear component concept with well-defined communication interfaces. We deployed Fawkes on several different robot platforms ranging from service robots to biped soccer robots. Following the component concept with clearly defined communication interfaces shows great benefit when porting robot software from one robot to the other. Fawkes comes with a number of useful plugins for tasks like timing, logging, data visualization, software configuration, and even high-level decision making. These make it particularly easy to create and to debug productive code, shortening the typical development cycle for robot software.

1 Introduction

The idea of component-based software engineering (CBSE) dates back more than 40 years with McIlroy's demand to create reusable software entities [1]. It follows three main ideas: (1) develop software from pre-produced part, (2) reuse software in other applications, and (3) be able to maintain and customize parts of the software to produce new functions and features [2].

In [3], Brugali et al. discuss current trends in component-based robotic software. Component-based robotic software deals with applying the principles of component-based software engineering to robotic control software. Following [4], they require four properties to be fulfilled for a robot software component suiting the component-based approach: (1) a component is a binary (non-source-code) unit of deployment, (2) a component implements (one or more) well-defined interfaces, (3) a component provides access to an inter-related set of functionalities, and (4) a component may have its behavior customized in well-defined

manners without access to the source code. In [3], the authors conclude that in a component-based robotic software (CBRS) system, the used component model allows for observing and controlling the internal behavior of the component, and that each component has well-defined *interfaces* and *data structures* from which the usage modalities follow. Moreover, a CBRS requires *communication patterns* that enables the *interconnection of reusable components*, while the level of abstraction of the communication infrastructure must be such that *heterogeneous components* can be connected. Finally, they demand for *component repositories* which simplify documentation, retrieval, and deployment of a large number of reusable components. Examples for systems which, at least, partially fulfill these requirements, as stated by the authors are, for instance, [5,6,7,8].

In this paper, we propose the Fawkes robot software framework (RSF) which follows the component-based approach. The aim is to provide a software environment which cuts short the development times for robot software components. Our target platforms are wheeled service robots as well as humanoid soccer robots. After an intensive study of available RSFs, we saw the need to develop a component-based framework which could be used cross-platform for all of our robots. Even frameworks like ORCA [8,9] which follow the same software paradigm, did not meet our demands, as we lay out below.

The key features of Fawkes are: (1) a well-defined component concept, (2) a hybrid blackboard/messaging infrastructure for communication, (3) well-defined interfaces, (4) run-time loadable plugin mechanisms inheriting certain predefined aspects such as communication, configuration, logging, timing or integrating computer vision and networking (following the aspect-oriented software design), (5) utilizing multi-core computation facilities by deploying POSIX threads for plugins, and (6) a network infrastructure for communicating with remote software entities.

In the following, we present the Fawkes software framework. In particular, we review recent existing approaches and related work in Section 2 before we state in Section 3 what the design criteria are that we based our work on. In Section 4 we describe the concepts of Fawkes in detail. We present the general interface structure, the communication middle-ware, software engineering concepts such as guarantees, and show how the main application can be distributed. We want to remark that Fawkes comes with a behavior engine [10], but is not limited to the built-in one. In Section 5 we present the application of Fawkes on our different robot systems. We conclude with Section 6.

2 Other Robot Software Frameworks

Several other robotics frameworks and middle-wares for robots exist. Here, we want to give a brief overview of the most recent and related ones. Amongst them, the Open Robot Control Software [11], Orocovis for short, is driven by the desire to standardize robotics software components and especially their interconnection. The aim is to provide a comprehensive framework for robotics applications. For communication, Orocovis deploys CORBA. Several libraries for robotics applications are available for Orocovis. The basic framework is defined in the Real-Time

Toolkit, further, there are libraries for Bayesian filtering, robot kinematics and dynamics. A spin-off of Orocos is Orca [9]. Orca provides a communication infrastructure that uses ICE instead of CORBA, which is a successor development of CORBA with a simplified API and where redundant features were left out. Orca comes with drivers for a wide range of common hardware and solutions for basic robotics problems such as path planning. Software components are run as different processes which communicate with each other. However, it seems that Orca as a framework is lacking coherence, as the overall design is only loosely defined and fragmented into several pieces. The main benefits of using a CORBA-based middle-ware is the possibility to use a variety of programming languages and the ability to distribute computations over many hosts transparently. For mobile robots, however, most if not all of the computation has to be done on the robot. Therefore it does not seem desirable to have a purely network-based communication framework for these application domains. Moreover, some general criticism on the use of CORBA states [12]: “First and foremost the API is complex, inconsistent, and downright arcane and writing any non-trivial CORBA application [is] surprisingly difficult.” Other technologies such as SOAP or XML-RPC are used as replacements, especially because of their simplicity.

Another recent and popular example for a robot software framework is the Robot Operating System (ROS) [13]. The idea behind ROS was to develop a framework for large-scale service robot applications. ROS offers a communication infrastructure which for one employs peer-to-peer messaging and for the other provides remote procedure calls (RPC). It uses a central broker for service registration and discovery. For one ROS processes, called nodes, directly communicate with each other by exchanging messages following a topic-based publisher/subscriber philosophy. In contrast to this, so-called service communication allows for strict request and answer message passing. In these messages, primitive data types and structures are allowed which can be composed to more complex messages. Communication with the broker and between nodes for communication negotiation employs XML-RPC. Topic communication and RPC is implemented as a custom protocol over TCP or UDP. Besides the basic communication services, ROS offers specialized packages such as predefined message ontologies (e.g. for navigation or sensor data), or data filters. To facilitate the programming with ROS, APIs for C++ and Python are offered. For ROS, a large number of third-party libraries for standard tasks such as localization or navigation is available. Although the ROS approach looks promising and seems to be very general, the different design criteria suggest Fawkes as a viable alternative. Most notably, ROS conveys systems of only loosely federated nodes, while Fawkes emphasizes a closely integrated system. This is beneficial for synchronization of different tasks, low latencies, and efficiency from embedded systems to multi-machine service robots. ROS and Fawkes share the idea of a component-based design, but in Fawkes this is set as a priority, e.g. by emphasizing re-use of well-known interfaces as much as possible, while in ROS it is common for each module to define its own message types.

3 Terminology and Design Criteria

In this section we present the design criteria for Fawkes. As our conceptual design follows the component-based software paradigm, we start with introducing the terminology used by this paradigm in Sect. 3.1 before we review some characteristics of robot software frameworks in Sect. 3.2. Finally, in Sect. 3.3 we overview our design goals.

3.1 Terminology

According to Szyperski [2], different forms of reuse are possible at the design level. He distinguishes between (1) sharing programming/scripting languages, (2) libraries, (3) interfaces, (4) messages/protocols, (5) patterns, (6) frameworks, and (7) system architectures. In the following, we briefly introduce the basic terms and address the different sharing options.

A *component* is defined as a binary unit of deployment that implements one or more well-defined interfaces to provide access to an inter-related set of functionalities, configurable without access to the source code [8]. It adheres to a specified “contract” and expects certain input data and produces and provides specified output data. The contract also states what the component expects from its context, e.g. certain timing constraints. If the requirements posed by a certain system infrastructure are not met by an interface, it has to be wrapped accordingly. A *module or library* is a coherent set of implemented functionalities, using an object-oriented data encapsulation as a useful (but not necessary) design paradigm. A module can be compiled separately, and is portable to different platforms given compatible compiler and operating system support. Modules inherently hardware-dependent, such as device drivers, are often not portable [9]. The *system architecture* is a specific choice of functional building blocks (*components*), in order to build a software system that performs according to a specification [1]. [2] basically distinguishes between (strict) layered architectures and the (strict) onion model. The disadvantage of strict layered architectures is that the extensibility of the system is restricted. A *framework* is a design and an implementation providing a possible solution in a specific problem domain. It is used to model a particular domain or an important aspect thereof. Frameworks are similar to software libraries, which means they are reusable abstractions of code wrapped in a well-defined API. Unlike libraries, however, the overall program’s flow of control is not dictated by the caller, but by the framework. This inversion of control is a distinguishing feature of software frameworks [15]. Finally, a *component architecture* determines the internal design of one single component in order to guarantee that the component performs according to its external interface/contract.

3.2 Characteristics of a Mobile Robot Software Framework

There exists a plethora of different robot frameworks today. On the one hand, different approaches how a robot should be controlled emerged and on the other

hand, different methods how modules are combined and connected were developed during the last decades. In [16], certain characteristics are defined for the evaluation of software frameworks that we summarize and extend below:

- *Robot hardware abstraction.* The framework should not be tailor-made for a specific robot platform, but it should rather be portable to a variety of platforms.
- *Extensibility and scalability.* The robot framework must be able to easily incorporate new software modules and to use hardware newly added to the robot.
- *Limited run-time overhead.* The run-time overhead can be measured in terms of memory and CPU requirements, i.e. the frequency by which the control loops are executed, and end-to-end latency, meaning the time that is required for a sensor reading to have an effect on the actuator command.
- *Actuator control model.* The actuator control model is twofold. For one, the overall structure of a robot software system imposes a certain preferred model of control (deliberative vs. reactive), for another different actuator control models must be available depending on the type of actuator.
- *Software characteristics.* Software for mobile robots shares the common requirements for good software, like completeness, simplicity, correctness, and consistency.
- *Tools and methods.* The complexity of robotics tasks demands for data introspection, monitoring, or debugging tools provided by the framework in order to allow efficient software development.
- *Documentation.* To achieve wide acceptance for a software platform rigorous documentation is crucial and has to be provided in forms of reference manuals and API documentation.

3.3 Design Goals of Fawkes

Inspired by the component-based software design paradigm, with Fawkes we tried to combine these with the requirements of a robot software framework. Moreover, we were influenced by our personal experience of having programmed robots in service robotics and robotic soccer domains for nearly a decade. Our goal was to design a software framework that is as flexible as possible and portable to our various robot platforms ranging from domestic service robots to biped soccer robots. Across the different platforms a consistent environment must be provided. The software must scale from embedded systems to multi-machine robotic applications. The number of concepts that a user needs to know should be as few as possible to minimize the overhead to get familiar with the framework. Finally, the run-time overhead needed to be as small as possible as we wanted to deploy Fawkes in real-time domains such as robotic soccer. The framework must be extensible to add new functionality over time.

In a nutshell, the framework has the following features. Each functional entity in Fawkes is a component, which can make use of several predefined aspects. Each component, implemented as a plugin that can be loaded at run-time, needs

to inherit a communication aspect to communicate with other components. Plugins are realized as threads in a monolithic program. However, distributed design is possible by synchronizing the data between multiple instances via a network protocol with a minimal timing overhead. Via synchronization among the components we ensure that no superfluous work is performed.

4 The Fawkes Robot Software Framework

In the following, we describe the Fawkes framework in detail. In order to do so, we start by showing the general structure of a Fawkes application in Sect. 4.1, and derive from there the different properties which qualifies Fawkes to be a component-based design. In particular in Section 4.2, we show the interface design and the communication infrastructure. Section 4.3 overviews the predefined software patterns that comes with Fawkes, so-called aspects. These predefined aspects allow for introducing the concept of guarantees, which provide some basic guarantees of the quality of service of the application.

4.1 The Framework Architecture

Following the component-based approach, we define components in Fawkes as logical elements. They manifest in the form of plugins. Generally, a single plugin implements one component.¹ The Fawkes core application only provides the basic infrastructure. The most important elements of that infrastructure are a central blackboard used for exchanging data between plugins, centralized logging facilities that allow for parallel logging to multiple logging targets (i.e., console output, log-files, etc.), and a centralized configuration database that is intended to store the configuration parameters for all components of the system. Moreover, it provides a default implementation of a main loop that might be replaced with a custom main loop at run-time. Generally, the main loop controls the execution order of the threads and ensures that certain timing criteria are met. Each run of the main loop is fractured into multiple stages; the default implementation represents a refined *sense-think-act* cycle: in the first stages in the loop the threads acquiring new data from the robot's sensors are run, afterwards the threads implementing deliberative or reactive decision-making components, and lastly the threads which send commands to the actual hardware.

The actual functionality that makes an arbitrary framework a robot software system is provided by plugins. The plugins are implemented as dynamically loadable libraries—shared objects on Linux systems. They implement a particular interface which gives access to descriptive and dependency information and a set of threads. Plugins can be loaded and unloaded at run-time. This allows for a fast development cycle. Usually a developer works on one plugin at a time. With the ability to reload only this plugin the program-compile-test cycle can be quicker, because only the changed plugin has to be reloaded, not the whole system.

¹ There are situations where it is useful to combine multiple components into a single plugin for efficiency or direct synchronization, which is supported by the framework.

Threads are one of the key elements of the Fawkes RSF. With the advent of modern multi-core CPUs it is considerably worthwhile to provide simple ways to exploit the multi-processing capabilities. With the decision to make every functional entity of plugins a thread, it is reasonably easy to exploit this feature. Threading is implemented based on the POSIX1 Threads API. They can operate in two different modes, either in *continuous* or in *wait-for-wakeup* mode. In the continuous mode, the thread runs all the time, until it exits or is terminated by another thread. In the wait-for-wakeup mode, the thread blocks, until it is woken up. When woken up, it executes a single iteration. Precisely, a plugin in wait-for-wakeup mode registers for a certain stage provided by the main loop. When all threads of the current stage in the main loop have finished their iteration (or if certain timing constraints are not met, i.e., a plugin runs longer than it is supposed to) the main loop proceeds to the next stage and wakes up all plugins registered for that stage. Note, the plugins registered for the same stage in the main loop run concurrently.

4.2 Interface Design and Communication Infrastructure

As already mentioned above the Fawkes core provides a blackboard that serves as a centralized storage unit that plugins can read data from and write data to (cf. Fig. II). The access to the data stored in the blackboard is managed using instances of interfaces whose types are known to all plugins. An interface defines a number of fields; the definition of each field consists of a basic data type and an unique identifier. For instance, an interface of type *Position2D* may contain fields of type *float* with the names *x* and *y*. To facilitate the design, tool support for creating and maintaining interfaces exists.

The plugins may request to open an interface of a certain type with a certain (unique) identifier at run-time. Additionally, the plugin may either open the instance as a writer or as a reader. Whereas multiple readers may attach to an instance of an interface, only a single writer is allowed to do so. This ensures the integrity of the data stored in the blackboard. With saying a plugin “opens an interface”, we actually mean that it is first checked whether an instance of that type of interface with the given identifier already exists in the blackboard. In that case, a proxy of the interface in the blackboard, i.e. a local copy (outside of the blackboard), is created for the plugin. In case no instance of an interface

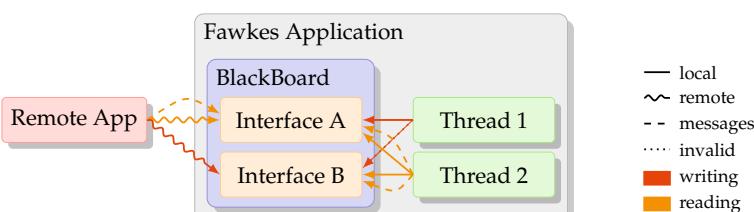


Fig. 1. Accessing the Fawkes blackboard

conforming to the request of the plugin exists the necessary memory is allocated by the blackboard first. The plugins can synchronize their proxies with the original in the blackboard either by copying the current data from the blackboard to the proxy or the other way around.

This reader-writer model allows to pass data from the writer to the reader. Readers may send commands to the writer of an interface instance by means of messages. Associated with each writing instance of an interface is a message queue that stores the messages sent by the readers in the order they have been sent and thus allows the writer to process the messages in the correct chronological order. Although neglected above, a definition of message types accepted by an interface is also part of the interface definition besides the definition of the data fields provided by an interface.

4.3 Aspects and Soft Guarantees

Plugins need to access features provided by the Fawkes framework, for instance, accessing the blackboard. In order to reduce the implementation effort we borrowed ideas from aspect-oriented programming. A so-called *aspect* denotes a specific ability or requirement. Now, when a thread of a plugin wants to make use of such an ability, it is “given that aspect”. In our C++ implementation this means that we have a class implementing each aspect; a thread “is given an aspect” by inheriting from that class. In a sense, we lift the classic aspect-oriented programming paradigm on the framework level and aspects assert concerns of threads regarding particular framework functions like centralized logging.

The framework allows to specify dependencies between different aspects. For instance, there is the *vision-master* aspect that is given to threads that provide access to cameras; the *vision* aspect is given to threads that need to access the images captured by the cameras. A modeled one-to-many dependency guarantees that a plugin which has at least one thread with a *vision* aspect can only be started when another having the *vision-master* aspect is already running; as long as at least one thread with a *vision* aspect is running it is not allowed to unload the plugin which owns the thread having the *vision-master* aspect.

Besides the *vision-master*- and *vision* aspect, the Fawkes framework implements aspects (among others) which allow to access the blackboard, the central configuration database, and the centralized logging facilities. Furthermore, there is a central clock in the Fawkes framework (accessible via the *clock* aspect). The *time-source* aspect allows threads to provide a proprietary clock which especially comes in handy when working with a simulation which might not run at real-time. In such a case a thread with the *time-source* aspect can provide the simulation time to all threads that have the *clock* aspect via the central clock.

Above we already mentioned a couple of (soft) guarantees. The idea behind those guarantees is that they provide a simple exception handling mechanism on the framework level. For example with the framework knowing the requirements of threads because of its aspects it can ensure that all requirements and dependencies are fulfilled. These guarantees are called “soft” because they are not checked all the time but only at certain moments. Dependencies, for instance, are only checked during initialization/finalization of the respective threads but not in between.

5 Evaluation and Case Studies

In this section evaluate the framework by the characteristics provided in Section 3.2 and show two configurations of Fawkes in different application domains. We start with our service robots and show how legacy software from our previous software framework is intertwined with Fawkes. When integrating legacy software, the component-based approach clearly pays off. The second domain is the robotic soccer domain. In this domain we participate in RoboCup competitions in the Middle-size league with wheeled robots, and in the Standard Platform League with the biped humanoid robot Nao.

5.1 Evaluation of Framework Characteristics of Fawkes

As mentioned in Section 3.2, certain characteristics can be used for a qualitative evaluation of RSFs. In this section we apply these to Fawkes.

Hardware abstraction is accomplished by encapsulating hardware-specific functions into separate plugins. Data is shared and commands are sent via the blackboard. Therefore, adapting to a new platform is done by implementing and loading the set of plugins that matches the chosen hardware. *Extensibility* is accomplished by the component-based approach. This makes it easy to use existing components and add new ones. For Fawkes *scalability* is especially targeted towards singular machines assuming all computation is done on the robot. Since Fawkes makes use of multi-threading at its core, there is a high potential for exploiting today's multi-core machines. Experiments suggest that the *run-time overhead* of the base system in relation to the functional plugins can be considered negligible. High frequency main loop iterations are achieved as long as the individual threads are bounded appropriately in time. In the future, a closer comparison to ROS might be performed. The *actuator control model* that is implemented by a plugin can be freely chosen. For the overall framework the hybrid deliberative-reactive paradigm is presumed. *Tool support* is very important when developing software in general, and for complex systems like a robot in particular. We have carefully designed the software to interact well with debuggers and performance analysis tools. *Documentation* of all public APIs is enforced in the development process. A wiki provides usage and developer documentation.

The Fawkes RSF is complete in the sense that it provides the *basic infrastructure to implement and interconnect a set of components* to control a certain robot. For a particular platform and domain the applicable components must be developed and integrated. The *correctness* is not automatically verified. For several parts of the software quality assurance applications have been written, that can be used to manually test parts of the software. To achieve *consistency* functional blocks have been bundled into appropriate libraries and well-known interfaces are used for communication wherever feasible.

5.2 Service Robots

Our service robot employs a differential drive; it is equipped with 360° laser range finder, a 6 DOF robotic arm, and a stereo camera. Additionally to two computers

in the base (900 MHz Pentium III) which handle localization, navigation, and processing of the laser distance readings we added another computer (2 GHz Core 2 Duo) that runs vision applications (object and face recognition), speech recognition, people tracker, and the control software for the robotic arm and the pan-tilt unit on which the stereo camera is mounted. As such this robot is a multi-node system on which a multitude of (partially) quite demanding applications is run.

Fawkes runs distributed on the three machines, and data and commands are transferred between the machines using the remote blackboard mechanisms provided by the framework. A specialty is that we integrated several applications developed for our old framework into the system (localization and navigation). This could be easily accomplished by extending the “old” applications by means of small adapters to exchange data and commands with Fawkes applications, again, using the remote blackboard mechanism. This shows that third-party software can be easily integrated into Fawkes and that even two frameworks that build on different design concepts can be used side-by-side. It has to be noted, though, that applications which are integrated in such a way are not equivalent to Fawkes plugins as their execution is not synchronized via the main loop, external applications cannot be given any aspects, and guarantees cannot be made for such external applications. These are no limitations of the Fawkes framework but a tribute one needs to pay due to different design principles underlying different frameworks. In our case, for example, the applications from the old framework handled their timing on their own. Consequently, it is not possible to synchronize them with the other plugins without major modifications which basically coincide with re-implementing the old applications as Fawkes plugins.

5.3 Humanoid and Wheeled Soccer Robots

Figure 2 gives an overview of the components and interfaces that are running on our wheeled soccer robots. On the left-hand side of the figure, hardware components can be found such as driver for the kicker mechanism (*kicker*),

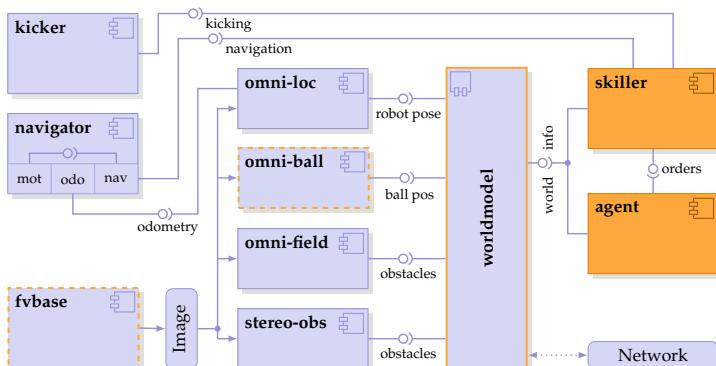


Fig. 2. Fawkes components of our wheeled soccer robot

the motors (*navigator*), or the camera (*fbase*). The kicker serves the kicking interface, while the navigation component fills the odometry and the navigation interfaces. The camera drivers (we have two camera systems mounted on the platform) provide images in a special interface. These images are used by several medium-level components, such as the localization with the installed omni-visual camera (*omni-loc*), the ball detection (*omni-ball*), and the obstacle detection (*omni-field* and *stereo-obs*). All these modules feed their particular interfaces which are amalgamated into a consistent world model. Note that the world model also has a link to the robot’s network to integrate data from other robots. The world model in turn feeds the *world info* interface which is used by the high-level decision making (the *skiller* to execute and monitor basic actions, and the *agent* to take decisions which action to perform). The architecture layout for our biped soccer robot Nao looks similar, although the platform is very different.

6 Conclusion

In this paper, we presented the Fawkes RSF and its design principles, which follow the component-based software paradigm. This paradigm turned out to be very beneficial, as we applied Fawkes to several of our robot platforms and the component-based approach facilitated this endeavor. Our evaluation and the examples in Sect. 5 suggest that the framework meets the requirements outlined in Sect. 3.2. It can be applied to different hardware systems, benefits from the component-based approach for its extensibility and is scalable across multiple machines. Experiments showed a negligible run-time overhead of the framework. We omitted figures here, but the fact that we deploy Fawkes on very different platforms such as our wheeled soccer robots or the Nao, which in particular has restricted computing facilities, shows that the run-time overhead is feasible. The project’s website can be found at <http://www.fawkesrobotics.org>

For our future work, we need to get even more experience with applying Fawkes to different platforms, acquiring more scalability and run-time overhead results, and with offering more sensor and actuator plugins for common robot hardware, in order to serve a broader user community. Other areas of future work are to interface with ROS, as the Robot Operating System is widely used by number of groups. Finally, we want to offer plugins for high-level reasoning. In particular, we are working on a lightweight implementation of Golog [17] and an interface between Golog and Fawkes.

Acknowledgments

A. Ferrein is currently a Feodor Lynen fellow supported by a grant of the Alexander von Humboldt Foundation. T. Niemueller was partly supported by the German National Science Foundation (DFG). We thank the anonymous reviewers for their helpful comments.

References

1. Mcilroy, M.D.: 'mass produced' software components. In: Software Engineering: Report On a Conference Sponsored by the NATO Science Committee, pp. 138–155 (1968)
2. Szyperski, C.: Component Software – Beyond Object-oriented Programming. Addison-Wesley, Reading (2002)
3. Brugali, D., Brooks, A., Cowley, A., Côté, C., Domínguez-Brito, A.C., Létourneau, D., Michaud, F., Schlegel, C.: Trends in robot software domain engineering. In: Brugali, D. (ed.) Software Engineering for Experimental Robotics. Springer Tracts in Advanced Robotics, vol. 30, pp. 135–142. Springer, Heidelberg (2007)
4. Collins-Cope, M.: Component based development and advanced OO design. Technical report, Ratio Group Ltd. (2001), <http://www.markcollinscope.info/W7.html>
5. Chaimowicz, L., Cowley, A., Sabella, V., Taylor, C.: Roci: a distributed framework for multi-robot perception and control. In: Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 266–271 (2003)
6. Dominguez-Brito, A., Hernandez-Sosa, D., Isern-Gonzalez, J., Cabrera-Gamez, J.: Component software in robotics. In: Proceedings of the 2004 2nd International IEEE Conference on Intelligent Systems, vol. 2, pp. 560–565. IEEE Press, Los Alamitos (2004)
7. Côté, C., Brosseau, Y., Létourneau, D., Raïevsky, C., Michaud, F.: Robotic software integration using marie. International Journal of Advanced Robotic Systems 3(1), 55–60 (2006)
8. Brooks, A., Kaupp, T., Makarenko, A., Williams, S., Orebäck, A.: Towards component-based robotics. In: Proc. IROS 2005, pp. 163–168. IEEE Press, Los Alamitos (2005)
9. Brooks, A., Kaupp, T., Makarenko, A., Williams, S., Oreback, A.: Orca: a component model and repository. In: Brugali, D. (ed.) Software Engineering for Experimental Robot, pp. 231–251. Springer, Heidelberg (2007)
10. Niemüller, T., Ferrein, A., Lakemeyer, G.: A Lua-based Behavior Engine for Controlling the Humanoid Robot Nao. In: Baltes, J., Lagoudakis, M.G., Naruse, T., Ghidary, S.S. (eds.) RoboCup 2009: Robot Soccer World Cup XIII. LNCS, vol. 5949, pp. 240–251. Springer, Heidelberg (2010)
11. Soetens, P.: A Software Framework for Real-Time and Distributed Robot and Machine Control. PhD thesis, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium (May 2006)
12. Henning, M.: The rise and fall of corba. Queue 4(5), 28–34 (2006)
13. Quigley, M., Conley, K., Gerkey, B., Faust, J., Leibs, T.B.F.J., Wheeler, R., Ng, A.Y.: ROS: an open-source Robot Operating System. In: Proc. of the Open-Source Software Workshop at the International Conference on Robotics and Automation (2009)
14. Mowbray, T.J.: Architecture in the large. Object Mag. 7(10), 24–26 (1997)
15. Riehle, D.: Framework Design – A Role Modeling Approach. PhD thesis, ETH Zürich (2000)
16. Orebäck, A., Christensen, H.: Evaluation of architectures for mobile robotics. Autonomous Robots 13(1), 33–49 (2003)
17. Levesque, H., Reiter, R., Lespérance, Y., Lin, F., Scherl, R.: GOLOG: A Logic Programming Language for Dynamic Domains. J. of Logic Programming 31 (1997)

Handling Hardware Heterogeneity through Rich Interfaces in a Component Model for Autonomous Robotics

Olena Rogovchenko and Jacques Malenfant

Université Pierre et Marie Curie-Paris 6, CNRS, UMR 7606 LIP6,
4 Place Jussieu, F-75252 Paris Cedex 05, France
`{Olena.Rogovchenko,Jacques.Malenfant}@lip6.fr`

Abstract. Coping with hardware heterogeneity is one of the hardest challenges in robotic software. Making software architectures capable of adapting to a large spectrum of rapidly changing hardware components, such as sensors and actuators, is indeed crucial. This paper proposes to address the challenge by abstracting components from resources through *resource interfaces*, that deal with the functional and non-functional constraints on resources, such as their calling protocol, minimal or maximal frequencies of activation, range of domain values, physical constraints (envelope, power consumption), etc. The key contribution of this paper is therefore to capture the requirements of the software architectures and the characteristics of the hardware platform as high level constraints and then to check their compatibility and find appropriate configuration settings by solving this set of constraints. To offer an extensible framework, our current implementation relies on a general constraint solver to check these first at composition and then at deployment times.

1 Introduction

In recent papers [12], we have proposed a component model for autonomous robot control architectures where composability and correctness-by-construction are the major focus. This model leverages the concept of rich interfaces [3] to represent all the necessary and sufficient information to verify the compatibility and compose the components independently of their implementation. In our previous work, rich interfaces have been used essentially to represent precedence and time constraints, as well as the available processors and their interconnection in the underlying platform to produce correct assemblies with regard to scheduling and synchronization. Yet the concept of rich interfaces need not be restricted to time constraints. Our goal is to capture all the facets of component composition within a unique general setting of constraint solving.

This paper extends the set of rich interfaces of our model [2] to include resource requirements. Reactive components in control architectures implement the real-time part of sensori-motor behaviors directly connected to hardware resources such as sensors and actuators. Hardware components have operational

constraints, such as maximal sensor reading frequencies, that directly impact the software. Instead of tailoring components to precise sensors and actuators, resource interfaces can express explicitly a range of matching operational constraints under which components can be programmed and composed. At composition and deployment times, such constraints can be viewed as requirements to be checked between components, or over the available hardware whose capabilities are also expressed as resource interfaces but offered by the containers.

The rest of the paper is organized as follows. The next section covers the background concepts while Section 3 briefly presents our component model and then concentrates on the resource model and resource interfaces. Section 4 develops the composition process towards the creation of assemblies and their deployment. A prototype implementation is described in Section 5. Discussions of related works, conclusions and perspectives then end the paper.

2 Component-Based Approaches in Robotics

2.1 Hardware Heterogeneity

Over the past years, a multitude of robotic platforms have developed, from humanoids to specialized medical robots. Conceiving general and reusable software for such a variety of hardware presents serious challenges. Nesnas [4] analyses the issues raised by hardware heterogeneity, and advocates (as others, *e.g.*, [5]) the development of generic software, abstracting oneself as much as possible from the platform. However many components, generic as they may strive to be, still make implicit assumptions about the resources they will be deployed on. The lack of an explicit description of these assumptions may lead to unexpected results when deploying on hardware that does not conform to them. Thus, the challenge is to both seek genericity and be conscious of the target platforms at the right level of abstraction. Yet current approaches to genericity (abstract classes, patterns, ... [5]) are too operational, hiding important properties, and impose unaffordable levels of indirection, given the performances expected from the real-time parts of robot software. Indeed, generic algorithms often need to be configured for a particular platform to be efficient.

2.2 Requirements for a Resource Model

To provide for flexibility and adaptability, a loose coupling between components and hardware is sought while making the tailoring of generic components to particular platforms automatic, transparent, straightforward and allowing to generate efficient code that makes the most of the hardware provided (distribute the code, optimize the use of the resources, ...). To support this, a good model must express platform-specific information, at different levels of abstraction, enabling the user to specify the requirements on the platform in as much detail as needed. A good resource model shall therefore exhibit the following qualities:

declarative: declarative resource constraints simplify the expression of the information and provide for flexibility and adaptability;

contract-driven: distinguishing between the resources required by the software, and the ones offered by the underlying platform, allows us to explicitly check for the compatibility of their mutual commitments;

flexible: while striving for declarativity, programmers must still be able to express directly their expertise through specific deployment directives;

query-oriented: as much as possible, required resource definitions should be interpreted as queries used to match hardware resources rather than their precise characteristics, to achieve more abstract code;

wide-spectrum: when defining provided resources, the aim is to describe as accurately as possible all of the platform properties, grouped in domains including but not restricted to

temporal: all the time related properties, such as *minimal actuator update frequency, warm-up time for a sensor, ...*

QoS: properties relevant to the quality of service, such as *maximum speed, possible camera resolutions, ...*

physical: physical and mechanical properties of the platform, such as *physical envelope, possible movement trajectories, ...*

crosstalk-aware: real resources are not independent, so their interactions must also be expressed in the model as, for example, interferences between wifi and bluetooth antennae situated in close proximity on the platform;

open-ended: the large spectrum of existing resources requires an extensible model to easily cope with new types of resources, new properties used to describe them, and new kinds of constraints that need to be expressed.

2.3 Rich Interfaces

Acknowledging the central role of interfaces, de Alfaro and Henzinger [3] have proposed the concept of *rich interfaces* to capture whole aspects of component interactions at their boundaries. Rich interfaces, at large, expose all the necessary and sufficient information to verify if components can be composed (compatibility) and then compose them into assemblies with predictable properties. The precise definition of component compatibility depends upon the component model and the properties dealt with in its rich interfaces. Beyond traditional syntactic compatibility, behavior [3], timing [6] and resource consumption [7] have been addressed.

Robotic control architectures, exhibiting so many different constraints (*synchronization, real-time, resource, ...*), require a very rich model of interfaces. The challenge in this paper is to push the rich interface concepts towards representing resource constraints, with the goal of handling hardware heterogeneity by abstracting their essential properties.

3 The Component Model

The work on resource representation presented in this paper extends an already defined component model [12]. This model is based on the globally asynchronous

and locally synchronous (GALS) paradigm, where systems are viewed as a loosely-coupled asynchronous composition of more tightly coupled synchronous entities. As synchronous entities, real-time tasks can be locally composed to get synchronous composites, which can then be asynchronously composed together as well as with the computation-intensive tasks also viewed as asynchronous.

In our model, robotic architectures are built as compositions of sensory-motor behaviors, themselves made of *synchronous (periodic, reactive)* and *asynchronous (active)* components, and nested within *containers* that serve as abstractions for the underlying platform. The execution of synchronous components is divided in periods, with the sensors read at the beginning of every period and the actuators updated at the end. Reactive components are constructed using codels (“code elements”), pieces of code with the following properties:

- each codel has a name (unique in its component) and an activation protocol;
- their execution times are bounded (worst-case execution time, WCET);
- communication and synchronization happens explicitly through named ports;
- each codel specifies the resources it requires and acquires them for the whole duration of their execution.

Codels are the atomic units of resource allocation and thus represent the lowest level of granularity for code scheduling and generation.

As for asynchronous active components, their internal operating model is left open. Although not subject to real-time constraints, and thus having no user defined temporal interfaces, they can still be under time constraints induced by communication with reactive components and the use of passive resources. Both types of components specify their connection, communication and resource interfaces.

Components are deployed in containers, autonomous processing units that combine both physical (*e.g.*, sensors, actuators) and logical resources (*e.g.*, presence of an interruption mechanism or a scheduler). Like in RTOS, every operation inside the reactive container (computation, data exchange, ...) has a bounded execution time. The set of container interfaces consists of resource, connection and communication interfaces. Whereas the resource interface contains all the information necessary to describe the available resources, the connection interfaces explicit how to exchange data with the physical sensors and the actuators. All together, these make up a container into which components can be placed to be deployed. In the rest of the section, the component model is further developed with regards to resource description.

3.1 Towards a Component-Oriented Resource Model

Resource interfaces serve as abstractions for the software-hardware interactions. As such, they propose two complementary views: a platform, container, view describing actual resources and a component view expressing properties of the resources sought by components in the underlying platform.

Container side. A platform provided resource interface can be seen as a type describing a set of hardware and/or software present in the platform. The aim

```

definition of resource types :
resource:{type:camera, props:[res([320x240, 640x480]) , envelope((0, [10-15], 0), 80)]}
resource:{id:bt, type:battery, props:[capacity(1200)]}
resource:{id:wheels, type:wheel_actuator, props:[speed([0-100]), warm_up(10)]}

ressource interface :
resources_pr {a, resources:[r(id:cam, type:camera),
                           r(id:bt, type:battery), r(id:wheels, type:wheel_actuator)],
               constraints:[ccsts{resources:{wheels, bt},
                           props:{wheel.speed, container:btc},
                           cnst:{partial_eq([(0-49], 4), ([50-100], 8)]}}]}

```

Fig. 1. An example of provided interfaces for a simple platform with a battery, a camera and a wheel actuator

is to describe accurately the properties of each resource and the way they affect each other. To this end, it needs to capture the following main concepts:

Individual resource properties jointly describe the resource, each having a domain and eventually a constraint on the set of possible values it can take. *Active resources*, having their own thread of control to produce or consume data or requests, have different constraints than *passive ones* that simply provide or consume data or requests when called.

Cross-constraints tie two or more properties of the same or of different resources, to define the relationship between their admissible values (Fig. II presents a simple example where cross-constraints express that higher speed implies higher battery consumption). Cross-constraints play a key role in the model, as they allow us to express all the dependencies between the different aspects of the system and in the end to compute an overall configuration that is both coherent and valid.

Statefull resources evolve from state to state as operations are called on them and may have constraints that are valid only in some states. For example, a sensor may require a warm-up time before it can be used. Constraints on the transitions, such as warm-up time and the precedence between operations, impact the codelets that use the resource.

Logical or composite resources combine physical or other logical resources to provide enhanced or synthetic services over them (*e.g.*, feedbacks from wheel motors used to represent a crude positioning sensor, several cameras combined to provide a wider angle logical camera).

Component side. For software components, the resource interface specifies the number, type and characteristics of the resources required by the component to execute. The component specifies minimal requirements, therefore any configuration that provides at least the minimal specifications and eventually more should be compatible. This is not as straightforward as it might seem, because of the cross-influence of the different properties on each other. Increasing the size of the battery and oversatisfying the autonomy constraint, for example, might end up violating the total weight constraint of the system.

```
rqr_interface{id:lf_res, constraints:[goal(autonomy(2)),
                                     cst(max_speed(80)), cst(min_res(320X240))]}
```

Fig. 2. An example of required resource interfaces for a line follower component

The resource interface integrates itself in the set of rich interfaces of the component, including its temporal, connection and communication interfaces. Reactive components specify their required resources both on component and codel levels, indicating the latter for synchronization purposes. Active components do so only at the component level. Two levels of abstraction are possible: *overall goals* and *specific constraints* on certain properties of resources. Overall goals are directives with a high level of abstraction, such as the robot autonomy on battery. These goals need to be refined in terms of lower level constraints to be directly verified (*e.g. energy consumption of each resource*). Specific constraints are aimed at users with a deeper understanding of the underlying platform.

In Figure 2, examples of both types of constraints are given. A specific constraint on the maximum speed ensuring that the robot will never lose the line, constrains the corresponding basic property of the physical resource. The higher level goal on the other hand expresses the minimal autonomy requirement of two hours on the system and corresponds to a set of constraints on the rate of battery use for every physical component of the platform.

3.2 Formal Resource Interface Model

The formalization of the resource interfaces (temporal aspects of the model are presented more in details in [2]) paves the way to the composition aspects discussed in the next section. Components are connected with containers through ports and entry points explicitly declared on both sides. Resource interfaces describe the required and provided resources and constraints on these ports and entry points. More formally, it is a tuple $I_R(n, \Gamma, \Pi, \beta, csts)$, where

- $n \in Ide_I$ is the resource interface identifier;
- Γ is a set of resource descriptions, each being a tuple $\rho(n, \tau, \epsilon, \mathcal{P})$ where
 - $n \in Ide_r$ is the name of the resource,
 - τ is the type of the resource (*e.g. processor*),
 - $\epsilon \subseteq Ide_\epsilon$ is a set of entry point identifiers, and
 - \mathcal{P} is a set of properties that can be constrained, each of them being described as a tuple $p(n, u)$ where n is a property name and u an optional measurement unit for values of this property;
- Π is the set of ports and entry points the interface applies to, as tuples $\pi(n, \epsilon)$ where n is the port identifier and ϵ the set of entry point identifiers attached to this port;
- $\beta : \Pi \rightarrow \Gamma$ binds ports to resource descriptions; and,
- $csts$ is the set of constraints, themselves tuples $cst(t, ps, e)$ where

- $t \in Ide_{cst}$ is the type of the constraint
- ps is a set of identifiers of the constrained properties, prefixed by the port and entry points identifiers if necessary, or the name of an overall goal (only for required interfaces), and
- e is the constraint expression.

Provided interfaces, noted I_{PR} , are attached to containers, while required ones, noted I_{RR} , are attached to components. They differ essentially by the kinds of constraints they impose. Required interfaces strive for wide-spectrum constraints and, whenever possible, overall goals, while provided ones explicit the precise constraints of hardware and logical resources, as discussed before. This formalization leaves several points of extension:

- The set of resource types τ , describing precisely the substitutability of resources, their properties as well as the domains of these, is meant to be extensible to ease the inclusion of new kinds of resource in the framework.
- The constraint expressions e can be as simple as equalities or inequalities (to define intervals of admissible values), as in QML [9]. The use of general purpose constraint solvers to verify the compatibility and to look for configurations (see the next sections) allows us to use various forms of constraints. The constraint language of the model is extensible to allow for new and more comprehensive constraint solvers to be easily incorporated in the framework.

Formal container definitions include, among other things, their provided resource interface. Formal definition of reactive and active components also include their required resource interface. Reactive components, as discussed earlier, exhibit their required ports and entry points, but codels also give the list of port and entry points they use, so that it can be taken into account when scheduling them, for example to impose mutual exclusion or temporal constraints from the provided interface of their deployment container. A formalization of this aspect of our component model is not shown in details due to space restrictions.

4 Composition and Composites

Resource interfaces are axed on properties and the interactions between them. Composition, on the other hand, is axed on the use of constraints by defining the way they are going to be processed. A distinction is made between two forms of composition: horizontal, between entities on the same abstraction level and vertical, between entities on different levels of abstraction [8].

First of all, as resource interfaces will be defined at different times by different people, a semantic alignment between terms and concepts may be required. Two solutions appear: an a priori alignment through standards or a more dynamic alignment through ontologies. In the rest of the paper, this problem will not be addressed any further, but either solution could be integrated in our resource model. The next step is to define how to add constraints and cross-constraints when composing horizontally, and how entailment is calculated for every type of property when composing vertically.

4.1 Horizontal Composition

At the resource level, horizontal composition refers to composition between two resource interfaces, to generate a common resource interface. This implies joining their resources and ports (and entry points), and combining their constraints. More formally, the composition can be seen as a parameterized operator, with the name n' and the cross-constraints $csts'$ of the composition, applied to the two resource interfaces:

$$\begin{aligned} I_R(n_1, \Gamma_1, \Pi_1, \beta_1, csts_1) \oplus_{(n', csts')} I_R(n_2, \Gamma_2, \Pi_2, \beta_2, csts_2) = \\ I_R(n', \Gamma_1 \cup \Gamma_2, \Pi_1 \cup \Pi_2, \beta_1 \oplus \beta_2, csts_1 \sqcap csts_2 \sqcap csts') \end{aligned}$$

where

- $\Gamma_1 \cup \Gamma_2$ is the union of resource descriptions modulo renaming (to avoid name clashes),
- $\Pi_1 \cup \Pi_2$ is the union of ports and entry points also modulo renaming,
- $\beta_1 \oplus \beta_2$ is the combination of the two mappings, also modulo renaming, and
- $csts_1 \sqcap csts_2 \sqcap csts'$ is the set of constraints that is the greatest lower bound of the three sets of constraints in the entailment relation ($cs \models cs'$ if all constraints in cs' are implied by the set of constraints cs), that is the maximal set of constraints that entails $csts_1$, $csts_2$ and $csts'$.

The precise definition of the entailment relation depends on the constraint expression language, and is therefore another point of extensibility in the framework. The constraint solvers used in the current implementation of the framework provide for a wide spectrum of constraint expressions.

4.2 Vertical Composition

Composition of components with containers, to generate a deployable composite is done in two steps: (1) verification of composability, which checks whether the provided interface entails the required one, and (2) computation of the composite itself. Our approach consists in building a “correct by construction” solution, which means that we address the two steps simultaneously. The challenges of component-platform composition are now discussed more in detail.

Abstraction levels mediation. When imposing a higher level goal, it is necessary to refine it to the constraints it enforces on the properties of the platform.

Example. Component side, the user requires at least two hours of autonomy from the platform (the time to get from one dock to another), but platform side the information is represented as a series of graphs describing energy consumption for each physical component. Figure 3 illustrates the two levels of abstraction. To bridge the gap between the two, it is necessary to refine the autonomy goal as a cross-constraint on the energy consumption of all the components.

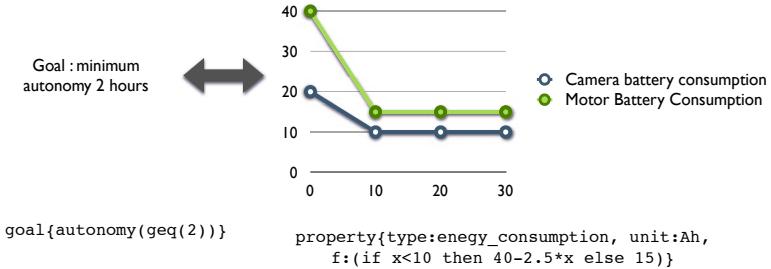


Fig. 3. Different levels of information representation

Simultaneous resolution of time and resource constraints. The aim when composing containers and components is to solve conjointly both constraint systems to find a suitable schedule and resource assignment scheme, by keeping track of all the cross-constraints.

Example. A submarine uses a sonar to exchange signals, but due to the specifics of water as a communication medium, not only must the submarine either emit or receive, but it must also coordinate with other submarines exchanging within a certain radius, so that only one emits or receives at a time. Now, if we only consider the resource constraints, we can enforce these restrictions by adding delays between subsequent emissions and receptions whenever necessary. This however might lead to violating the systems real-time constraints. If on the other hand we solve both the scheduling and the resource constraints simultaneously, then if the temporal constraints permit it, the submarine receptions and emissions can be scheduled in a way that ensures that the necessary delays are respected. A group of submarines is handled similarly by calculating their schedules conjointly and keeping them synchronized. This requires the deduction of constraints on time-slot allocation of resources and temporal constraints from the temporal interface.

Resource allocation. When placing constraints on resources, the user has several degrees of control:

- the easiest way is not to specify distinct abstract resources, but to put constraints on resource types: a codel can for example require the use of a resource of type processor, which means that any available physical processor can be assigned to it, thus abstracting the user from the platform completely;
- when a finer control is needed, ensuring for example that several codels use the same resource, the user indicates the identifier of an abstract resource;
- assuming the user has the necessary knowledge of the platform, it is possible (for optimization purposes, for example) to bind an abstract resource to a particular physical one, with a constraint added at composition time:
assign{virtual = virtual_id, physical = physical_id}

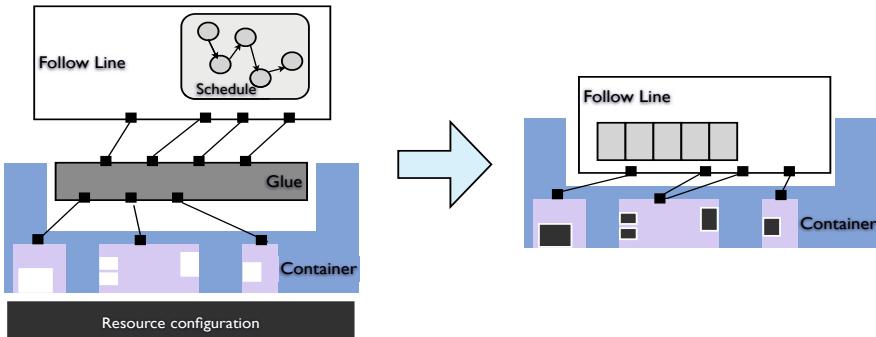


Fig. 4. A schematization of the composition process

Vertical composition. Simultaneous resolution of all the constraints generates a set of deployment directives that are applied to generate executable code. Figure 4 shows the different steps of this transformation:

- the ports and entry points of the required and provided interfaces need to be connected;
- they are then used to deduce the connection between the resource descriptions, with the help of β_p and β_r ;
- the high-level goals need to be decomposed into lower-level constraints, and the coherence of constraint sets pointing to the same resource must be verified;
- the resulting set of constraint is solved;
- the scheduling, resource assignment and resource configuration directives calculated are used together with the connection protocols of the required and provided resources to generate deployable composites.

5 Prototype and Implementation

In the current implementation, a Java-based syntax has been chosen for components and containers. Resource interfaces are extracted from these and all the computations are done by a kernel based on the formal component model. The high-level declarative component representation is then transformed into a formal description of the constraint and once a formal component model is calculated, it is used to generate the deployable component in the chosen language. This makes the kernel less dependent on the external component representation.

The current implementation of the kernel is written in ECLⁱPS^e (an extension of Prolog for constraint programming) using resolution, backtracking and labeling techniques to build constructive proofs of composite validity providing the schedules and resource assignments. Constraint programming, allows for a very intuitive representation of the component constraints.

6 Related Works

Amongst the problems discussed by Nesnas [4] is the well-known issue of hardware dependencies, in particular in the context of software and hardware heterogeneity. Our rich interface approach takes into account most of his recommendations by making constraints and choices more explicit. SAIA [10] proposes very interesting approaches to dealing abstractly with sensors and actuators that inspired us. Schlegel [11] provides one of the most comprehensive treatment of abstracting communication in robotic control architectures. Vaughan and Gerkey [12] discusses issues in accessing devices uniformly through protocol abstractions.

Configuration of resources is also a complex task, currently done manually, as exemplified by ORCA [13] and MARIE [14] component models, but that would better be done automatically, as our model proposes. Brugali [15] proposes stability analysis for robot mobility where the embodiment concept is thoroughly integrated in robotic concerns; our resource interfaces appear as a tool of choice to support such a crucial aspect of control architectures.

Constraints in resource interfaces and QoS contracts share common grounds. QML [9] is a QoS modeling language tackling contract compatibility, but neither the composition nor the building of composite contracts. On the other hand, most works in QoS composition address the issue of “summing” the QoS of components to get the overall QoS of assemblies. Resource interfaces of Chakrabarti et al. [7] address exactly this issue related to our goal-oriented required interface, but for a limited number of resource aspects.

7 Conclusion and Future Works

The main contribution of this paper stems from the integration of a resource model into a component model for autonomous robotics in order to abstract the software control architectures from the precise hardware resources available on the robotic platform. The resource model is viewed as an extensible framework aimed at representing all kinds of physical resources, through a precise description of their many-faceted properties. With composition in mind, our work leverages the concepts of rich interfaces of Alfaro and Henzinger to capture, as declarative constraints, the dependencies of the control software on physical resources in required resource interfaces, while the actual robot platforms exhibit their hardware resources in provided interfaces. The model is rich enough to capture all kinds of properties upon which the control architecture may depend: calling protocols, functional properties (*range of values, precision, ...*), non-functional properties (*time constraints in the use of resources, ...*) as well as physical properties (*consumption, mechanical envelopes, ...*).

In our approach, control architectures are obtained from the horizontal composition of components under the constraints of their required resource interfaces. Robotic platforms are abstracted by containers that also compose horizontally their resource interfaces to obtain a provided composite resource interface. The whole system is then made deployable by composing vertically the components

and the containers, a process which includes the verification of compatibility between the required interfaces of the formers and the provided ones of the lat-ters. The composition process is formalized as a constraint-solving problem, that provides for compatibility checking and selection of precise configuration values imposed by the cross-constraints among resources.

Perspectives of this work include addressing a larger spectrum of resources, but more ambitiously, integrating this resource-aware, composable component model into an end-to-end development methodology that could evolve towards a hardware/software co-design approach specific for autonomous robots.

References

1. Rogovchenko, O., Malenfant, J.: Composants et composition pour les architectures de contrôle de robots. *Revue des systèmes série JESA. Journal Européen des Systèmes Automatisés* 42(4), 423–438 (2008)
2. Rogovchenko, O., Malenfant, J.: Composition and Compositionality for Robotics Components. In: *Proceedings of Software Composition, SC 2010* (June/July 2010)
3. de Alfaro, L., Henzinger, T.A.: Interface-based design. In: *Engineering Theories of Software Intensive Systems. of NATO Science Series*, vol. 195, pp. 83–104. Springer, Heidelberg (2005)
4. Nesnas, I.A.: The CLARAty Project: Coping with Hardware and Software Heterogeneity. In: [5]
5. Brugali, D. (ed.): *Software Engineering for Experimental Robotics*. Springer Tracts in Advanced Robotics, vol. 30. Springer, Heidelberg (April 2007)
6. de Alfaro, L., Henzinger, T.A., Stoelinga, M.: Timed Interfaces. In: Sangiovanni-Vincentelli, A.L., Sifakis, J. (eds.) *EMSOFT 2002. LNCS*, vol. 2491, pp. 108–122. Springer, Heidelberg (2002)
7. Chakrabarti, A., de Alfaro, L., Henzinger, T.A., Stoelinga, M.: Resource Interfaces. In: Alur, R., Lee, I. (eds.) *EMSOFT 2003. LNCS*, vol. 2855, pp. 117–133. Springer, Heidelberg (2003)
8. Crnković, I., Sentilles, S., Vulgarakis, A., Chaudron, M.R.: A Classification Framework for Component Models. To appear in *IEEE Transaction on Software Engineering* (2010)
9. Frølund, S., Koistinen, J.: QML: A Language for Quality of Service Specification. Technical Report HPL-98-10, Software Technology Laboratory, Hewlett-Packard (February 1998)
10. DeAntoni, J.: SAIA: un style architectural pour assurer l’indépendance vis-à-vis d’entrées/sorties soumises à des contraintes temporelles. PhD thesis, INSA de Lyon (2007)
11. Schlegel, C.: Communication patterns as key towards component interoperability. In: [5]
12. Vaughan, R.T., Gerkey, B.: Reusable robot software and the player/stage project. In: [5]
13. Brooks, A., Kaupp, T., Makarenko, A., Williams, S., Orebäck, A.: Orca: A component model and repository. In: [5]
14. Coté, C., Letourneau, D., Raïevsky, C., Brosseau, Y., Michaud, F.: Using marie for mobile robot component development and integration. In: [5]
15. Brugali, D.: Stable analysis patterns for robot mobility. In: [5]

Design Abstraction and Processes in Robotics: From Code-Driven to Model-Driven Engineering

Christian Schlegel¹, Andreas Steck¹, Davide Brugali², and Alois Knoll³

¹ Department of Computer Science, University of Applied Sciences Ulm
89075 Ulm, Germany

{schlegel,steck}@hs-ulm.de

² Department of Computer Science and Mathematics, University of Bergamo
Dalmine 24044, Italy

brugali@unibg.it

³ Department of Informatics, Technische Universität München
85748 Garching bei München, Germany

knoll@in.tum.de

Abstract. Advanced software engineering is the key factor in the design of future complex cognitive robots. It will decide about their robustness, (run-time) adaptivity, cost-effectiveness and usability.

We present a novel overall vision of a model-driven engineering approach for robotics that fuses strategies for *robustness by design* and *robustness by adaptation*. It enables rigid definitions of quality-of-service, re-configurability and physics-based simulation as well as for seamless system level integration of disparate technologies and resource awareness.

We report on steps towards implementing this idea driven by a first robotics meta-model with first explications of *non-functional properties*. A model-driven toolchain provides the model transformation and code generation steps. It also provides design time analysis of resource parameters (e.g. schedulability analysis of realtime tasks) as step towards *resource awareness* in the development of integrated robotic systems.

1 Introduction

The difference of robotics compared to other domains is, e.g., neither the huge number of different sensors and actuators nor the diversity of hardware- and software platforms. It is the *deviation between design-time and run-time optimality* due to the overwhelming size and tremendous complexity of the *problem space* and the *solution space*.

The problem space is huge: as uncertainty of the environment and the number and type of resources available to the robot increase, the definition of the best matching between current situation and correct robot resource exploitation becomes overwhelming, even for the most skilled robot engineer. *The solution space is huge:* in order to enhance robustness of complex robotic systems, existing cognitive methods and techniques need to adequately exploit robotic-specific resources. This means that the robotic engineer should master highly heterogeneous technologies in order to integrate them in a consistent and effective way.

The overall objectives of the proposed novel robot development process are: (i) to promote the synergy between robotics, cognitive system engineering, and software engineering in order to face with new instruments the challenges to build next generation cognitive robotic systems, (ii) to reduce time and cost of the entire development process from requirements specifications of new robotic systems to the reuse of existing software and hardware resources, (iii) to enhance robotic system ability to exploit at best internal and external resources according to the dynamic evolution of the operational context, (iv) to facilitate experimentation of advanced cognitive technologies with increasingly complex robotic platforms and in real-world application scenarios.

In order to achieve this goal, we propose a new design process that aims to reduce both the robotic system engineering problem space and solution space.

2 Design Abstraction for Robotic System Engineering

The *solution space* can be managed by giving the robot engineer the ability to formally model and relate the different views relevant to robotic system design (including their intrinsic variability) and to reason about their correctness by means of offline simulation techniques (i.e., *robustness by design*).

The *problem space* can be mastered by giving the robot the ability to reconfigure its internal structure and to adapt the way its resources are exploited according to its understanding of the current situation (i.e., *robustness by adaptation*). This can be possible thanks to the adoption of online model simulation and (re)configuration techniques.

We think that both *robustness by design* and *robustness by adaptation* can be achieved by raising the level of abstraction at which both, the system engineer and the robot reason about the problem and the solution spaces. This means to rely, as for every engineering endeavour, on the power of models.

Robustness by design refers to the definition of a novel software and system development process that leverages the power of software models to seamlessly and consistently transform application-specific requirements to platform-specific control systems and to validate each individual design decision by means of simulation tools. *Robustness by adaptation* refers to the ability of the control system to reason about and manipulate the exact same software models at run-time in order to autonomously adapt them according to changing operational conditions and to casually reflect the adaptations of the models in the run-time system.

This innovative robot development process illustrated in fig. 1 moves the focus from trying to find all-time optimal solutions at design-time to making the best possible decisions at run-time and thus providing the foundations to achieve both robustness by design and robustness by adaptation.

Engineering of robust robotic systems is tackled by providing means to cope with the unavoidable deviation between design-time and run-time optimal solutions based on exploiting the power of software models. Designers will use offline simulation and analysis tools to validate the robot control system configurations, while the robot control system will use online simulation to manage adaptation to changes in its internals or in its operational context.

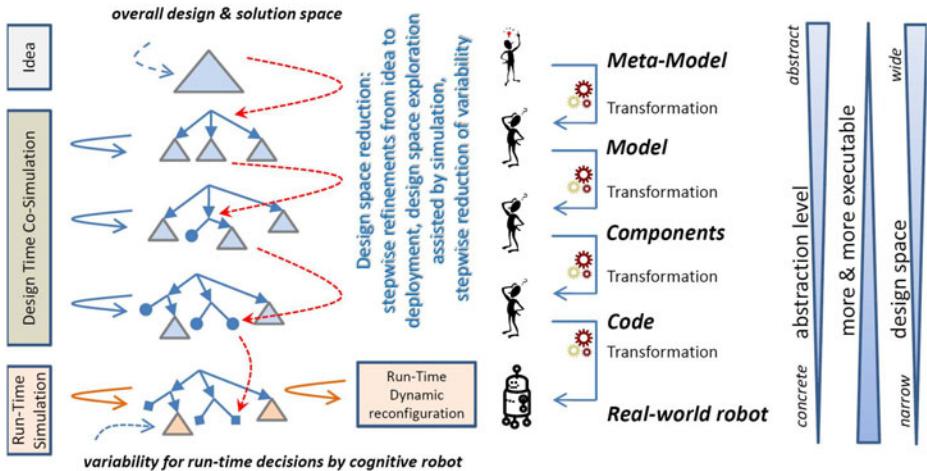


Fig. 1. A novel workflow seamlessly bridging design-time and run-time model-usage for exploiting purposefully left open variation points for run-time decisions

Simulation to support design decisions (model-driven design) or to generate embedded software (model-driven implementation) is considered as a way to master the complexity in system design. Simulation can be done at the intermediate steps of the model transformation and the development process for better predictability and explication. *Offline simulation* supports robustness by design. In this approach, simulation is also a key approach to check and validate alternative robot control system configurations, operational conditions and parameter settings at run-time. Critical situations can be checked online, but before the robot would have to deal with them in real world. This allows to select the most appropriate next action based on simulation of various options and not based on complex reasoning. Run-time monitoring of the selected configuration by simulation-based predictions of expectations can support safety issues and fault detection. Hence, *online simulation* supports robustness by adaptation.

Dynamic reconfiguration: The proposed approach reflects the current trend in the development of self-adaptive systems [1], which is to support increasing automation of decision making with respect to adaptation of systems by exploiting software models not only at design time but also at run-time [2]. These models provide a high-level basis for reasoning efficiently about variability aspects of a system, offer enough details to reflect the run-time variation of its internal and external behavior, and enable its design specifications to evolve while the system is running. Engineers can develop dynamically adaptive systems by defining several variation points (resources, algorithms, control strategies, coordination policies, cognitive mechanisms and heuristics, etc.). Depending on the context, the system dynamically chooses suitable variants to realize those variation points. These variants provide better *Quality of Service (QoS)*, offer new services that did not make sense in the previous context, or discard some services

that are no longer useful [3]. The approach allows to integrate in a model-driven environment mechanisms to monitor and reason about control system configurations at run-time, to validate alternative control system configurations with online simulation tools, and to automatically reconfigure the control system.

Non-functional properties are considered being mandatory for embodied and deployed robots in real-world. Their relevance arises from at least two observations of real-world operation: (i) some components have to guarantee *QoS* (e.g. adequate response times to ensure collision avoidance), (ii) the huge number of different behaviors needed to handle real world contingencies (behaviors cannot all run at the same time and one depends on situation and context aware configuration management and resource assignment). For example, if the current processor load does not allow to run the navigation component at the highest quality level, the component should switch to a lower quality level, resulting in a reduced navigation velocity, but still ensuring safe navigation. Instead of allocating all resources statically, the system needs to be able to adapt itself dynamically thereby taking into account appropriate *QoS* parameters and resource information.

So far, fundamental properties of robotic systems required for system level integration are typically not being made detailed enough nor explicit and they are not yet addressed in a systematic way in most robotics software development processes. Thus, these properties cannot be taken into account neither during design, development and system deployment (e.g. composable out of *COTS* components to foster reuse, ensure maturity and robustness) nor at run-time to support dynamically changing run-time configurations.

3 State-of-the-Art and Related Work

Robotics is a science of integration rather than a fundamental science, and the integration becomes ever more complex [4]. It is far too easy to say *details can be worked out* and it is often comfortable to focus on one aspect in isolation. Robots are complex systems that depend on systematic engineering. Currently, there are many tools that enable to separately design, test and deploy every single aspect of a robot (mechanical parts [5][6], control and algorithmic [7][8][9][10][11], software architecture [12][13][14], etc.). Typically, the different aspects of a robot are separately developed (following a separate process supported by a separate tool), and manually integrated afterwards. This leads to solutions very difficult (when not impossible) to evolve, maintain, and reuse. Systematic engineering processes are not applied at system level.

Software development in robotics is too complex and too expensive because (i) there is too little reuse, (ii) technology changes faster than developers can learn, (iii) knowledge and practices are hardly captured explicitly and made available for reuse, (iv) domain experts cannot understand all the technology stuff involved in software development.

In recent years, the Model-Driven Engineering (MDE) paradigm [15][16][17][18], has proven to mitigate many of the aforementioned limitations, in domains where systems are also highly complex, need a multi-disciplinary development

approach, and require high reliability and robustness, such as: embedded systems [19][20][21][22][23], automotive [24] and home automation [25][26]. *Artist2* [22] addresses highly relevant topics concerning realtime components and realtime execution platforms. *AUTOSAR* [24] comprises software components and their model-driven design. The ongoing *RT-Describe* project [27] addresses resource aspects and run-time adaptation. The *OMG MARTE* [28] activity provides a standard for modeling and analysis of realtime and embedded systems. They provide a huge number of *non-functional properties* to describe both, the internals and externals of a system. All these approaches can be reused for robotics but need to be extended towards model usage at run-time to address the deviation between design time and run-time optimality inherent in robotics. The robotics domain is far more challenging than many *DRE* systems in e.g. automotive or avionics due to the context and situation dependent dynamic reconfigurations of interactions and prioritized resource assignments at run-time.

The most advanced approach of Models@Run.Time [29] is the ongoing *DiVA* project [30] showing promising results in adapting business system architectures. However, it is not obvious how these achievements can be tailored to robotics where not only the system architecture but also the component level needs to be accessible for run-time adaptivity.

Tremendous code-bases (robotic frameworks, tools, libraries, middleware systems etc.) [31][32][33][34][35][36] coexist without any chance of interoperability and each tool has attributes that favors its use. Interestingly enough, new middleware systems are still introduced in robotics (like *ROS* [33]) although advanced standards like *DDS* [36] already provide publish-subscribe mechanisms even with *QoS* support. The robotics community still does not make the step towards *MDSD* that is define meta-models as a meaningful representation for robotic systems to overcome the tight coupling between robotic concepts and implementation technologies.

An introduction into robotics component-based software engineering (*CBSE*) can be found in [37][38]. The BRICS project [39] specifically aims at exploiting *MDE* as enabling approach to reducing the development effort of engineering robotic systems by making best practice robotic solutions more easily reusable. The *OMG Robotics Domain Task Force* [40] develops a standard to integrate robotic systems out of modular components. The *3-View Component Meta-Model V³CMM* [41] comprises three complementary views (*structural view*, *coordination view* and *algorithmic view*) to model component-based robotics systems. *GenoM3* [42] proposes scripting mechanisms to bind component skeleton templates. These ongoing activities are covering first steps towards MDE in robotics. However they do not yet make the step towards seamless migration from design time to run-time model usage as proposed here.

4 The SmartSoft MDSD Toolchain

The SMARTSOFT project [43][44][45] has developed an open source toolchain based on the *Eclipse Modeling Project* [46] that supports model-driven development and integration of robotic software components on a variety of middleware

infrastructures. Its current state fits well into the proposed overall development process although it does not yet exhaustively model a robotics system with all views. However, there is already a huge benefit in even partially managing and explicating *non-functional properties*, e.g. for design-time realtime schedulability analysis, and in supporting the dynamic wiring pattern for run-time dynamic reconfiguration.

The instantiation of the novel work flow by SMARTSOFT concepts is illustrated in fig. 2. The development process starts with an idea. The model is enriched during development time until it finally gets executable in form of deployed software components. *First*, the system is described in a model-based representation (*platform independent model - PIM*) which is independent of the underlying framework, middleware structures, operating systems and programming languages. In this design phase, several system properties are either unknown or only known as requirements.

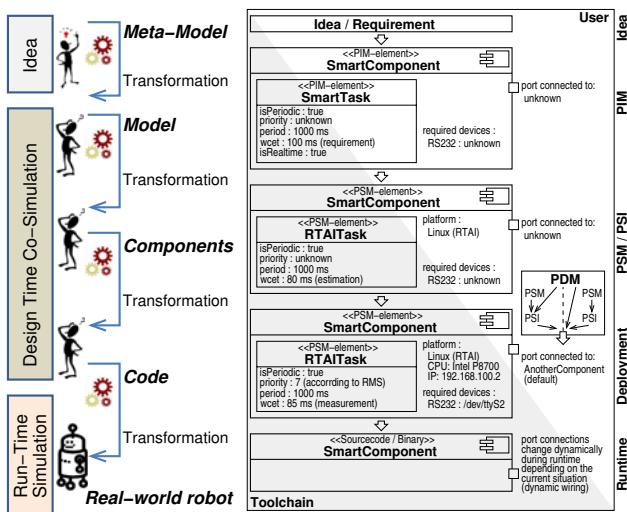


Fig. 2. The development process at-a-glance

Second, after successfully checking the *PIM*, it is transformed into a *platform specific model (PSM)* that provides bindings for platform specific refinements. Some characteristics can still be unknown and are added not until the deployment of the component. Finally, the *PSM* is transformed into the platform specific implementation (*PSI*).

Third, components are deployed to the target computers and robots. Their capabilities and characteristics are de-

fined by the *platform description model (PDM)* which provides further bindings for so far left open model elements. Further model checkings are performed to verify constraints attached to the components against the capabilities of the system (e.g. is executable only on a certain type of hardware, needs one serial port, fulfills realtime guarantees etc.). *Fourth*, the system is run according to the deployment model. Certain properties can still be unknown and will be reasoned during run-time resulting in finally complete bindings of variation points. For example, the set of activated components, the configuration of the components and the wiring between the components can change during run-time depending on situation and context.

4.1 Development of Components Using SmartMARS

SMARTMARS can be seen as both, an abstract meta-model as well as the current implementation as a UML profile. The basic concept behind the meta-model [44] are loosely coupled components offering/requiring services. Services are based on strictly enforced interaction patterns [45] providing precisely defined semantics. They decouple the sphere of influence and thus partition the overall complexity of the system since component internal characteristics can never span across components.

The major steps to develop a SMARTSOFT component are depicted in fig. 3. The component developer focuses on modeling the component hull, which comprises for example service ports and tasks – without any implementation details in mind. The generated component hull provides both, a stable interface towards the user-code (inner view of component) and a stable interface towards the other components (outer view of component). Due to the stable interface towards the user-code, algorithms and libraries can be integrated independent of any middleware structures. The integration is assisted and guided by the toolchain. Fig. 4 illustrates how the glue logic looks like to link existing libraries (or code

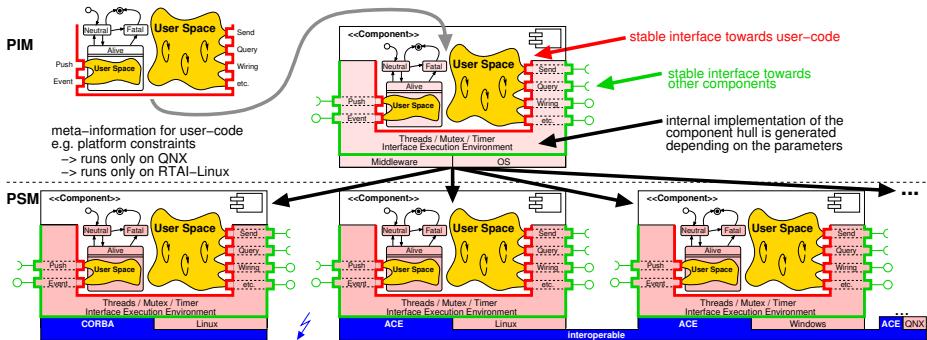


Fig. 3. Refinement steps of component development

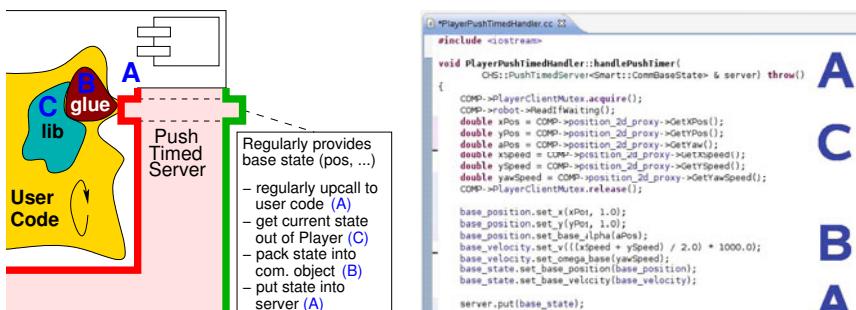


Fig. 4. Gluing user-code to service ports: example of a service port (based on a push-timed pattern) to regularly provide the base state (pose, velocity)

generated by other tools) to the generated component hull. The generation gap pattern [47] protects the user-code from modifications by the code generator. Tags on the component are used to indicate user code constraints (e.g. runs only on RTAI-Linux) and need to be set in the model by the user. The stable interface towards the other components ensures replacement and rearrangement of components.

4.2 Mapping of Abstract Concepts to Specific Technologies

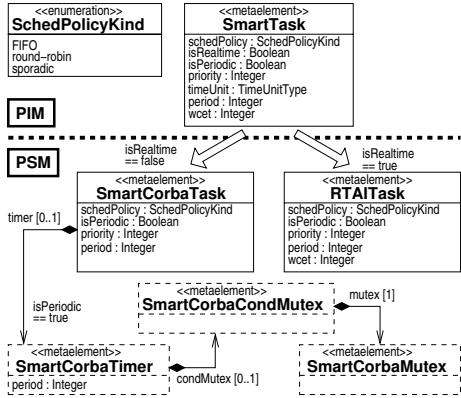


Fig. 5. PIM to PSM transformation

Fig. 5 details the transformation of the *PIM* and the *PSM* by the example of the *SmartTask* meta-element (illustrated using the *CORBA* based *PSM*, but the concepts are generic to any *PSM*). The *SmartTask* (*PIM*) comprises several parameters which are necessary to describe the task behavior and its characteristics independent of the implementation. In case the attribute *isRealtime* is set to *true*, the *SmartTask* is mapped onto a *RTAITask*. For the *RTAITask* the *wcet* and the *period* needs to be set according to the platform as described by the *PDM*. These parameters can be passed to appropriate tools like

CHEDDAR [48] to perform schedulability analysis. In case the target platform does not offer hard realtime capabilities, the toolchain reports this missing property. In case the attribute *isRealtime* is set to *false*, the *SmartTask* is mapped onto a non-realtime *SmartCorbaTask*. In case, the attribute *isPeriodic* is set to *true*, the toolchain extends the *PSM* by elements needed to emulate periodic tasks. The interface (inner view) of the *SmartTask* towards the user-code is stable independent of the internal mechanisms selected by the MDE toolchain.

4.3 Realtime Guarantees / Non-Functional Properties / Simulation

Selected views of the overall software model can be transformed into specific representations for evaluation by external tools (e.g. *CHEDDAR*). For example, schedulability analysis is required for hard real-time tasks and one has to export their parameters and their processor assignment. The M2M transformation rules for converting the appropriate parts of the software model into the desired target format are specified in the toolchain. At design time, the schedulability analysis including the model transformation can be triggered by the robot engineer (offline simulation) as soon as the deployment information and target platform characteristics are added to the model (fig. 6). At run-time, the very same analysis can be triggered by the task sequencing component (online simulation). The

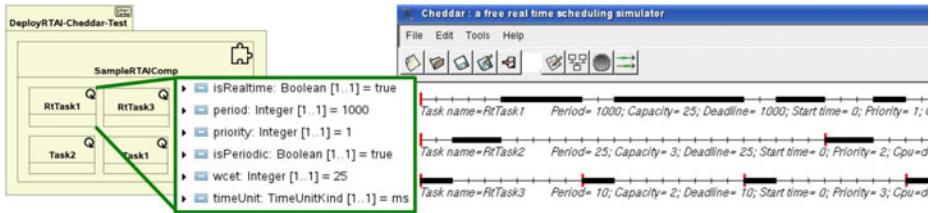


Fig. 6. CHEDDAR simulation based on the model parameters of the realtime tasks

task sequencer [49] comprises task nets to describe action plots. Task nets are expanded during run-time to executable behaviors. Prior to enabling a configuration, the selection of a certain configuration (e.g. set of activated components) can be based on the result of the schedulability analysis at run-time. How to invoke this analysis and its results (feasible or not) can be represented as behavior usable inside task nets. This already illustrates the advantages of explicating parameters in a modeling level for design-time and run-time simulation. It is one example of *QoS* and *resource awareness* in robotic systems.

4.4 Dynamic Wiring and Online Reconfiguration

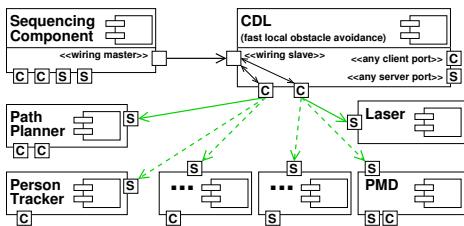


Fig. 7. Online dataflow adaptation

laser or the *PMD* component. *Dynamic wiring* is among others needed by the task sequencer for execution of task nets. The wiring pattern is used to compose the various components to different behaviors.

Dynamic wiring is also used to temporarily replace the robot component with the robot simulator component at run-time (Gazebo [32] with physics engine). This online simulation is used to determine at run-time the maximum allowed velocities taking into account the current payload of the robot. Prior to real-world execution, the variation point of the maximum allowed and save velocity is bound to a situation-dependent value. Again, this configuration is also represented as behavior (see above, same as for run-time schedulability analysis).

Fig. 7 shows the application of the wiring pattern [50] which supports run-time configuration of the communication paths between components. A *wiring master port* can be used to rearrange the client connections of components if these exhibit a *wiring slave port*. For example, the *CDL* component can receive its intermediate goals either from a *path planner* or a *person tracker* and its distance information from either the

5 Real-World Application and Evaluation

The model-driven toolchain has been used to build numerous robotics components (navigation, manipulation, speech, person detection and recognition, simulators) reusing many already existing libraries within the component hulls (*OpenRAVE*, *Player/Stage*, *GMapping*, *MRPT*, *Loquendo*, *VeriLook*, *OpenCV*, etc.). These components are reused in different arrangements (fig. 8) to implement, for example, *Robocup@Home* scenarios (*Follow Me*, *Shopping Mall*, *Who-is-Who*).

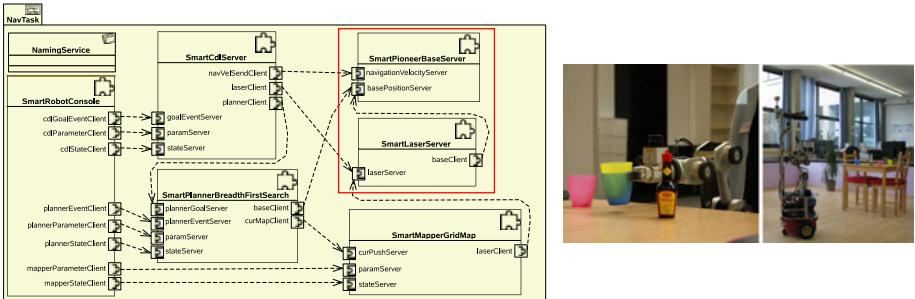


Fig. 8. Deployment diagram of a navigation task. Switching to simulation is done by replacing services (e.g. Stage/Gazebo component provides services of P3DX and LRF).

The model-driven approach, even in its current state, already proved to significantly reduce efforts of maintaining the various scenarios and composing new ones out of existing building blocks.

6 Conclusion and Future Work

We propose a novel design process for robotic system engineering which provides the ground for processes for the overall lifecycle of complex robotic systems (development, simulation, testing, deployment, maintenance). It allows to collaborate at the meta-models, transformations etc. and to compete at their implementation. It provides the freedom to choose whatever implementational technology is most adequate in a target domain. Furthermore, it provides the ground for a component market based on models, interoperability and services (e.g. value-adding expertise can be made available by custom-models). Most important, it provides a perspective of how to tackle *robustness by design* and *robustness by adaptation* by seamlessly bridging design-time and run-time model-usage for exploiting purposefully left open variation points for run-time decisions.

Future work will focus on further exploiting run-time dynamic reconfiguration and more extensive integration of run-time simulation coordinated by task-nets for which the current state of the model-driven design process, toolchain and component model (dynamic wiring, resource annotations) already proved to be a suitable starting point.

Acknowledgment

Work presented in sections 4 and 5 has been conducted within the *ZAFH Servicerobotik* (<http://www.zafh-servicerobotik.de/>), which receives funding from state of Baden-Württemberg and the European Union. Jan Broenink, Cristina Vicente-Chicote and Paul Valckenaers substantially contributed to the ideas detailed in sections 1 and 2.

References

1. Bencomo, N., Grace, P., Flores, C., Hughes, D., Blair, G.: Genie: Supporting the model driven development of reflective, component-based adaptive systems. In: ICSE, pp. 811–814 (2008)
2. Blair, G., Bencomo, N., France, R.B.: Models@run.time. IEEE Computer (2009)
3. Elkhodary, A., Malek, S., Esfahani, N.: On the Role of Features in Analyzing the Architecture of Self-Adaptive Software Systems. In: 4th Workshop on Models@run.time, MODELS 2009 (2009)
4. Bruyninckx, H.: Robotics Software: The Future Should Be Open. IEEE Robotics & Automation Magazine 15(1), 9–11 (2008)
5. ANSYS Simulation Driven Product Development, <http://www.ansys.com/>
6. Adams for Multibody Dynamics, <http://www.mscsoftware.com/Contents/Products/CAE-Tools/Adams.aspx>
7. Matlab/Simulink, <http://www.mathworks.com/>
8. 20-SIM, <http://www.20sim.com/>
9. SCILAB, <http://www.scilab.org/>
10. MODELICA, <http://www.modelica.org/>
11. Labview Robotics, <http://www.ni.com/Robotics>
12. SHE / POOSL, <http://www.es.ele.tue.nl/she>
13. The Vienna Development Method Portal, <http://www.vdmportal.org/>
14. Overture: Open-source Tools for Formal Modelling, <http://www.overturetool.org/>
15. OMG Model-Driven Architecture, <http://www.omg.org/mda/>
16. Völter, M., et al.: Model-Driven Software Development: Technology, Engineering, Management. Wiley, Chichester (2006)
17. Bézivin, J.: On the unification power of models. Journal of Systems and Software 4(2), 171–188 (2005)
18. Innovations in Systems and Software Engineering Special Issue Model based development methodologies, vol. 5(1). Springer, Heidelberg (2009)
19. MEDEIA: Model-driven embedded system design environment for the industrial automation sector, 7FP EU Project, <http://www.mededia.eu/>
20. SATURN: SysML bAsed modeling, architecTUre exploRation, simulation and synthesis for complex embedded systems, 7FP EU Project, <http://www.saturn-fp7.eu/>
21. QUASIMODO: Quantitative System Properties in Model-Driven Design of Embedded Systems, 7FP EU Project, <http://www.quasimodo.aau.dk/>
22. ArtistDesign, European Network of Excellence on Embedded System Design, 7FP EU Project, <http://www.artist-embedded.org/>
23. Truscan, D., Lundkvist, T., Alanen, M., Sandström, K., Porres, I., Lilius, J.: MDE for SoC design. Innovations Syst. Softw. Eng. 5, 49–64 (2009)

24. AUTOSAR: Automotive Open System ARchitecture, <http://www.autosar.org/>
25. POBICOS: Platform for Opportunistic Behavior in Incompletely Specified, Heterogeneous Object Communities, 7FP EU Project, <http://www.ict-pobicos.eu/>
26. AMPLE: Aspect Oriented, Model Driven and Product Line Engineering, 6FP EU Project, <http://www.ample-project.net/>
27. RT-Describe, <http://www.esk.fraunhofer.de/EN/press/pm0911RTDescribe.jsp>
28. OMG MARTE, A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Beta 2, ptc/2008-06-08, Object Management Group (June 2008)
29. Blair, G., Bencomo, N., France, R.B.: Models@ run.time. Computer 42(10), 22–27 (2009)
30. Morin, B., Barais, O., Nain, G., Jezequel, J.-M.: Taming Dynamically Adaptive Systems using models and aspects. In: IEEE 31st International Conference on Software Engineering (ICSE), pp. 122–132 (2009)
31. Robot Standards and Reference Architectures (RoSTA),
<http://wiki.robot-standards.org/index.php/Middleware>
32. Player/Stage/Gazebo, <http://playerstage.sourceforge.net/>
33. ROS: Robot Operating System, <http://www.ros.org/>
34. Microsoft: Microsoft Robotics Developer Studio,
<http://msdn.microsoft.com/en-us/robotics/default.aspx>
35. OMG CORBA, <http://www.corba.org>
36. OMG DDS: Data Distribution Service for Real-time Systems (DDS) v1.2,
http://www.omg.org/technology/documents/dds_spec_catalog.htm
37. Brugali, D., Scandurra, P.: Component-based Robotic Engineering. Part I: Reusable building blocks. IEEE Robotics and Automation Magazine (2009)
38. Brugali, D., Shakhimardanov, A.: Component-based Robotic Engineering. Part II: Models and systems. IEEE Robotics and Automation Magazine (March 2010)
39. BRICS: Best Practice in Robotics, <http://www.best-of-robotics.org/>
40. OMG Robotics: OMG Robotics Domain Task Force, <http://robotics.omg.org/>
41. Alonso, D., Vicente-Chicote, C., Ortiz, F., Pastor, J., Alvarez, B.: V³CMM: a 3-View Component Meta-Model for Model-Driven Robotic Software Development. Journal of Software Engineering for Robotics 1(1), 3–17 (2010)
42. Mallet, A., Pasteur, C., Herrb, M., Lemaignan, S., Ingrand, F.: GenoM3: Building middleware-independent robotic components. In: Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA), Anchorage (2010)
43. SmartSoft, <http://smart-robotics.sf.net/>
44. Schlegel, C., Haßler, T., Lotz, A., Steck, A.: Robotic Software Systems: From Code-Driven to Model-Driven Designs. In: Proc. 14th Int. Conf. on Advanced Robotics (ICAR), Munich, Germany (2009)
45. Schlegel, C.: Communication Patterns as Key Towards Component Interoperability. In: Software Engineering for Experimental Robotics. STAR Series, vol. 30, pp. 183–210. Springer, Heidelberg (2007)
46. Eclipse Modeling Project, <http://www.eclipse.org/modeling/>
47. Vlissides, J.: Pattern Hatching - Generation Gap Pattern (1996),
<http://researchweb.watson.ibm.com/designpatterns/pubs/gg.html>
48. The Cheddar project, <http://beru.univ-brest.fr/~singhoff/cheddar/>
49. Schlegel, C., Illmann, J., Jaberg, H., Schuster, M., Wörz, R.: Integrating Vision-Based Behaviors with an Autonomous Robot. VIDERE - Journal of Computer Vision Research, 32–60 (2000) ISSN 1089-2788
50. Schlegel, C.: Communication Patterns as Key Towards Component-Based Robotics. Int. Journal of Advanced Robotic Systems 3(1), 49–54 (2006)

Using Simulation to Assess the Effectiveness of Pallet Stacking Methods

Stephen Balakirsky¹, Fred Proctor¹, Tom Kramer¹, Pushkar Kolhe²,
and Henrik I. Christensen²

¹ National Institute of Standards and Technology, Gaithersburg, MD 20899-8230

{stephen.balakirsky,frederick.proctor,thomas.kramer}@nist.gov

² RIM@GT, CoC, Georgia Institute of Technology, Atlanta, GA 30332-0280
{pushkar,hic}@cc.gatech.edu

Abstract. Stacking objects onto pallets is the most widely used method of bulk shipping, accounting for over 60% of the volume of goods shipped worldwide. Due in part to the wide variation in size, shape and weight of objects to be palletized, successful stacking is more of an art than a science. Pallet stacking is an example of the three dimensional cutting stock problem, a variant of the combinatorial NP-hard knapsack problem. This makes it hard to find a common heuristic to satisfy requirements for major shippers and receivers. In this paper, we explore the role that simulation can play in comparing how well automated metrics can gauge the quality of a pallet. A physics enabled simulator allows us to study the interlocking among hidden internal boxes and makes it possible to study metrics that cannot be determined by mere statistics. Results from a competition held in May 2010 are used to illustrate the concepts.

Keywords: Palletizing, Simulation, Competition, Performance Metrics, USARSim.

1 Introduction

Consider the problem of distributing packed grocery items to various retailers. One possible solution is to have each class of goods (e.g. milk or cookie brand x) arranged by workers into individual pallets and then shipped on trucks. This operation takes a substantial amount of time and is subject to mistakes. The process becomes more efficient if more items are shipped per pallet. However, for some vendors and retailers a full pallet exceeds their total demand. This is very commonplace in manufacturing industries as the Just-In-Time and Efficient Consumer Response strategies have become popular. To solve this problem, various commercial logistics solutions allow products to be shipped in mixed pallet loads, where multiple classes of products are grouped onto a single pallet. Most of these solutions use heuristic approaches or formulate the problem as a mixed integer linear program to solve the manufacturer's bin packing problem. However the heuristics used in these problems are statistical, and there is no way to

know if a pallet can be created at all. Since the solution is NP-hard it is hard to compare two different commercial solutions.

We employed a newly created pallet quality evaluation simulator known as Pallet Viewer and the Unified System for Automation and Robot Simulation (USARSim) framework to test the palletizing process scenario. The Pallet Viewer software allows us to evaluate various performance metrics of a pallet after each item is placed on the pallet stack. Since USARSim uses physics, we can use it to simulate the process of creating a pallet. This allows us to verify the correctness of the planner and the robot motion planning algorithms. This combination of software tools allows us to compare different pallet packing plans and motion solutions.

The remainder of this paper is organized as follows. Section 2 briefly discusses related work in this area. Section 3 decomposes and describes the entire palletizing process. Section 4 discusses our proposed metrics for evaluating the quality of a mixed pallet, while Section 5 describes the software that implements these metrics. Section 6 describes a recently run competition in which teams built both simulated and real mixed pallets that were judged by the metrics mentioned above. Finally, Section 7 presents conclusions and areas for future work.

2 Related Work

There are many commercially available palletizing solutions that offer software planners as well as complete robotic systems. Cape Pack from Cape Systems and TOPS Pro from Tops Engineering Corp¹ are software packages that offer palletizing solutions. Almost all such solutions use statistical heuristics to determine the best pallet and present the user with these statistics to determine the pallet pattern to use. Only an expert user can make sense of such statistics. On the other hand, our system uses statistics and dynamics of a physics enabled simulation engine to guarantee the solution. Our system also uses an openly published interface which vendors can use to compare various pallet solutions.

The manufacturer's bin packing problem has also been studied in research, and there has been significant work in finding good heuristics. It is well known that the problem is fundamentally NP-hard and as such the emphasis is on use of heuristics and formalization of the problem as discussed in [10]. Most of the recent work has concentrated on generating stable pallets by using pre-defined pallet patterns which are known to be theoretically stable as discussed in [23]. Our simulation software tries to evaluate pallets on heuristics not known by the palletizer software. These heuristics weigh stability and density of the pallet equally, and are therefore suitable for evaluation purposes. In our last venture of describing this software at a conference we also showed that we can confirm

¹ Certain commercial software tools and hardware are identified in this paper in order to explain our research. Such identification does not imply recommendation or endorsement by the authors, nor does it imply that the software tools and hardware identified are necessarily the best available for the purpose.

the simulation results in the real world. We demonstrated the pallet stacking process on a real robot cell which closely resembles our simulation environment at ICRA 2010.

3 Modeling and Analysis

Consider a manufacturing plant such as a beverage company that produces products of various kinds. When the plant receives orders from vendors, the different products are packaged and shipped to the vendors in the most efficient way possible. This process can be subdivided into:

- | | |
|----------------------|----------------------|
| 1. <i>Presorting</i> | 3. <i>Storing</i> |
| 2. <i>Packaging</i> | 4. <i>Retrieving</i> |

The logistics system in a plant makes the plan to process a given order. The planner outputs a package configuration which specifies the number of pallets to be created and the configuration of each product on each of those pallets. The presorting stage consists of a series of conveyors and stops that sort the packages. The packages are sorted such that each package goes to the correct palletizing system. A palletizing system consists of several robots picking up packages from the conveyor and arranging them onto one or many stacks. A given warehouse system can consist of many such palletizer systems. The packaging stage places all the products on their respective pallets. The storing and retrieving stage is usually an independent system. However, some plants also consider the storage and retrieval heuristics in their planner.

3.1 The Palletizing System

Most palletizing systems consist of a single robot being fed by a single conveyor. The arm takes the packages from the conveyor and arranges them on a pallet. If you consider an existing manufacturing plant, any improvement in the sorting stage would mean redesigning the plant structure. The palletizing system can be improved by improving the mixed palletizing algorithm the planner uses. Even a small improvement in the planner software can make the entire process faster and more efficient. This is the reason for our interest in modeling the simplest palletizing environment in USARSim simulation software and the Virtual Manufacturing Automation Competition (VMAC).

The Robot System. The palletizing environment also depends on the type of palletizing robot and gripper being used. In general, there are two types of systems - one in which the robot can pick up individual packages and the other in which the robot picks up packages in a layer. In both of these scenarios, a suction gripper is the most commonly used gripper. If the packages are flexible, however, other grippers are used.

Modeling in USARSim. At its current development stage, the simulator can simulate multiple robot arms stacking multiple pallets while picking packages from a single conveyor. At this stage, we only model a suction gripper on each robot arm that can lift one package at a time. For the presorting stage, we allow the user to sort the order and spawn packages in the order in which the planner is supposed to feed them into a palletizing system. This means that pallets cannot be constructed in parallel, but it achieves the objective of verifying the construction of every pallet in a dynamic simulation environment.

3.2 Planning and Analysis

The entire planning process can be thought of as a three-layered hierarchical planner.

1. The uppermost level processes the order from various vendors and develops a plan for each and every package. This plan is run only once for each order. From this plan, the system knows the final position of each package. This plan takes into account various heuristics such as time required to construct the pallet, the stability of the pallet, etc.
2. The middle level coordinates the higher level planning process and the lower level motion planners. The sorting of packages through the conveyors and coordination between multiple robots sharing the same workspace is done by this planner. In the case of multiple robots, it can schedule each robot's pick from the conveyor.
3. The lowest level is a motion planner for the robotic arms. This planner makes plans for every package. For every package, it finds a collision free path from the conveyor to the final position on the pallet.

4 Metrics

4.1 Introduction

Roughly speaking, a metric for palletizing is a quantitative measure of some aspect of any of the following:

- one box that is part of a stack on a pallet
- the entire collection of boxes in a stack on a pallet
- a set of stacked pallets
- the process of building stack(s) of boxes on pallet(s).

Our work in developing metrics for palletizing has only just begun. The Pallet Viewer utility, which displays pallets stacked with boxes, calculates eleven metrics as follows. Two of these pertain to one box (the first type above). The rest pertain to a collection of boxes in a stack on a pallet (the second type above).

- Number of file errors
- Total weight
- Stack height
- Volume of boxes
- Pallet storage volume
- Volume density
- Package N overlap fraction
- Package N connections below
- Pallet average overlap fraction
- Pallet average connections below
- Current file error count

These are described further in Section 5.1. They are automatically updated and displayed on the screen each time a box is added to the stack or removed from it. The program gets its information from an order form, a plan for a pallet, and a description of the finished pallet. While the Pallet Viewer displays both the pallet as planned and the pallet as built, the metrics are calculated only for the pallet as planned.

We could readily calculate more metrics, but we would like to calculate only those that are important in industry. Towards this end, we have studied the web sites of major shippers and collaborated with a supplier, Kuka Logistics, of an automated palletizing system.

Further study and additional collaborations with commercial users and vendors of palletizing systems will be needed to ensure that our metrics are useful in practice. In addition, we will need to devise a weighting scheme for combining metrics. Different weighting schemes may be required for different operating environments.

4.2 Weight Support Mode

There are at least two different common modes for the way in which boxes are supported by other boxes. What constitutes a good stack is very different between the two.

In one mode, which we might call the *cardboard* mode, when anything (such as another box) is put on top of a box, it is the material from which the box is made that bears the weight. The *cardboard* mode occurs with boxes containing breakable items such as television sets and glassware. These boxes may have additional internal supports such as columns at the corners.

In the other, *contents* mode, it is the contents of the box that provide most of the support for any weight placed on top of the box. The *contents* mode occurs with boxes containing relatively robust items such as cans of soda or reams of paper. In the *contents* mode, the material from which the box is made may have very little resistance to compression. The contents have it, instead.

The goodness or badness of metrics such as how much boxes overlap varies widely between the two modes. In *cardboard* mode, overlap is usually a bad thing because the edges of the boxes (the load-bearing part) are not lined up vertically. One web site [6] says overlap (also called interlocking) “can destroy up to 50% of the compression strength”. In *contents* mode, overlap is generally good, since it tends to hold the stack together. The edges of the boxes are not load bearing in *contents* mode. All shippers expect pallets to be wrapped with plastic and possibly also with straps. The wrapping and strapping hold the stack together and fasten it to the pallet, so the benefits of overlap are usually not large.

The work we have done so far has used pallets of boxes of supermarket items. These are almost all *contents* mode boxes, so overlap has been looked upon favorably.

4.3 Box Robustness

Boxes of different items have different maximum loads. No box on a pallet should have its maximum load exceeded. The XML schema for boxes we have been using, Article.xsd [7], includes an optional Robustness element, which is intended to represent maximum load. Robustness has as sub-elements: MaxPressureOnTop, SourcePalletLayers, and RelativeRobustness. The last two of these are optional. The pallet metrics we have devised thus far do not include using Robustness.

MaxPressureOnTop is given in grams per square millimeter.

SourcePalletLayers, the number of layers on a pallet of identical boxes on which a product is received, is an empirical measure of robustness. If the manufacturer piled boxes N layers deep for shipment, and they arrived intact, it should be OK for the builder of a mixed pallet to do likewise.

RelativeRobustness is an enumeration of VeryWeak, Weak, Normal, Strong, and VeryStrong. This is in order of increasing robustness, but no quantitative meaning has been assigned.

4.4 Additional Metrics

Several additional metrics may be useful, as follows.

The four metrics listed above for connections and overlap are currently calculated only for the bottoms of boxes. It may be useful to calculate them for the tops as well.

The point of finding connections and overlaps is to get a measure of how well the stack holds together. It may be useful to calculate a more direct measure of holding together, such as the number of connected sub-areas on the top side of a layer (fewer being better). Intuitively, a sub-area is formed by a number of boxes in a layer being linked together by boxes in the layer above.

It may be useful to calculate the center of gravity (COG) of the stack of boxes. The main concern with the COG is that the pallet may be hard to handle if the COG is high or is off to one side of the pallet. Assuming that the COG of each box is at its center, the COG of the stack should be easy to calculate.

To unload a pallet efficiently, it should be possible to unload all boxes that have the same destination at once without having to move boxes that have a different destination. To support this idea, the schema Article.xsd defines Family as an element of Article. A metric could be calculated giving the number of Families that can be unloaded from a pallet without moving other boxes.

Overhang is a measure of the stack extending outside the pallet. It might be calculated as the sum over the four sides of the pallet of the maximum distance any box extends past the side. Many shippers, including the US Postal Service [8] and UPS [9], require that there be no overhang. The Great Little Box web site [6] says overhang is bad: “With as little as 1/2 inch hanging-over, as much

as 30% of their strength is lost!”. In some situations, however, overhang may be acceptable or desirable.

It will be useful to calculate the number of planning errors automatically. Planning errors include having boxes floating on air, planning to put a box on the stack before all the boxes it rests on are on the stack, having boxes intersect, failing to put all the boxes on the pallet that belong there, and putting boxes on the pallet that do not belong there. There may also be syntax errors in the XML file representing the plan; non-fatal errors could be counted.

4.5 Other Considerations of Metrics

We have not yet tackled metrics for multiple pallets. For example, if more than one pallet is required, is it going to be better to have roughly equal loads on all pallets than to load all but the last pallet full and put the remaining boxes (which may be few in number) on the last pallet?

For some metrics, such as determining automatically whether loading paths are collision-free, it may be necessary to incorporate a solid modeler.

The representation of a plan we are currently using allows placing a box in only two orientations – bottom down with sides parallel to the sides of the pallet. It would be practical to allow a wider range of orientations, but calculating metrics in that case will probably require using a solid modeler.

It is practical to treat differences of a millimeter or so in height as unimportant in building stacks of boxes of the size typically stacked on pallets. However, it is not clear how to model a stack with non-zero but negligible height differences or how to calculate metrics in this case.

A metric for spacing between the sides of boxes should be devised. Some automatic planning systems leave spaces between the sides of boxes and have the edges of every layer line up with the edges of the pallet. An example of this is shown in Figure 1(a). Other planning systems leave as little space as possible between the sides of boxes (see Figure 1(c)). Some boxes may be safely manipulated by suction grippers holding on to the top of the box. For other boxes, side and/or bottom gripping is necessary. In the latter case, it may be necessary to leave spaces between boxes for the gripper.

5 Evaluation of Metrics through Simulation

5.1 Pallet Viewer

The Pallet Viewer utility displays in a 3D color view a pallet and the as-planned stack of boxes on it. The figures in this paper are all images produced by Pallet Viewer. The Pallet Viewer executable is called with at least two arguments: the name of an order file and the name of a plan file. A third, optional, argument may also be given: the name of an as-built file. If the third argument is given, the Pallet Viewer shows the as-built stack in a color wire frame view off to the side

of the as-planned stack. In addition, the Pallet Viewer calculates and displays eleven metrics for the as-planned stack. These are:

- Number of file errors - total number of overlap errors in the file.
- Total weight - total weight of boxes on the pallet.
- Stack height - from the top of the pallet to the highest point on the stack.
- Volume of boxes - total volume of all the boxes on the pallet.
- Pallet storage volume - area of the pallet times the stack height.
- Volume density - volume of boxes divided by the pallet storage volume.
- Package N overlap fraction - fraction of the area of the bottom of box N that sits directly on the pallet or the top of some other box.
- Package N connections below - number of boxes that box N is directly on top of.
- Pallet average overlap fraction - average of the overlap fraction of all boxes on the stack.
- Pallet average connections below - average over all boxes on the stack of the number of connections below. For boxes that are directly on the pallet, the number of connections is 1. For boxes that are on other boxes, the number of connections is the number of such other boxes.
- Current file error count - number of overlap errors up to and including the most recently placed box.

The stack of boxes shown in Pallet Viewer may be built or unbuilt by using keyboard keys. Each time the g key is pressed, the next box is added to the stack (until there is no next box). Each time the f key is pressed, the last box is removed from the stack (until no boxes are left). The eleven metrics are recalculated each time a box is added or removed.

The Pallet Viewer is a C++ program using OpenGL graphics. Manipulating the view is done entirely with the mouse, except that the h key returns the view to its default position. The stack may be rotated and translated. The view may be zoomed in and out.

The Pallet Viewer program is useful for analyzing the process of creating a pallet. Since metrics are calculated for placement of every box on the stack, the software can assess whether the stack will remain stable when it is being constructed. Another important aspect of palletizing is the order in which boxes are put on the stack. By observing the Pallet Viewer display an expert user can to determine if the planner is making motion planning hard or impossible.

5.2 USARSim

The Pallet Viewer application is used to evaluate the quality of a pallet plan based on the geometry of the objects and their placement. Its metrics apply to the geometry of a planned pallet and answer the question, “will this be a good pallet when built?” These metrics don’t answer the questions, “how well does this pallet lend itself to being built?” or “when the pallet was actually built, how good was it?” It is a static evaluation that does not evaluate dynamic effects

such as objects sliding, tipping or being crushed. It also does not evaluate the pallet building process; there may be many ways to stack objects onto a pallet to satisfy the plan, and some are more efficient than others.

An intermediate step toward dynamic simulation is to use Pallet Viewer to apply static quality metrics after each object has been stacked, so that problems with the intermediate condition of a pallet can be detected. However, since Pallet Viewer does only rudimentary geometric analysis, problems such as objects sliding will not be detected.

We use simulation to supplement Pallet Viewer's static evaluation with dynamic evaluation. There are two aspects of dynamic evaluation: qualitative visualization by an expert, and evaluation of the final as-built pallet that may be different from the planned pallet due to object slipping, tipping, crushing or misplacement.

A shortcoming of dynamic evaluation is that the source of problems can't be easily isolated to the pallet building process or the pallet plan itself. For example, if an object slips or tips, it could be due to an improper stacking order as chosen by the robot controller, or it could be due to the plan itself, and no stacking order would fix the problem.

We use the Unified System for Automation and Robot Simulation (USARSim) [5] to evaluate the pallet build process and resulting built pallet. USARSim runs in real time. As a consequence, the evaluation takes as long as the pallet build process, typically on the order of tens of minutes. This makes it unsuitable as a way to compare many different object stacking plans to quickly select the best plan for execution.

The physics simulation within USARSim is plausible, and includes effects such as friction and gravity that will make sliding and tipping problems evident. However, the fidelity of the simulation does not equal the real world, and lacks the capability to represent continuum mechanics effects like compression or buckling of objects on the pallet, or bending of the robot arm as it moves throughout the work volume. It should be considered as a tool to get figures of merit on the object stacking sequence and intermediate steps toward the complete pallet.

In addition to quantitative metrics that can be applied to physically plausible behavior in simulation, the USARSim visualization can be watched by an expert person who can make subjective observations. Such observations include statements like, "the pallet looks good, but when you try to build it, it falls over before you get finished," or "with the boxes arranged like this, the robot can't reach the far side without knocking over these boxes." Subjective statements like these are unsuitable for metrics, but they can be used as a check that the scores from quantitative metrics match up with an expert's opinion. If, for example, a metric scores a pallet highly when an expert disagrees, the source of the discrepancy can be tracked down and the metric can be revised.

Practical experience with USARSim has brought to light several issues that static analysis of ideal pallets cannot find. These include:

- objects being placed outside the robot's work volume, or at locations or orientations that are awkward for the robot to reach
- objects being knocked off by collisions with an object to be placed or the robot arm itself

- excessive motion resulting from conservative approach- and retract-points that could be optimized
- temporary sensitivity of objects to perturbations before they are stabilized by objects placed atop them

Through experience garnered by visual simulations, we hope to improve the pallet metrics so that they can be easily applied and yield results consistent with expert opinions.

6 ICRA Virtual Manufacturing Competition

The first real trial of the metrics being developed for this effort occurred during the 2010 Virtual Manufacturing Automation Competition (VMAC) [4] that was part of the IEEE International Conference on Robotics and Automation (ICRA) robot challenge. During this event, three different palletizing approaches were evaluated through the use of our metrics. These approaches included a university created neural network learning-based approach, a university created deterministic planning approach, and a commercial product that is commonly used by industry.

Participants were required to accept an eXtensible Markup Language (XML) based order file and to generate an XML based packing list. The order file contained information on the pallet (e.g. pallet size, maximum load, maximum

Table 1. Results of the 2010 Virtual Manufacturing Automation Competition. Both the Neural and Deterministic teams experienced errors in their plans for round 1, pallet 1. These errors were corrected by hand and the scores of our metrics were recomputed in round 1, pallet 1*. The Neural team was unable to produce any pallets for round 2 while the Deterministic team was unable to produce more than one pallet per order file and did not place all of the packages in round 2. Round 1's order file contained 40 packages and round 2's contained 79 packages. Entries containing a '-' indicate that the algorithm was unable to find a solution.

VMAC 2010 Results								
Round	Pallet	Team	Errors	Time (s)	Height (m)	Density	Overlap	Connections
1	1	Neural	15	2054	0.252	0.447	0.60	0.75
		Deterministic	17	58	0.168	0.671	0.52	1.43
		Commercial	0	10	0.167	0.663	0.93	2.20
	1*	Neural	0	2054	0.252	0.447	0.93	1.32
		Deterministic	0	58	0.168	0.671	0.92	2.07
		Commercial	0	10	0.167	0.663	0.93	2.20
2	1	Neural	-	-	-	-	-	-
		Deterministic	19	1500	0.403	0.68	0.64	1.00
		Commercial	1	28	0.375	0.79	0.90	2.26
	2	Neural	-	-	-	-	-	-
		Deterministic	-	-	-	-	-	-
		Commercial	0	NA	0.083	0.23	0.99	1.00

stacking height) as well as various package classes (e.g. package size, weight, fragility, family) and the quantity of each class of package that was required. The team-generated packlist file was required to specify the sequence in which the packages should be sent to the palletizing system as well as each package's final 4 degree-of-freedom (4-DOF) resting location (x , y , z , and yaw) and a series of 4-DOF approach points to guide the robot arm in package placement. The teams were also notified of the metrics that would be used to judge the competition.

The desired approach was to run all three algorithms on increasingly difficult order files. After each round, the teams would be allowed to build their pallets in simulation using the USARSim framework. Teams that were successful in simulation would then be allowed to construct their pallets on our 1/3 scale real palletizing cell. The results from the two rounds of the competition are shown in Table II. As may be seen from this table, only the commercial software was able to place the full complement of boxes on pallets for both rounds. The Neural software never converged to a solution for the larger package order, and the Deterministic software was unable to plan for a second pallet that would be required to accommodate all of the packages. Due to only having one successful algorithm in round 2, order files of higher complexity were not attempted.

6.1 Round 1 Details

All of the teams successfully created packing plans for the round 1 order. Details of the results of applying the metrics are shown in Table II and images of the solutions are shown in Figure II. As shown in Table II, the Neural and Deterministic teams suffered from multiple errors that significantly affected their overall performance. These errors were due to the placement of packages in either a manner that would cause collisions in a real implementation of the plan or left the package suspended in mid-air. During the determination of the scores of our

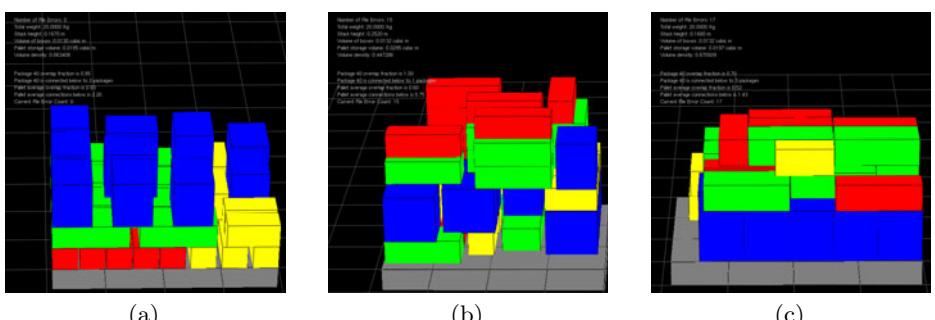


Fig. 1. Results from “round 1” of the competition which consisted of 40 packages to stack. Image (a) depicts the pallet produced by the commercial software package. Image (b) depicts the pallet produced by the neural network learning algorithm, while image (c) depicts the pallet produced by the deterministic approach.

metrics, packages which caused errors were given reduced credit (based on the portion of the box in error) for the computations. Table II Round 1, pallet 1* shows the improvement in scores that is possible if the package placements are adjusted slightly to avoid collisions. The remainder of this section will examine the corrected plans and the resulting higher scores.

From an execution time perspective, the commercial solution is a clear winner as it is close to six times faster than its nearest competitor. However, if time is discounted as a major scoring criterion, the results become much less clear. The scores for density, overlap, and package connections for both the commercial package and the deterministic algorithm are very similar. However, this similarity of scores using our metrics does not appear to indicate that the constructed pallets are similar. Figure II(a) shows that the commercial system has distributed the top layer of packages over the entire surface of the layer while Figure II(c) shows that the deterministic algorithm has concentrated the top layer towards the center of the pallet. This difference in the distribution of weight over the pallet will affect pallet stability and is not captured by the currently implemented set of metrics. The discrepancy between the difference in the pallet appearance and the scores on the metrics show that more work is required to develop a set of metrics that captures all aspects of the constructed pallet. In fact, a palletizing expert will need to be consulted to advise which pallet should indeed be deemed superior. Through the use of the pallet viewer software, we are able to easily view and share these results without the need for costly construction and evaluation of actual pallets.

6.2 Round 2 Details

The round 2 pallet order contained too many packages for a single pallet. The commercial software again computed a solution while the two university

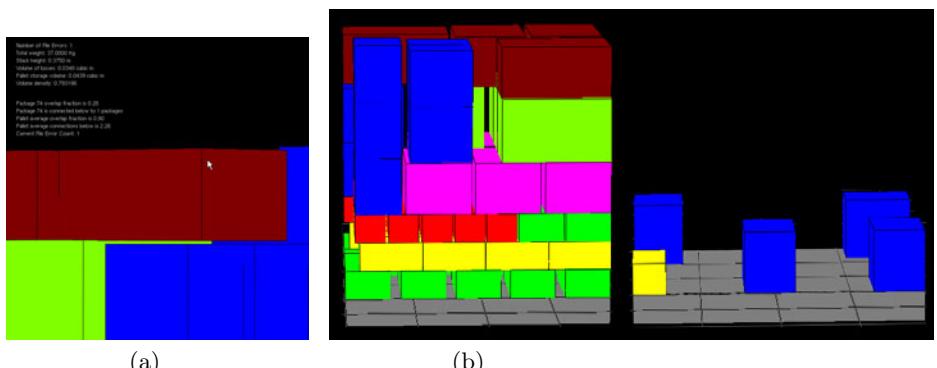


Fig. 2. (a) Error or no error? The metrics software designates an error for the top right package since it does not sit on both of its support packages. In reality, this box would probably lean to be supported by both of its supports. (b) The commercial software constructed these two pallets to solve the round 2 problem.

algorithms had more difficulty. The neural network learning algorithm was unable to converge to a solution and the deterministic approach was only able to complete the first of the two pallets. However, this round of the competition also raised some interesting problems for our metrics. Figure 2 shows the package that caused an error to be generated for the commercial system. In Figure 2(a) a slight gap can be observed between the top package and the supporting package on its right. Under real conditions, this package would simply “settle” onto its supporting package and have a slight tilt. The question that needs to be answered is if this arrangement will be stable or will cause the box to slide off of the pallet. Construction of the pallet by USARSim and the evaluation of the as-built pallet can answer this question.

Figure 2 shows the two pallets that were constructed for this order by the commercial software. The first pallet’s height was maximized, while the second pallet was left mostly empty. In addition, the first pallet’s construction contained numerous “towers” of packages where each package was only supported by a single package (no interlocking). The currently defined metrics were unable to judge the joint quality of this solution and instead were forced to judge the pallets individually.

7 Conclusions

We have adopted a model of palletizing, used it to define metrics, and held a palletizing competition. A Pallet Viewer utility has been built that calculates the metrics, displays them, and displays a naive simulation of executing a palletizing plan. The USARSim system has been used to simulate execution of a palletizing plan more realistically and produce an as-built description of a stack of boxes on a pallet. We have collaborated with industrial partners to ensure that our work is practical.

While we have accomplished much, more remains to be done, and we plan to continue our work. We are aware of quite a few additional metrics that might be calculated. We need to consult with more commercial firms to determine what metrics are important in what environments.

The palletizing model we are using is limited in the following ways:

- It does not support loading more than one pallet at a time.
- It supports placing a box in only two orientations.
- It mixes the design of a finished pallet with the plan for producing the design, which allows little freedom in the way in which the plan is realized.
- It does not provide for using any buffer space where boxes might be stored temporarily during palletizing.
- It does not allow for using more than one robot to load a pallet.
- It does not allow for expressing a constraint on the minimum space between boxes (which may be necessary for automatic equipment).

We need to improve our palletizing model so that it addresses these limitations.

References

1. Baltacioglu, E., Moore, J.T., Hill Jr, R.R.: The distributor's three-dimensional pallet-packing problem: a human intelligence-based heuristic approach. International Journal of Operational Research 1(3), 249–266 (2006)
2. Bischoff, E.E.: Stability aspects of pallet loading. OR Spectrum 13(4), 189–197 (1991)
3. Carpenter, H., Dowsland, W.B.: Practical considerations of the pallet-loading problem. Journal of the Operational Research Society 36(6), 489–497 (1985)
4. VMAC Executive Committee: Virtual manufacturing automation competition homepage (June 2010), <http://www.vma-competition.com>
5. USARSim Community. Usarsim wiki homepage (June 2010),
http://usarsim.sourceforge.net/wiki/index.php/Main_Page
6. Great Little Box Company. Palletization (June 2010),
<http://www.greatlittlebox.com/packaging/storing>
7. Kuka SysTec GmbH. Xml schema for article (June 2010),
<http://moast.cvs.sourceforge.net/viewvc/moast-devel/src/manufact/sampleData/Schema/Article.xsd?view=markup>
8. US Postal Service: Special standards - pallets, pallet boxes, and trays on pallets (June 2010), <http://pe.usps.com/text/QSG300/Q705b.htm>
9. UPS. Ups: Preparing your freight shipment (June 2010),
<http://www.ups.com/content/us/en/resources/ship/packaging/prepare/freight.html>
10. Yaman, H., Sen, A.: Manufacturer's mixed pallet design problem. European Journal of Operational Research (January 2008)

Analysing Mixed Reality Simulation for Industrial Applications: A Case Study in the Development of a Robotic Screw Remover System

Ian Yen-Hung Chen¹, Bruce MacDonald¹, Burkhard Wünsche¹,
Geoffrey Biggs², and Tetsuo Kotoku²

¹ University of Auckland, New Zealand
`{i.chen,b.macdonald}@auckland.ac.nz,`
`burkhard@cs.auckland.ac.nz`

² Intelligent Systems Research Institute, National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba, Japan
`{geoffrey.biggs,t.kotoku}@aist.go.jp`

Abstract. A Mixed Reality (MR) simulation aims to enable robot developers to create safe and close-to-real world environments from a mixture of real and virtual components for experimenting with robot systems. However, the reliability of the simulation results and its usefulness in solving practical problems remain to be validated. This paper presents an evaluation of an MR simulator by examining its use for the development of a robotic screw remover system. Quantitative evaluation compares the robot's trajectories produced in our MR simulation with those from a real world experiment, yielding results that indicate the MR simulation reliably represents the real world. A user study was conducted and the results demonstrate that the MR simulator gives users a stronger confidence of accurate results in comparison to a virtual simulator.

1 Introduction

While virtual simulation is useful for debugging and prototyping, it does not eliminate the need for eventual testing in the real world. A common problem faced is the discrepancies between the results produced in simulation and those from the actual operation. Mixed Reality (MR) simulation [5] is proposed to facilitate reliable transfer of simulation results to reality by bridging the gap between virtual simulation and real world tests. The simulator is founded on the concept of mixed reality [7] and enables developers to design a variety of scenarios for evaluating robot systems involving real and virtual components.

MR simulation can be considered as a form of hardware-in-the-loop simulation. In addition, it stresses the importance of seamless integration between the real and the virtual world to create a coherent environment for experimentation. Without strong coherency and synchronisation, such simulations could generate unnatural behaviour and produce inaccurate results that are unreliable than

other forms of simulation. In our MR simulator, a server maintains a representation of the physical environment where the experimentation takes place. With a model of the real world available, interactions with objects in the virtual world can be simulated based on the framework described in [6].

Thorough evaluation of a simulation tool is important to extrapolate its use in practice. USARSim [3] is a virtual robot simulator that has undergone extensive validation. Efforts have been put into assessing the accuracy of its sensor, actuator, and environment models. However, similar efforts have not yet been applied to evaluation of MR/hybrid environments designed for robot development. While these experimentation methods exist, e.g. [9], the work has been limited to system design and implementations.

Equally important is the need for evaluating the visual interfaces of a simulator. A high fidelity robot simulator is of little use if the users cannot extract meaningful information from the simulation. The proposed MR simulator adopts Augmented Reality (AR) and Augmented Virtuality (AV) visualisation techniques which need to be evaluated on their effectiveness in conveying robot and context information to developers. Similar work was done by Nielsen *et al.* [8] who evaluated their 3D AV interface for assisting robot teleoperation tasks by comparing it with a traditional 2D interface. In contrast, we compare the AR and AV interfaces provided in our MR simulator to identify the strengths and weaknesses of each approach.

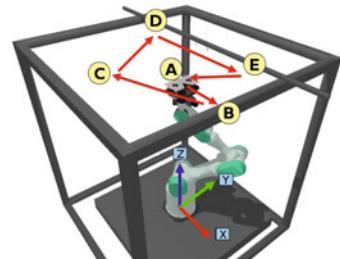
This paper presents an evaluation of MR simulation by case study. The goal is to assess the use of MR simulation for solving real world problems commonly faced during the robot development process. In this evaluation, an MR simulator is used to aid the development of a ceiling beam screw remover system to be deployed during the building demolition and renovation process. The remainder of the paper is organised as follows. Section 2 gives a background of the screw remover project. Section 3 describes the MR simulation created for the screw remover system. Section 4 presents results from quantitative evaluation. Section 5 describes the user study and its findings.

2 Screw Remover Project

In Japan, building interior renovation commonly requires removing equipments, such as lights and air conditioning vents, mounted on suspended beams of the ceiling. The old ceiling panels attached to the beams by self-tapping screws also need to be removed. A construction worker must then remove these screws by hand. This is a long and physically demanding process, due to the position of the screws above the worker's head. It is, however, simple and repetitive, making it ideal for automation. A ceiling beam screw removal robot [2] is proposed for the screw removal step. Rather than cutting or pulling the screws from the beam, which damages both the screws and the beam, the system uses a more sustainable solution, using a custom screw removal tool to unscrew the screw, leaving the materials in a reusable condition. The robot used is a Mitsubishi PA10 7 degree-of-freedom industrial robot manipulator. Mounted on the robot's



Fig. 1. A typical screw remover system test setup. **Fig. 2.** Screw removal operations Right: Ceiling beam and screw removal tool.



end-effector are a custom screw removal tool, a force-moment sensor, and a laser scanner (see Fig. 1). The custom screw remover tool consists of two rotating wheels that grip and turn the screw to remove it as the tool moves along the ceiling beam. The force-moment sensor is used to detect contact between the tool and the screws and in keeping the tool pressed against the beam. The laser is used to locate the beam above the robot and align the tool with the beam.

2.1 Screw Removal Operations

To perform the screw removal task, a human operator is required to press buttons on a Graphical User Interface (GUI) to start and stop the system. Once the operator starts the system, the robot will autonomously carry out the screw removal process. The operator monitors the robot operations and stops the system when the screws have been removed. The operations are shown in Fig. 2.

- **A to B:** The operator starts the system. The end-effector moves in the x -direction and takes three to five laser scans to locate the beam above.
- **B to C:** The end-effector moves to a position slightly below the beam.
- **C to D:** The end-effector approaches the beam at constant velocity until the force-moment sensor detects contact with the beam.
- **D to E:** The end-effector moves along the beam to remove screws.
- **E to A:** The operator stops the removal process and sets the robot to return to the starting position.

3 Mixed Reality Simulation

MR simulation integrates virtual resources into the real experimental setup to provide a cost-effective solution for experimenting with robot systems. The use of MR simulations can aid in the testing of the screw remover system because:

- There is a reasonable level of risk involved. The PA10 is a powerful robot and can cause harm to the human operator and the environment during development. Dangerous or expensive components can be virtualised in MR simulation to ensure safety.

- Testing of the system consumes resources such as spare beams and screws, which can be damaged by the robot due to errors in the system. The tool's wheels become worn out over time and need to be replaced. MR simulation is able to minimise the resources consumed by simulating these materials.

In this MR simulation, we are interested in studying the behaviour of the robot in the real world, making sure the controllers are generating the correct movements before moving onto a completely real world test. We would also like to minimise resource requirements in the testing process. Therefore we chose to virtualise the ceiling beam and the screws. The components in this setup are:

Real: robot manipulator, screw removal tool, laser sensor, force-moment sensor, and frame (for holding the ceiling beam).

Virtual: a ceiling beam fitted with three screws.

It is important that in an MR simulation, real and virtual objects are able to seamlessly interact with one another as if they exist in a coherent environment. To facilitate interaction between the real robot components and the virtual beam and screws, the sensors and actuators of the screw remover system need to be augmented with virtual information. The real laser needs to detect the location of the virtual beam as well as the real frame, while the real force-moment sensor needs to detect contact between the screw removal tool and the virtual beam as well as other external forces in the real world. The motion of the real robot manipulator will also be influenced by friction between the real screw removal tool and the virtual beam, and contacts between the screw removal tool and the virtual screws. Interaction between real and virtual components has been created based on the framework proposed in [6]. Exact simulation of the actual screw removal process (wheels gripping and turning the screw) is considered unnecessary for this application as we are interested in simulation at an integration level. Thus, the screw removal process is simplified at the cost of lower simulation accuracy. In our simulation, once the screw has a force exerted on it exceeding a threshold over a certain amount of time, the screw falls from the beam.

To create an MR simulation for the screw remover system that runs on the OpenRTM framework [1], the MR simulator described in [5] has been enhanced with the support of simulating OpenRTM-based systems, and the simulator is now based on the Gazebo simulation environment for OpenRTM [4]. OpenRTM-aist is a distributed, component-based framework. Each part of the system is implemented as a separate software component. These components communicate over known interfaces. Fig. 3 shows the system diagram of the components in the MR simulation. The screw remover system now exchanges data with MR sensors and an MR robot manipulator, instead of reading data from a real laser or force-moment sensor and sending commands to the real PA10. In summary, MR sensors make use of the real sensor components, modify their raw data with inputs from the virtual world created using Gazebo, before publishing to the connecting components. The MR robot manipulator models any physical contacts with virtual objects then sends commands to move the real PA10 accordingly.

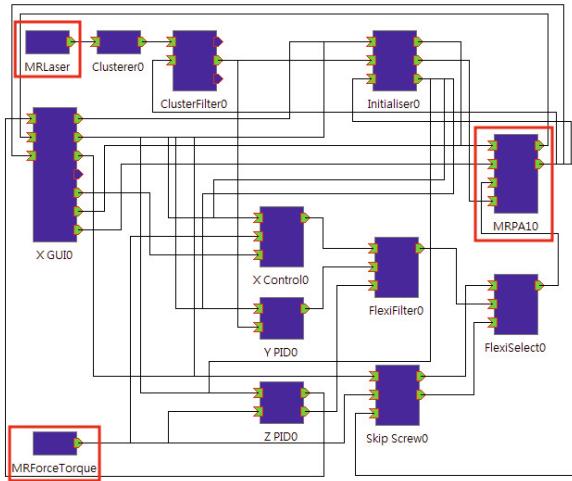


Fig. 3. System diagram of components in the MR simulation. The components highlighted in red are MR sensors (MRLaser, and MRForceTorque) and an MR robot manipulator (MRPA10) that exchange data with the original screw remover system.

3.1 MR Interfaces

The AR and the AV interface provided are shown in Fig. 4. In the AR interface, virtual objects are overlaid in geometric registration onto the real world scene using a markerless AR technique that tracks point features in the scene to compute the camera pose. A fixed overhead camera is used to capture a view of the whole experimentation environment which enables the users to monitor the overall robot manipulator's movement and the screw removal process.

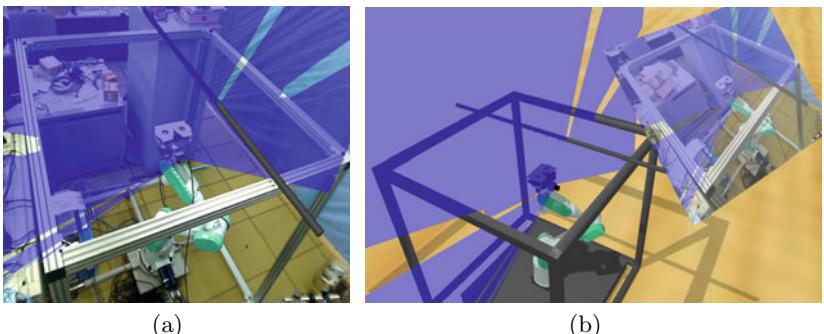


Fig. 4. Screenshots of MR interfaces: a) AR interface showing visualisation of laser data, virtual beam, and virtual screws in context with the real world, b) AV interface showing the simulation in a virtual environment augmented with real world laser and camera data

The AV interface provides a view in a virtual environment augmented with real world information. It is designed based on the ecological interface paradigm [8] that integrates multiple sensor information in an integrated display. A free-look camera is available in the AV interface that can be controlled to move freely within the environment for observing the simulation.

4 Quantitative Evaluation

An experiment was conducted to compare results obtained from the MR simulation and a real experiment. Each experiment was run five times, and the trajectories taken by the end-effector of the robot manipulator recorded. The time for a single screw removal was set to be 6 seconds in the MR simulation.

The average trajectories are shown in Fig. 5. One thousand evenly spaced sample points were taken from each trajectory to calculate the averages. The results show that the average path taken by the robot in the MR simulation closely resembles the one in the real experiment. This indicates that the registration of the virtual beam and screws in the real world is accurate and the robot successfully detected these virtual objects. Note that the deviation in the last segments of the trajectories between MR simulation and the real experiment are caused by the inconsistency of the human operator when signaling the robot to stop the screw removal operation and return to the starting position. The Root Mean Square Error (RMSE) between the two average trajectories was calculated by comparing the distances between the corresponding sample points from the two paths, and is found to be 18.08mm. The RMSE is approximately 1.10% with respect to the average distances travelled by the end-effector, which is 1621.79mm in MR simulation, and 1643.79mm in the real experiment.

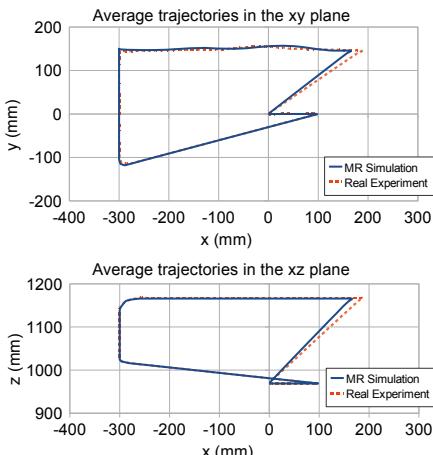


Fig. 5. Comparison between the average trajectories of the robot's end-effector in xy - and xz -plane

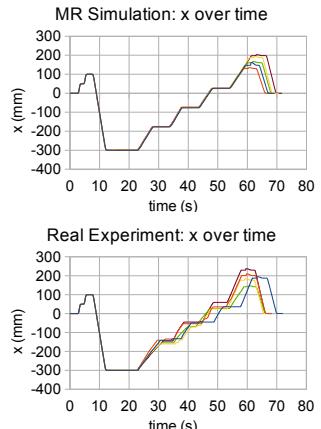


Fig. 6. Raw trajectories of the robot's end-effector over time

Table 1. Average Times

Screw Removal Task Completion		
MR Simulation	6.00 s/screw	69.97 s
Real Experiment	4.90 s/screw	68.95 s

Fig. 6 shows the trajectories over time for the five runs. As the robot’s end-effector moves along the beam in the x -direction to remove screws, a step-like pattern can be observed. The results clearly show that more variations occur in the real experiment which the MR simulation is unable to produce due to the simplified screw removal process and the fixed screw removal time. Table II shows the average times for removing a single screw, and for completing the entire task. The average time for removing a single screw in the real experiment is shorter than predicted, resulting in a short task completion time compared to the MR simulation.

5 User Study

A within-subject user study was conducted to compare the user’s experience in using MR simulation with a purely virtual simulator for performing a robotic task. Each participant carried out the task twice: once in the virtual simulation, and once in the MR simulation. To minimise any learning effects, some participants carried out the task in virtual simulation first, while some started with MR simulation.

In virtual simulation, the participants focused on a single virtual interface for carrying out the specified task. In comparison, MR simulation provides two interfaces: the AR interface and the AV interface. The participants were free to switch between the two interfaces provided for completing the task in MR simulation. The user’s experience in using the two provided interfaces was also compared. Desktop videos were taken to examine the participants’ actions and use of the provided interfaces when carrying out the task.

5.1 Task

The participants played the role of the human operator in the screw removal task outlined in Section 2.1. They were required to operate the screw removal robot and monitor its operation to ensure the robot completed its task correctly. The participants were first told about the screw remover project and the procedure for removing screws using the PA10 robot. The procedure involved the user clicking on buttons on a GUI for 1) initialising the system, 2) starting the screw removal operation, 3) stopping the screw removal operation, and 4) returning the robot to the starting position.

The participants were then introduced to the MR simulator and the virtual simulator, Gazebo. They were given the opportunity to familiarise themselves

with the keyboard and mouse controls of Gazebo that move a free-look camera for observing the simulation environment. Note that the same free-look camera was also available for them to use in the AV interface of MR simulation.

The operation of the screw removal task in virtual simulation was straight forward, with the participants using the GUI and the free-look camera for completing the task. In MR simulation, the participants were asked to perform an extra step that merges the real world and the virtual world. On the live video imagery captured by the camera overlooking the physical experiment environment, the participants needed to click with a mouse on the four corners of the frame that holds the ceiling beam. This calibrated the camera and registered the virtual objects into the real world for setting up the AR interface. The participants could then switch between the AR interface and the AV interface using the keyboard.

After completing the task using both simulation methods, the participants were asked to fill a questionnaire, followed by a short interview investigating any problems that they encountered when using the two simulators.

5.2 Dependent Variables

The time for task completion was recorded. The task completion time includes the time taken to position the free-look camera before operating the robot, and any time taken by the participant to observe the simulation before confirming the completion of the operation. The task completion time excludes the time taken to set up the AR interface in MR simulation (users manually clicking on four corners for calibrating the camera), which was recorded separately.

The time spent on each visual interface in MR simulation was also recorded to investigate if there are any preferences in using the provided interfaces for this specific task.

Paired t-tests are used to identify any significant differences between the times in this repeated-measures user study. A p-value smaller than 0.05 suggests the difference is significant.

5.3 Questionnaire

The questionnaire comprised three sections. Section one collected the participant's demographic information. Section two measured the participant's experience in using virtual simulation and MR simulation. Section three measured the user's experience in using the AR interface and the AV interface for carrying out the specified task. A 7-point Likert scale was used throughout the questionnaire for measuring the user's experience.

Two-tailed Wilcoxon Signed Rank tests are used to identify any significant differences between the users' experiences in using virtual simulation and MR simulation, and similarly, the users' experiences in using the AR interface and the AV interface.

5.4 Hypotheses

1. Virtual simulation offers a safer environment than MR simulation for robot development.
2. MR simulation is a more accurate representation of the real world than virtual simulation.
3. MR simulation produces more reliable results than virtual simulation.
4. The AR interface provides a more intuitive display of information than the AV interface
5. The AV interface provides a more flexible view of the environment than the AR interface
6. The AV interface increases the user's awareness of the robot status and potentially dangerous situations.

5.5 User Study Results

11 participants were recruited for this study. Out of the 11 participants, 10 participants had experience in computer programming and/or robot development (Computer programming: 9 participants with mean 14.10 and Standard Deviation (SD) 6.12 years of experience; Robot development: 8 participants with mean 6.83 and SD 6.85 years of experience) and 8 had used a simulation tool before for the development of computer/engineering systems. One participant had no experience in computer programming or robot development.

6 participants were randomly assigned to carry out the task in virtual simulation first, while 5 participants began with MR simulation.

Simulation Experience. The users completed the task faster in virtual simulation (mean 90.73 seconds, SD 13.75) than in MR simulation (mean 102.00 seconds, SD 23.21). Although the difference is not significant ($t = -1.49$, $p = 0.17$), it was observed that the participants were being more cautious in MR simulation since a real robot was used, taking more time to observe the simulation environment through the interfaces provided and keeping an eye on the robot in the real environment throughout the task.

The questionnaire results on simulation experience are shown in Fig. 7. The users rated MR simulation significantly higher than virtual simulation for debugging ($Z = -2.33$, $p < 0.05$). The high rating is believed to be leveraged by the use of the AR interface in MR simulation, which helped users understand complex robot data.

The questionnaire results suggest that virtual simulation is a safer method than MR simulation and helps to minimise any potential harm. Despite the finding being statistically insignificant ($Z = -0.96$, $p = 0.33$), using a real robot in MR simulation does have an impact on the safety of the experiment. In the later interview we found that 4 participants expressed concern about the robot colliding with other real world objects in the environment, such as the ceiling beam frame, and 3 of whom reported being nervous while the screw removal operation was taking place in MR simulation.

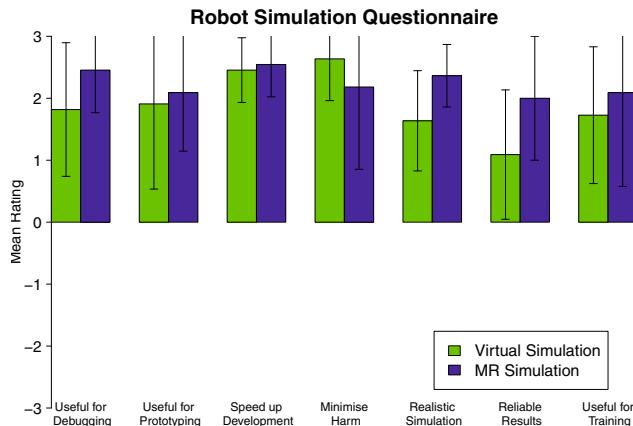


Fig. 7. Differences between the users' experiences in using virtual simulation and MR simulation

MR simulation was rated significantly higher than virtual simulation for producing a more realistic simulation ($Z = -2.06$, $p < 0.05$), and more reliable results ($Z = -2.46$, $p < 0.05$), hence accepting hypothesis 2 and 3. The finding is particularly encouraging as it indicates that MR simulation gave users a stronger confidence of accurate results, potentially allowing the simulator to be used for extended testing and evaluation of robot systems beyond the prototyping stage.

MR simulation was rated higher than virtual simulation for prototyping ($Z = -0.53$, $p = 0.60$), speeding up development ($Z = -0.58$, $p = 0.56$), and teaching and training ($Z = -0.97$, $p = 0.33$), but the differences are not significant.

MR Interface Experience. On average, the user focused more on the AR interface (mean 61.73 seconds, SD 26.18) than the AV interface (mean 40.27 seconds, SD 22.64) for carrying out the screw removal task, but the difference is not significant ($t = 1.65$, $p = 0.13$). The mean time for setting up the AR interface before starting the task in MR simulation is found to be 14.73 seconds (SD 5.59); the extra time is an overhead for using MR simulation and needs to be minimised in the future.

The questionnaire results on MR interface experience are shown in Fig. 8. The users rated the AR interface significantly higher than the AV interface for intuitive display of information ($Z = -2.07$, $p < 0.05$), and also commented that overlaying virtual robot data in the physical world made understanding robot data and debugging intuitive. Similarly, MR simulation also received a higher mean rating for learning robot data ($Z = -1.93$, $p = 0.05$), and identifying inconsistencies between the user's view and the robot's view of the world ($Z = -1.63$, $p = 0.10$), although not statistically significant at the $p < 0.05$ level, these two attributes are believed to be strengths of AR visualisation.

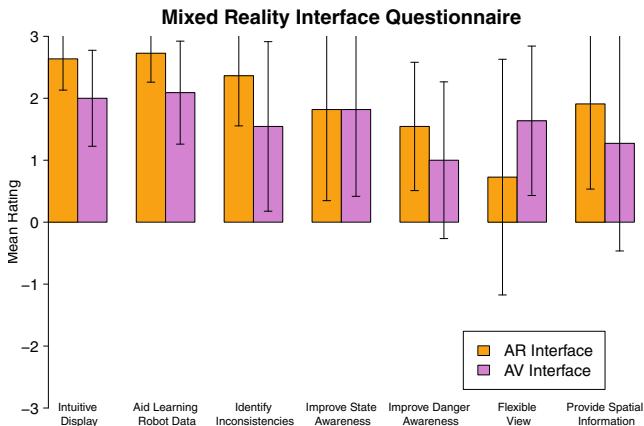


Fig. 8. Differences between the users' experiences in using the AR interface and the AV interface

The users rated the AV interface higher for providing a more flexible view than AR interface, as expected, but the difference is not significant ($Z = -1.73$, $p = 0.08$). However, a number of participants commented that they would prefer having the ability to move the physical camera around while using the AR interface rather than using a fixed camera. The results could differ if an on-board or movable camera was used. This needs to be further investigated.

Interestingly, a number of observations were not expected. The AR interface was rated equally important as the AV interface for improving awareness of robot states, and the AR interface was rated higher for improving awareness of dangerous situations ($Z = -1.19$, $p = 0.24$). This rejects hypothesis 6. Furthermore, the users rated the AR interface significantly higher for providing spatial information than the AV interface ($Z = -2.33$, $p < 0.05$). This was a surprise as the AR interface is only a 2D display of information.

Overall, the users rated in favour of the AR interface. The results are believed to be influenced by the user's unfamiliarity with the control of the free-look camera in the AV interface. Although the users were given time to practice until they were comfortable, the later interview found that 5 participants (45%) commented on the difficulty of using keyboard and mouse inputs in the AV interface. Suggestions include using a 3D mouse or a joystick as an alternative input device. Reviewing the recorded desktop videos discovered that many participants did not continue to move the free-look camera after the screw removal operation started. The free-look camera was left at a fixed position either looking over the entire simulation environment or zoomed in at the three screws to be removed. Two participants became lost in the AV interface and immediately switched to the AR interface to continue monitoring the screw removal operation. This finding suggests future work is necessary to improve the user input interface in order to exploit the full potential of the AV interface.

6 Conclusions and Future Work

While previous research has demonstrated various ideas for MR/hybrid simulations, we have for the first time presented an evaluation of an MR robot simulator in an industrial application. An MR simulation has been created for a screw remover project. Comparative evaluation shows an RMSE of approximately 1.10% between the average trajectories taken by the robot in the MR simulation and the real experiment. User study results reveal that users are positive about MR simulation and believe it offers a more reliable transfer of results to reality. However, the visual interfaces need improvements. The AR interface helped users understand and debug robot data but provided a limited view of the scene; a movable viewpoint needs to be considered in the future. The AV interface lacks an intuitive user input interface for observing the simulation environment; more natural interaction methods are necessary to improve the usability of the system.

Acknowledgments. The research was financially supported by the National Institute of Advanced Industrial Science and Technology, Japan, and the University of Auckland Doctoral Scholarship.

References

1. Ando, N., Suehiro, T., Kotoku, T.: A software platform for component based rt-system development: OpenRTM-aist. In: Carpin, S., Noda, I., Pagello, E., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2008. LNCS (LNAI), vol. 5325, pp. 87–98. Springer, Heidelberg (2008)
2. Biggs, G., Kotoku, T., Tanikawa, T.: Ceiling beam screw removal using a robotic manipulator. In: Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, St. Louis, USA, October 11-15 (2009)
3. Carpin, S., Stoyanov, T., Lewis, Y.N., Wang, M., J.: Quantitative assessments of USARSim accuracy. In: Proceedings of PerMIS 2006 (2006)
4. Chen, I., MacDonald, B., Wünsche, B., Biggs, G., Kotoku, T.: A simulation environment for OpenRTM-aist. In: Proceedings of the IEEE International Symposium on System Integration, Tokyo, Japan, November 29, pp. 113–117 (2009)
5. Chen, I.Y.H., MacDonald, B., Wünsche, B.: Mixed reality simulation for mobile robots. In: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2009, Kobe, Japan, May 12-17 (2009)
6. Chen, I.Y.H., MacDonald, B.A., Wünsche, B.C.: Designing a mixed reality framework for enriching interactions in robot simulation. In: Proceedings of the International Conference on Computer Graphics Theory and Applications, Angers, France, May 17-21, pp. 331–338 (2010)
7. Milgram, P., Colquhoun, H.: A taxonomy of real and virtual world display integration. In: Mixed Reality-Merging Real and Virtual Worlds, pp. 5–28 (1999)
8. Nielsen, C., Goodrich, M., Ricks, R.: Ecological interfaces for improving mobile robot teleoperation. *IEEE Transactions on Robotics* 23(5), 927–941 (2007)
9. Nishiwaki, K., Kobayashi, K., Uchiyama, S., Yamamoto, H., Kagami, S.: Mixed reality environment for autonomous robot development. In: Proceedings of the IEEE International Conference on Robotics and Automation, Pasadena, CA, USA, May 19-23, pp. 2211–2212 (2008)

A Parameterless Biologically Inspired Control Algorithm Robust to Nonlinearities, Dead-Times and Low-Pass Filtering Effects

Fabio DallaLibera¹, Shuhei Ikemoto², Takashi Minato⁴, Hiroshi Ishiguro^{3,4}, Emanuele Menegatti¹, and Enrico Pagello¹

¹ Dep. of Information Engineering (DEI), Faculty of Engineering,
Padua University,
I-35131, Padua, Italy

² Dep. of Multimedia Engineering,
Graduate School of Information Science and Technology,
Osaka University,

565-0871, Suita, Osaka, Japan

³ Dep. of Systems Innovation,
Graduate School of Engineering Science,
Osaka University,

560-8531, Toyonaka, Osaka, Japan

⁴ ERATO, Japan Science and Technology Agency,
Osaka University
565-0871, Suita, Osaka, Japan

Abstract. A biologically inspired control algorithm for robot control was introduced in a previous work. The algorithm is robust to noisy sensor information and hardware failures. In this paper a new version of the algorithm is presented. The new version is able to cope with highly non-linear systems and presents an improved robustness to low-pass filter effects and dead-times. Automatic tuning of the parameters is also introduced, providing a completely parameterless algorithm.

1 Introduction

Biologically inspired algorithms have been intensively studied in the robotics field. In particular, we focused on robotic control problems and in [4] we presented a simple biologically inspired control that is robust to environmental changes and noise. The algorithm takes inspiration from bacteria *chemotaxis*, the process by which bacteria [1,10,8] sense chemical gradients and move with directional preference toward food sources.

Among the most thoroughly studied organisms with regards to chemotaxis we can certainly cite *Escherichia Coli* (E. Coli).

This bacterium has only two ways of moving, rotating clockwise or counter-clockwise [1]. When it rotates counter-clockwise the rotation aligns its flagella into a single rotating bundle and it swims in a straight line. Conversely clockwise rotations break the flagella bundle apart and the bacterium tumbles in place.

The bacterium keeps alternating clockwise and counterclockwise rotations. In absence of chemical gradients, the length of the straight line paths, i.e. the counter-clockwise rotations, is independent of the direction. Thus, the movement consists in a random walk. In case of a positive gradient of attractants, like food, E. Coli instead reduces the tumbling frequency. In other terms, when the concentration of nutrients increases the bacterium proceeds in the same direction for a longer time. This strategy allows biasing the overall movement toward increasing concentrations of the attractant. Such a simple mechanism works despite the difficulties in precisely sensing the gradient. Actually, the spatial gradients in concentration cannot be sensed directly due to the small dimensions of the bacteria, so temporal difference in the concentration is used to estimate the nutrient distribution.

A wide spectrum of models, from a very abstract point of view to the modeling of the protein interactions are available in literature [9,27].

The movement of E. Coli was mimicked in robotics as well. In [5] it was shown that while gradient descent is faster for tracking a single source, biased random walks perform better in the presence of multiple and dissipative sources and noisy sensors and actuators. Furthermore the stochastic nature of the algorithm prevents it from getting stuck in local minima.

However, the robustness to hardware damages and noisy sensory information achievable by biased random walk were not fully exploited. Expressly, in [5] the hardware already provides two basic movements, proceed straight and change direction randomly, and the biased random walk is performed at the behavior level. This approach limits the robustness for unexpected hardware failures. In fact, if due to a hardware failure, the behavior corresponding to such commands will be different from the expected one, the task will not be accomplished. Imagine for instance to have a mobile robot with two wheels powered by independent motors and that due to an encoder problem one of the motors starts to rotate in the opposite direction. In this case when the “go forward” motor command is provided the robot spins around itself and the target will never be reached. With our approach [4], instead, a biased random walk is executed directly in the *motor command space*, i.e. the behaviors themselves are determined online through the random walk. This gives great robustness in case of hardware failures since new behaviors that exploit the current hardware behavior are found online by biased random walk. For instance, in case of the encoder failure described, the robot would explore new motor commands, until it finds that rotating the motors in opposite directions the distance from the target can be decreased.

Concretely in [4] the control of the robot reduces to the single equation

$$u_{t+1} = u_t + \alpha sgn(\Delta A_t) \frac{u_t}{\|u_t\|} + \beta \eta_t \quad (1)$$

where $u_t \in \mathbb{R}^m$ is the motor command control signal provided at time t . The equation is composed of two terms, a bias term $b_t = sgn(\Delta A_t) \frac{u_t}{\|u_t\|}$ and a purposely added random perturbation $\eta_t \in \mathbb{R}^m$, multiplied respectively by two scalar coefficients α and β .

The quantity $\Delta A_t \in \mathbb{R}$ appearing in the bias term expresses the variation of the “quality” of the robot state during the t -th time step. For instance, in [4] the task consists in reaching a red target, and the quantity ΔA_t represents the variation in the number of red pixels acquired by the camera image. Therefore, essentially, the bias term states that if the robot got closer to the target using command u_t then command u_{t+1} should be similar. The second term is the most interesting part. In [4] it was shown that by adding random perturbations of opportune magnitude to the control input, it is possible improve the performances.

Figure 1 reports an example. Let us suppose to have a holonomic robot initially placed at $[10, 10]^T$ (arbitrary units) that has to reach a target placed at $[0, 0]^T$, and the motor command to be simply the translations along the two axes. If the perturbation is too small, then the effect of the bias will be strong, and even nearly tangential movements that bring the robot slightly closer to the goal will be used for a long time. If the perturbation is too big then the robot changes direction too frequently, even when the movement is headed straight to the goal. If the perturbation amplitude is appropriate than the robot will reach the goal with a good trajectory. This result is confirmed observing the distribution of the robot heading direction compared to the optimal direction in Fig. 2. When the perturbation is too little essentially all headings that do not face backwards are chosen with uniform distribution. When the noise is too big, the robot tries all directions, with a small bias on the good ones. Finally, when the perturbation level is appropriate the distribution has a peak in the optimal heading.

In [4] the robustness of the algorithm was highlighted by experiments both with a simulated and a real robot. In particular it was shown that the algorithm is able to drive to the target a simulated robot that undergoes substantial damages. Secondly real world experiments proved that even very noisy information can be used to accomplish the task. Results showed that the performance depends essentially just on the ratio between the two coefficient α and β , and not on their values itself. However, the optimal ratio depends on the hardware and environment conditions. The next section will highlight three shortcomings of the previous algorithm, and section 3 will present a new algorithm that faces these aspects. Furthermore a criterion to automatically adjust the perturbation level is introduced. Successively section 4 provides a quantitative comparison of the two versions of the algorithms, and finally section 5 concludes summarizing the paper and presenting future works.

2 Generality Limitations of the Previous Version

In most systems opposite motor commands generate an opposite effects, at least in some regions of the motor command space, and that a scaled version of the motor command provides a similar effect with different intensity. Observing carefully Eq. 1 it is possible to observe that the algorithm previous presented partially exploits this fact. In fact it assumes that if command u_t is beneficial for the robot

¹ A stochastic resonance effect can be observed. For details see [3, 6, 12].

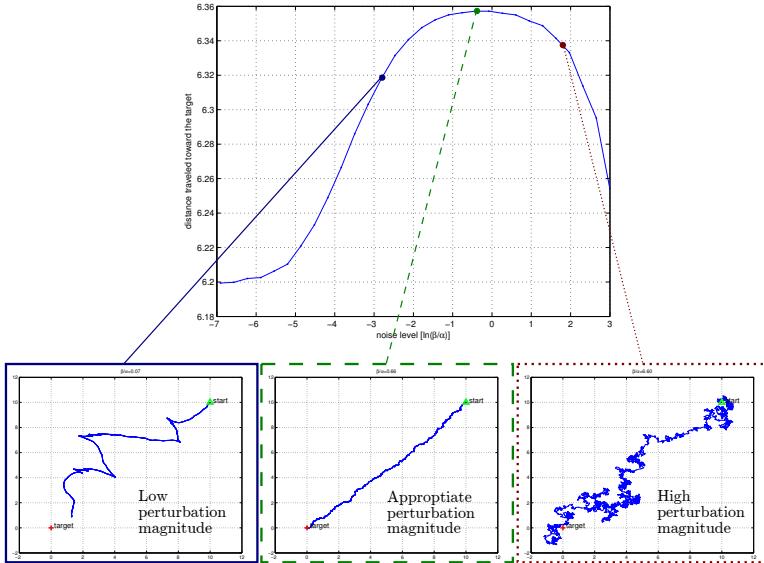


Fig. 1. Performance for different perturbation magnitudes. The top panel reports the performances for different values of β when $\alpha = 10^{-2}$. The performance is measured as the average distance traveled toward the goal in 1000 time steps, calculated over 10000 simulations. The bottom panels report examples of trajectories obtained for different values of β . Precisely, the first trajectory was obtained for a low perturbation level, $\beta = 0.07$, the second trajectory corresponds to an opportune level of perturbation, $\beta = 0.66$ and finally the bottom right panel reports a trajectory generated with $\beta = 6.6$.

state at time t , then $\frac{u_t}{\|u_t\|}$ will probably be a good bias for the following command u_{t+1} . Furthermore, if u_t worsened the conditions during time t then it is assumed that $-\frac{u_t}{\|u_t\|}$ will be an appropriate bias for u_{t+1} . These considerations however do not hold for many motor command spaces. Using the previous example, suppose again to have a holonomic robot whose coordinates are $x_t \in \mathbb{R}^2$. Assume now $u_t \in \mathbb{R}^2$ and its two components to represent the velocity in polar coordinates, i.e. $x_{t+1} = x_t + |u_t|^2 \begin{bmatrix} \cos(u_t^2) \\ \sin(u_t^2) \end{bmatrix}$. The non-linearity near the origin introduced by the cosine prevents the algorithm from being able to drive the robot to the goal.

Another disadvantage of using $-u_t$ as bias when the conditions worsen is that the control algorithm performs badly if there are dead times in the response. Imagine for simplicity a unidimensional case where the goal is at $+10$ and the position x_t , initially 0 , changes by $x_t = x_{t-1} + u_{t-2}$, $x_t, u_t \in \mathbb{R}$. In this case the performance increases when the bias is positive. Suppose to start with u_0 and u_1 negative, and suppose the random perturbations to be small enough that the sign of the motor command u_t is determined by the sign of the bias b_t . Since u_0 is negative ΔA_2 will be negative, and the bias b_2 and signal u_2 will become positive. However, in the next step the effect of u_1 will lead to a negative ΔA_3 ,

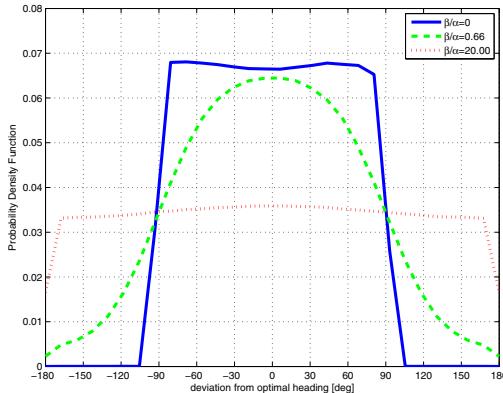


Fig. 2. Distribution of the robot heading, compared to the optimal heading, for different perturbation levels. The blue continuous graph reports the distribution for a low perturbation magnitude, the green dashed graph represents the probability density function obtained for an opportune perturbation coefficient ($\beta = 0.66$) and the third, red dotted curve, shows the distribution of the heading for an high perturbation level ($\beta = 20$).

which in turn will bring the bias b_3 and u_3 to become negative again. The effect of u_2 will provide a positive ΔA_4 , so the bias b_4 will have the same sign of u_3 , i.e. it will be negative. At this point the evolution of the bias will repeat, in a loop that contains two negative biases and a positive one. If the magnitude of the biases is similar for the positive and negative case, in general the bias will tend to bring the robot farther from the target instead of bringing it closer, and the system will never reach the target +10. Table II reports other examples of bias sequences that reveal to be a nuisance instead of being beneficial to reach the target.

For similar reasons the bias can become deleterious if the system includes delays introduced by low pass filters. This can be a serious disadvantage of the algorithm, since many physical systems present this kind of behavior. A simple example can be provided by introducing an Infinite Impulse Response filter in the example of a holonomic robot moving on the plane. Concretely, assume the robot to change its position by $x_{t+1} = x_t + v_t$ where $v_t = (1 - 10^{-\rho}) \cdot v_{t-1} + 10^{-\rho} \cdot u_t$.

Table 1. Bias sequences leading to performance decrease

u_0	u_1	Sequence
-	-	ΔA_t : -, -, +, ...
	u_t	+, -, -, ...
-	+	ΔA_t : -, +, -, ...
	u_t	-, -, +, ...
+	-	ΔA_t : +, -, -, ...
	u_t	-, +, -, ...

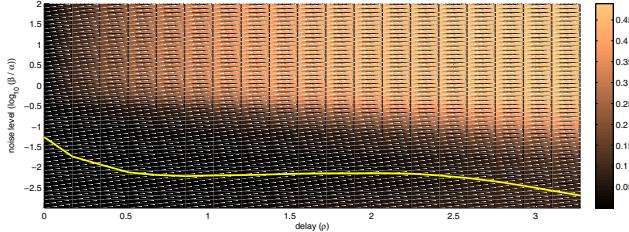


Fig. 3. Probability of biases that would bring the robot further from the goal. The X axis represents the delay level (ρ), the Y axis represents the perturbation level (β/α) and the color indicates the probability of bad biases (lighter color indicates a higher probability). The yellow line indicates the noise level that gives the lowest bad bias for each delay level.

Let us define a “bad bias” a bias that has a heading that differs more than 90 degrees from the optimal one and would therefore bring the robot further from the goal. Figure 3 reports how the probability of bad biases changes by varying the random perturbation level (β/α ratio) and the delay level (ρ). Although the probability of bad biases can be minimized by changing the random perturbation level, it can be observed that as the delay increases the probability increases. For instance, for a value of ρ equal to 3 the probability of bad biases cannot be lower than 0.09.

It is worth noting that for a wide range of problems it is often possible to find expedients that allow to mitigate the weak points presented in this section and that allow to use the previous version of the algorithm. In fact, adopting an adequate motor command space reveals to be sufficient in most of the cases. However, this paper aims at showing that the generality of the algorithm can be greatly improved without increasing the algorithm complexity.

3 A New Version of the Algorithm

As reported in the introduction, *Escherichia Coli* proceeds by movements in random directions, but when moving toward increasing concentrations of nutrients the movement in that direction is prolonged. A similar behavior can be obtained by taking $u_{t+1} = u_t$ if $\Delta A_t \geq 0$ and selecting u_{t+1} randomly² if $\Delta A_t < 0$. As in the previous version of the algorithm, a random perturbation can improve the performances. In particular, it is sufficient to add a perturbation to each of the components of the input when $\Delta A_t \geq 0$, i.e. $u_{t+1}^i = u_t^i + \eta^i R$, $R \sim \mathcal{N}(0, 1)$ for each of the components of the input ($1 \leq i \leq m$).

Choosing the bias at random when the system is getting further from the goal removes any assumption of linearity of the system. Clearly this leads to

² In the following we assume to select $u_t \in \mathbb{R}^m$ using a uniform distribution over the whole motor command space, but the results remain essentially the same using different distributions.

a performance decrease for systems that are effectively linear, but considerably improves the generality of the algorithm. Furthermore, in case of dead times periodic bias sequences with negative effects are unlikely generated. For instance, in the case of the unidimensional example provided in the previous section, if by chance a positive bias is followed by another positive bias then the system will keep a positive bias and reach the target³. Similarly, better performances are expected when delays arising from low pass filter effects are present. Imagine in fact to have a sequence of good motor commands that are not recognized as such because their effect comes later. In the meanwhile new commands will be generated. If the system responds with an opposite behavior when the input is negated (as for all linear systems and many other setups) then choosing a random command is less deleterious than choosing the negated motor command.

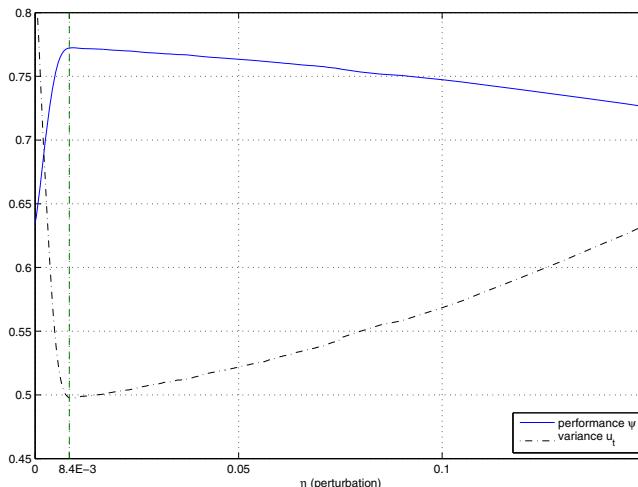


Fig. 4. Average performance and input variance obtained for different values of η . The graphs were obtained placing the robot in $x_0 = [10, 10]^T$ with $s = 10^{-6}$, simulating $N = 10^5$ steps and repeating the test 10^5 times.

Intuitively, the algorithm operates in a very simple way. It keeps using the same motor command as long as the command is beneficial, otherwise it picks up a new one at random. This provides intuition of how to adjust the magnitude of the random perturbations. Expressly, if the random perturbations are appropriate and in general good inputs are selected these will be used for a long time. Observing the variance of the produced motor commands we can therefore have an idea of the quality of the motor command. In order to dynamically adapt η_t^i we can therefore estimate the variance of u_t^i by picking some samples, slightly increase(/decrease) η_t^i , and estimate the variance of u_t^i again. If

³ As previously stated, assuming the perturbations to be small enough. Notice, however, that a sequence of two positive biases is sufficient for recovery if perturbations change the sign.

the variance decreased then we increase(/decrease) η_t^i once more, otherwise we decrease (/increase) it. This kind of effect can be showed by a simple example. Suppose as previously done to have a holonomic robot that must approach a target located in $[0, 0]^T$. To reduce the problem to a system with a unidimensional motor command, assume the robot to move by steps of fixed length s , along the angle indicated by u_t . Formally let $x \in \mathbb{R}^2$ be the robot position, $u_t \in \mathbb{R}$, $x_{t+1} = x_t + s \cdot \begin{bmatrix} \cos(u_t) \\ \sin(u_t) \end{bmatrix}$. Assume then to express the performance ψ as the average decrease in the distance to the goal for a single step over N steps, i.e. $\psi = \frac{\|x_0\| - \|x_N\|}{N \cdot s}$.

Fig. 4 reports the average performance ψ and variance of u_t for different values of η . We notice that the maximum performance corresponds to the minimum variance. For more complex setups the two peaks could not coincide, but choosing the perturbation that gives the lowest motor command variance appears to be a reasonable choice in most cases.

Assuming to estimate the variance using just two samples⁴ we derive the following algorithm

$$\begin{aligned} u_{t+1}^i &= \begin{cases} u_t^i + \eta_t^i R & \text{if } \Delta A_t \geq 0 \\ \text{random selection} & \text{otherwise} \end{cases} \\ \delta_0^i &= 1.1 \\ \sigma_t^i &= \frac{(u_t^i - u_{t-1}^i)^2}{2} \\ \delta_{t+1}^i &= \begin{cases} 1/\delta_t^i & \text{if } t \text{ odd} \wedge \sigma_t^i \geq \sigma_{t-2}^i \\ \delta_t^i & \text{otherwise} \end{cases} \\ \eta_{t+1}^i &= \begin{cases} \eta_t^i \delta_{t+1}^i & \text{if } t \text{ odd} \\ \eta_t^i & \text{otherwise} \end{cases} \end{aligned}$$

4 Experiments

As a first step we compared the performances of the two algorithms when coping with nonlinear systems. In this experiment, the movement of the robot was set to $x_{t+1}^i = x_t^i + s \cdot f^i(u_t)$ where $f^i(x) = \frac{1}{\pi} \arctan \left(\frac{(\sin(2\pi x + \xi^i))^T Q^i \sin(2\pi x + \xi^i)}{(\sin(2\pi x + \zeta^i))^T P^i \sin(2\pi x + \zeta^i)} \right)$. In this expression the \sin function is applied element-wise and $Q^i, P^i \in \mathbb{R}^{2 \times 2}$ and $\xi^i, \zeta^i \in \mathbb{R}^2$ were randomly initialized. Figure 5 shows the distance traveled toward the goal in different trials. In detail the robot was placed in 6 different initial positions ($10 \cdot [\sin(2k\pi/6) \cos(2k\pi/6)]^T$, $k \in \mathcal{N}, 0 \leq k \leq 5$) and for each position the experiment was repeated 10^4 times. We notice that as expected

⁴ A higher number of samples provides a better estimate of the variance and therefore of the variance change, but slows down the adaptation. Note that, however, two samples are always sufficient to guarantee a right estimation of the whether the variance increased or not with a probability higher than 0.5.

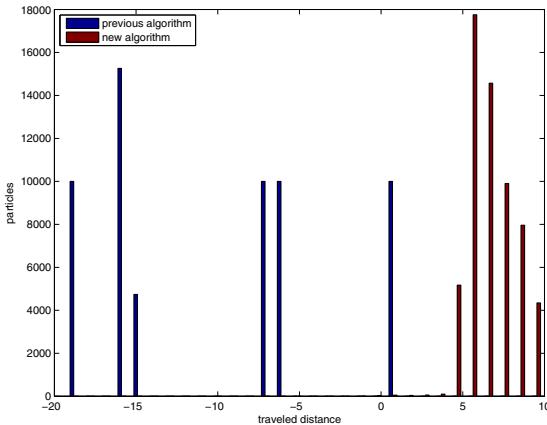


Fig. 5. Distribution of the distance toward the goal traveled in $N = 10^4$ steps of size $s = 10^3$ using the nonlinear functions $f^i(x)$. The robot was placed in 6 different positions and for each position the test was repeated 10^4 times.

the performance is generally higher for the newer version of the algorithm. The newer version of the algorithm is able to drive the robot to the goal even in case of the highly nonlinear mapping introduced in the experiment.

The second test deals with dead times in the system. Expressly, we simulated the case $x_{t+1} = x_t + u_{t-d}$, $d \in \mathcal{N}$, $x_t, u_t \in \mathcal{R}^2$, $-s \leq x_t^i \leq s$ for $N = 10^4$ time steps. As visible in Fig. 6, with the previous version of the algorithm the distance traveled toward the goal drops off as soon as there is a dead time d . The performance of the new version of the algorithm degrades as the dead time d increases, but does not reach 0, i.e. the algorithm is still able to drive the robot to the goal whatever the dead time is.

We then tested the algorithms on a system that includes a low pass filter as the one described in the previous section, i.e. we assumed the movement to be given by $x_{t+1} = x_t + v_t$ where $v_t = (1 - 10^{-\rho}) \cdot v_{t-1} + 10^{-\rho} \cdot u_t$. Figure 7 reports the distance traveled toward the goal for different values of the filtering effect ρ . We can observe that in the previous version of the algorithm the distance traveled toward the goal decreases as ρ increases and becomes nearly 0 for a value of $\rho = 3$. The newer version has better performance for all the ρ settings, and interestingly the performance increases for $\rho > 2.4$. This is due to a smoothing effect introduced by the low pass filter that makes the trajectories more straightly headed to the goal.

Using ODE⁵, we finally compared the two algorithms with a simulated a mobile robot. The robot is equipped with three spherical wheels having diameter of 15 cm. The two front wheels are directly actuated by two independent motors whose maximum velocity is 0.5 rad/s while the rear wheel is free to rotate in any direction. The task is to reach a red hemisphere of radius 10 m placed at a

⁵ Open Dynamics Engine, a free library for simulating rigid body dynamics. For details see <http://www.ode.org>

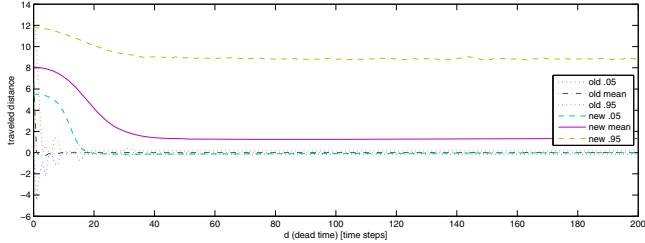


Fig. 6. Average movement toward the goal for different dead time values. The plot reports the average distances traveled using the two algorithms, as well as the 0.05 and 0.95 quantiles, i.e. the distances traveled toward the goal are reported with their 90% confidence interval. The graphs were obtained setting the maximum velocity $s = 10^{-3}$, placing the robot at $x_0 = [100, 100]^T$, simulating the movement for $N = 10^4$ steps and repeating the experiment 10^4 times.

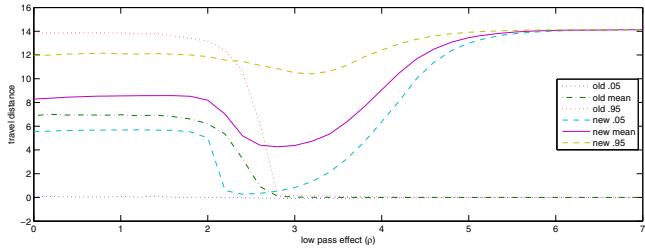


Fig. 7. Average movement toward the goal for different values of the low pass filter entity ρ . The plot reports the average distances traveled using the two algorithms, as well as the quantile function for 0.05 and 0.95, i.e. the distances traveled toward the goal are reported with their 90% confidence interval. The graphs were obtained setting the maximum velocity $s = 10^{-3}$, placing the robot at $x_0 = [100, 100]^T$, simulating the movement for $N = 10^4$ steps and repeating the experiment 10^4 times.

distance of 30m. Sensory information comes from a simulated omni-directional camera and the value of ΔA_t is determined observing the change in the number of red pixels in the image. In particular, if the R component of a pixel is more than double the maximum of the G and B components, then the pixel is considered red.

The robot was simulated in five different conditions, in the normal condition and with four types of damages, as done in [4]:

1. one wheel size is reduced to two thirds of its normal size
2. one wheel becomes uncontrollable, i.e. its movement is completely random
3. one wheel rotation axis direction is turned 90 degrees along the Z axis and becomes parallel to the longitudinal axis, i.e. the rotation of the wheel instead pushing the robot forward and backward pushes the robot sideways
4. 20% of the camera image becomes obscured

As a result, the newer algorithm provides faster reaching times than the previous version (with α/β set as to maximize the performances) in most of the cases. Better performances are obtained by the previous version in the uncontrollable tyre case, because with that setup automatically setting the noise level becomes difficult. For further information see <http://robotics.dei.unipd.it/fabiodl/papers/material/simpar10/>.

5 Conclusions and Future Works

In this paper, we presented a simple and very general control algorithm than can be applied to a wide variety of robots without any knowledge on the hardware structure. The robustness of the algorithm was tested by the simulation of extremely nonlinear system and systems that include large delays. We notice that the main focus here was in proposing a very general and simple control algorithm, and that for specific problems, certainly there are more dedicated and better performing algorithms. In particular the algorithm does not store any information about the world except whether the robot conditions improved or not over the previous time step. This can reveal advantageous when the world conditions changes so often that storing information would be unworthy or when sensory information, computation capabilities and the memory available are very limited. In fact, we must not forget that this algorithm is inspired from the movement of very primitive organisms like bacteria. If modeling the world dynamics is expected to be beneficial and richer sensory information and resources are available then it would be possible to include reinforcement learning to improve the performances of the presented algorithm. In detail, a straightforward approach would be to use the behavioral rule presented in this paper as the Actor of the classical Actor-Critic architecture [1]. In this setup the TD error provided by the critic would constitute the ΔA_t of our algorithm. Furthermore the actor could use the signals from the critic to modify its policy by altering the probability used to choose a new action when ΔA_t is negative. Expressly, the probability for a new action would be a function of the state, and learning would try to maximize the probability of choosing optimal actions for each state. Using our algorithm as an actor would likely provide the system with an efficient bootstrap, and would balance in a simple way exploitation and exploration.

Another very interesting point that will be analyzed more deeply in future works is the mechanism underlying the performance improvement due to the introduction of random perturbations of opportune magnitude. The performance visible in log scale in Fig. 1 and in linear scale in Fig. 4 clearly resembles the Signal to Noise Ratio curves arising in presence of stochastic resonance. Briefly, stochastic resonance is a phenomenon for which adding noise improves the sensitivity of sensors in nonlinear sensing systems. In recent years, many biological researches have focused on stochastic resonance to explain the high robustness and sensitivity of sensory organs of living beings. Analyzing how a stochastic resonance effect emerges in a very simple algorithm like the one presented here will surely be interesting from an engineering point of view. Furthermore, while

most of the biological researches on stochastic resonance focus on the sensing mechanisms of creatures and how they exploit stochastic resonance, our results focus on an improvement of the performances of a control algorithm. As a final result, we hope that clarifying the mechanism underlying the algorithm could help understanding the control mechanism of living creatures.

References

1. Adler, J.: The sensing of chemicals by bacteria. *Scientific American* 234, 40–47 (1976)
2. Baker, M.D., Wolanin, P.M., Stock, J.B.: Systems biology of bacterial chemotaxis. *Current Opinion in Microbiology* 9(2), 187–192 (2006); *cell Regulation / Edited by Werner Goebel and Stephen Lory*
3. Benzi, R., Sutera, A., Vulpiani, A.: The mechanism of stochastic resonance. *Journal of Physics A: mathematical and general* 14, 453–457 (1981)
4. DallaLibera, F., Ikemoto, S., Minato, T., Ishiguro, H., Menegatti, E., Pagello, E.: Biologically inspired mobile robot control robust to hardware failures and sensor noise. In: Robocup 2010, Singapore (2010)
5. Dhariwal, A., Sukhatme, G.S., Requicha, A.A.G.: Bacterium-inspired robots for environmental monitoring. In: 2004 IEEE International Conference on Robotics and Automation (ICRA 2004), New Orleans, USA, pp. 1436–1443 (2004)
6. Gammaiton, L., Hägggi, P., Jung, P., Marchesoni, F.: Stochastic resonance. *Reviews of Modern Physics* 70(1), 223–287 (1998)
7. Jiang, L., Ouyang, Q., Tu, Y.: Quantitative modeling of escherichia coli chemotactic motion in environments varying in space and time. *PLoS Comput. Biol.* 6(4), e1000735 (2010)
8. Rao, C.V., Kirby, J.R., Arkin, A.P.: Design and diversity in bacterial chemotaxis: A comparative study in escherichia coli and bacillus subtilis. *PLoS Biol.* 2(2), e49 (2004)
9. Schnitzer, M.J.: Theory of continuum random walks and application to chemotaxis. *Phys. Rev. E* 48(4), 2553–2568 (1993)
10. Segall, J.E., Block, S.M., Berg, H.C.: Temporal comparisons in bacterial chemotaxis. *Proceedings of the National Academy of Sciences of the United States of America* 83(23), 8987–8991 (1986)
11. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning). The MIT Press, Cambridge (March 1998)
12. Wiesenfeld, K., Moss, F.: Stochastic resonance and the benefits of noise: from ice ages to crayfish and SQUIDs. *Nature* 373(6509), 33–36 (1995)

Exploration Strategies for a Robot with a Continuously Rotating 3D Scanner

Elena Digor, Andreas Birk, and Andreas Nüchter

Jacobs University Bremen gGmbH,

School of Engineering and Science,

Campus Ring 1, 28759 Bremen, Germany

{e.digor,a.birk,a.nuechter}@jacobs-university.de

Abstract. To benchmark the efficiency of exploration strategies one has to use robot simulators. In an exploration task, the robot faces an unknown environment. Of course one could test the algorithm in different real-world scenarios, but a competitive strategy must have good performance in *any* environment that can be systematically constructed inside a simulator. This paper presents an evaluation of exploration strategies we developed for a specific sensor. A continuously rotating 3D laser scanner that scans only into one direction at a time moves through the environment sampling the surrounding. Our evaluation framework features an efficient scanning and robot simulator for kinematic feasible trajectories. We will show that shorter trajectories do not necessarily imply quicker exploration. A simple simulator framework is sufficient for evaluating these properties of path planning algorithms.

1 Introduction

In the last decade, the path planning problems of autonomous mobile robots have received a lot of attention in the communities of robotics, computational geometry, and on-line algorithms. Online exploration is still a crucial issue in mobile robotics. Given an unknown environment, a robot has to find a tour, from which it can eventually see the whole environment. Autonomous robots that can solve the simultaneous localization and mapping (SLAM) problem, need an exploration strategy to operate with true autonomy [10,16]. However, most of the related problems in computational geometry are NP-hard and it is challenging to program good strategies in a robot control architecture. For example, using a point-like mobile robot model in a polygonal environment and a vision system that is able to *see* with a 360 degree field of view with infinity range, Hoffmann et al. have presented a strategy yielding a competitive factor of 26.5 [6]. The competitive ratio is the cost of exploration in relation to the optimal exploration with full information. This implies that in the worst case the path of the robot is at most 26.5 longer than the optimal path. Other related theoretical grounded strategies consider the problem of how to look around the corner [3,4] or optimal search [5].

Despite these impressive theoretical results, real robots face practical problems. Reliable sensors systems are still in the development, real-world

environments are not (simple) polygons, and often a decent approximative solution is already considered a good robot behavior.

This paper addresses the specific problem of finding a good exploration strategy for a mobile robot with a continuously rotating 3D scanner. Fig. 1 shows such a robot. The main sensor is a RIEGL VZ-400 laser scanner [11] that continuously rotates around the vertical axis and is therefore able to acquire 3D scans while moving. In earlier work, we constructed the mobile robot Kurt3D with a nodding 3D scanner with the stop-scan-plan-go operation mode [14] and a Kurt3D version with a continuously rotating SICK scanner [1]. These robots are regularly used as rescue robots, e.g., in RoboCup rescue, as inspection robots, e.g., for mapping abandoned underground mines, or as surveying system, e.g., for factory design, facility management urban and regional planning. In this paper we develop a simulation-based evaluation framework that allows us to quickly benchmark different strategies while considering kinematic motion constraints.



Fig. 1. The mobile robot Irma3D (Intelligent robot for mapping applications in 3D) with its main sensor, the RIEGL VZ-400

2 Related Work

Most known approaches use a stop-scan-plan-go method. This problem is an extension of Art Gallery problem [15]: Determine how many guards are sufficient to see every point in the interior of an n -wall art gallery room. The room is assumed to be represented by a simple polygon, i.e., the room can be fully covered by one boundary, which does not cross over itself. The position of the guards can be seen as the points in which the autonomous robot should stop and scan the environment. Lee and Lin [12] proved that the solution to the original Art Gallery problem is NP-hard. Surprisingly, the Watchman problem, i.e., a single mobile guard that moves through the gallery and has to completely see it, can be solved in polynomial time. Unfortunately, robotic exploration is harder.

The robot does not know the environment beforehand, so it cannot predict the shape of the environment, and hence any paths it plans cannot be assured to be optimal. Its laser range scanner covers a finite area, in discrete angles and with a maximal distance.

A brute force solution to exploring the whole region is to follow the walls and to avoid obstacles. Yamauchi et al. [18] proposed an improved solution for 2D maps, by suggesting to approach the frontiers, i.e. the border between unknown and known regions. The map is usually saved in an evidence grid format, that is a 2 dimensional regular grid, in which a cell contains evidence or probability, based on accumulated sensor readings, that the particular patch of space is occupied. Frontiers are considered to be the cells for which the occupancy probability is equal to a prior probability. The path planning is, then, based on depth first search for finding the frontier regions and for avoiding the obstacles. The robot moves next to a frontier region to take the next scan and to update the evidence grid. We say the environment has been fully explored when the evidence grid has no more frontier regions which are big enough for the robot to go through.

A very close solution to Yamauchi, but in 3D this time, is given by Nagatani et al. [13]. Since keeping 3D point clouds is very expensive, the data is preprocessed and it is saved in Multi Level Surface (MLS) map. An MLS map is a 2D grid map, where each cell keeps information about height of objects seen there.

Recently, Holz et al. have evaluated [7] exploration strategies. According to their findings, frontier based approaches are sufficiently good in real application scenarios.

3 Exploration Strategies

Our exploration strategies are based on frontiers in a 2D grid representation of the environment and extend the flood fill algorithm. Flood fill is often used in image processing applications. The original algorithm colors recursively all the pixels, that neighbor a starting pixel, and that have the same color as it. A neighbor pixel is considered to be the pixel which is up/down/left or right next to the current pixel. An extension of the algorithm's neighbor definitions, allows us to use it in finding a shortest path to a desired point on the map, in our case, to a frontier point. Since our robot is also allowed to go diagonally at a higher cost, we integrate that part in our exploration model.

The classical flood fill algorithm uses depth first search to find and color the neighbors. This is unsuitable for path planning and thus the more efficient Breadth First Filling (BFF) is used. We calculate the radius, i.e., the length of the path from the starting point to the current point, at every step and enqueue the encountered points in a priority queue, instead of an ordinary one, yielding a Breadth First Filling with Priority Queue (BFFPQ) algorithm. The priority queue allows us to sort the points based on their total distance value. Since the map resolution is 1 grid cell = 1 cm, a diagonal move is taken to be equal to $\sqrt{2} \approx 1.44$ cm at sub grid cell accuracy. For every grid cell we keep record of the shortest distance from the source (cf. Fig. 2). We implemented the priority queue

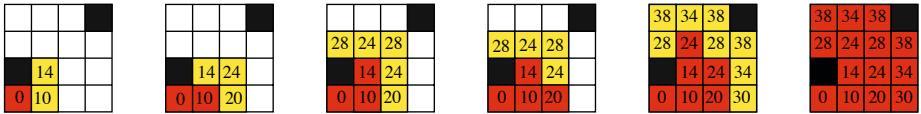


Fig. 2. Breadth first filling with priority queue. The grid cells that have been processed are shown in red, the cells in the queue are shown in yellow and the distances (in mm) from the source to the corresponding cell are given in digits.

using a heap. The sort is according to the distance of the inserted points from the source. Hence, an insert and removal from the priority queue has complexity $O(\log N)$ in worst case scenario. In practice this algorithm runs faster, due to the fact that the sorting operations are on top of the list.

The BFFPQ algorithm guarantees to find out the closest frontier. However, we are also interested in the actual path that the robot has to follow to reach that frontier. For finding it we run a backward search algorithm that calculates the path from the frontier point, step by step back to the source.

Next, we describe four strategies to explore the whole environment. Their basis is BFFPQ for automatic exploration missions.

3.1 Stop-Scan-Replanning-Go Strategy

The Stop-scan-replanning-go strategy starts with a 360° scan, and then runs BFFPQ and finds the first frontier grid cell. After finding the path to this cell, the robot follows it to reach the goal.

Once it reaches the goal, it stops, and does a 360° scan. Next, as the name of the algorithm suggests, the robot re-plans its new goal by finding the closest frontier and the path towards it. The algorithm continues this way until the robot has no more reachable frontiers.

In this strategy the 3D scanner is constantly rotating, i.e., also during the path following.

3.2 Scan-Replanning-Go Strategy

A second algorithm is a small deviation from the previous one. Instead of doing constant stops for a full scan of the environment, we instantly compute a new frontier cell to reach. The incentive is to save the time required for stopping and scanning in favor of a shorter overall total time for an entire map exploration.

3.3 Continuously-Replanning-with-Stopping Strategy

The following strategy takes full advantage of the fact that while moving the robot also scans the environment, which in most of the cases might result in opening the goal-frontier, before actually reaching it. Hence, we implemented an algorithm according to which, once the robot opens the frontier point while following a path towards it, the robot instantly searches for another goal point and changes its path towards the new one.

Recall that whenever the robot aims to reach a frontier, it tries to reach the closest one. Hence, if the robot opens it before actually reaching it, there is a very small chance that a full 360° scan would actually lead to new discoveries on the map. Hence, we do not stop the robot, but make the robot go instantly to the next closest frontier. Reaching the goal point however, implies that the rotating scanner failed to explore the frontier pixel while being on the way towards it. As a result, it seems that most probably the robot is located in a quiet unexplored area, and it would be advantageous for it to stop and do a full 360° scan.

3.4 Continuously-Replanning Strategy

The Continuously-replanning strategy is a modification of the latter one, with the constraint that the robot never stops. Similarly to the scan-replanning-go strategy we aim at saving overall exploration time.

The robot starts with a 360° scan, and afterwards it finds the closest frontier to go to. It follows the path towards it, and if while moving towards the goal the robot explores the frontier, it sets a new goal and a new path to follow. Again, this exploration is complete when there are no more frontiers on the map that the robot can reach.

4 Scanner and Robot Simulation

We set up a basic simulation framework to simulate the constantly rotating scanner and the mobile robot.

4.1 The Scanner

Scanning is an essential part of our exploration missions. However, since simulator is currently in 2D we have to simulate 72 scans per second to yield a scan resolution of 1° . Hence a full 360° scan takes 5 seconds, which corresponds to our used hardware the Riegl VZ-400. By having the length of a beam, we can easily calculate a far-most point (in our coordinate system) for each scan line. Hence, at every time step, the simulator has to mark all grid cells starting with the current robot position (x, y) and ending either with far-most point or at the closest encountered obstacle. We use the Bresenham algorithm to quickly simulate the beams [2]. Beam divergence does not have to be considered, since it is negligible for the VZ-400. However, the used Bresenham algorithm has to be altered, if other laser scanners are simulated. For example the laser spot projected by the SICK LMS-200 is already at a distances of 5 meter roughly 4 cm and therefore considerable high.

Frontier computation. Since our exploration algorithms are all based on closest frontier method [18], at any point in time the exploration algorithm has to record the current frontiers. A frontier is defined as all unexplored grid cells which directly border with already explored free cells. The frontier pixels are marked on the robot's generated map. Hence, the frontier regions are continuously marked with every beam that the laser scanner emits (cf. Fig. 3).

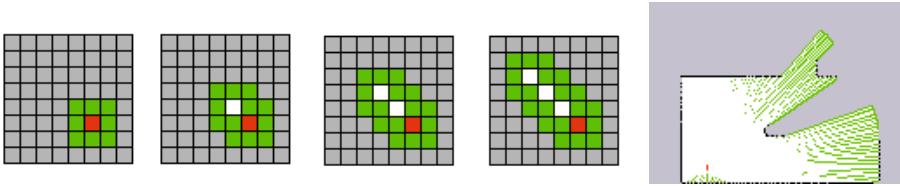


Fig. 3. Left: A single laser line (white cells) generation via Bresenham algorithm, with frontier markings (green cells) at every construction step (from left to right). Right: A simulated scan in a room. Due to the rotational resolution not all walls are completely sampled.

4.2 The Robot Platform

The robot is the main component being simulated. Irma3D is a differential drive robot. Besides a manual control, we simulate the kinematics of the differential drive vehicle on the set velocities v_l and v_r for the left and right wheel on a common axis at a distance $2b$ one from the other. Let $u \mathbf{e}_1$ and $\omega \mathbf{e}_2$ be the linear and angular velocities of the mid point of such axis being \mathbf{e}_1 and \mathbf{e}_2 body fixed unit vectors as depicted in Fig. 4. Let $\mathbf{e}_1 v_r$ and $v_l \mathbf{e}_1$ be the velocities of the center of the right and left wheels that are assumed to roll perfectly. The kinematic model linking the scalars u , ω , v_r and v_l is:

$$u = \frac{1}{2} (v_r + v_l) \quad \omega = \frac{1}{2b} (v_r - v_l) \quad (1)$$

where $|v_r|$ and $|v_l|$ will be bounded by some given value V_m . As known, a differential drive vehicle modeled by equations (1) can move on paths of arbitrary curvature κ , as

$$\kappa = \frac{1}{b} \frac{v_r - v_l}{v_r + v_l}. \quad (2)$$

Indeed for any $v_l = -v_r$ the corresponding path curvature would be infinite, i.e. the vehicle would turn on the spot. Arbitrarily large curvature values can be implemented if the wheels rotate in opposite directions. Yet if the vehicle should be required to move nicely and possibly at high linear speeds, commanding wheel speeds of different sign in order to make sharp turns should be avoided in practice in order not to overstress the electromechanical structures of the robot, e.g. gear boxes, tires, DC motor H-Bridge power circuits. If the wheel speeds v_l and v_r should be constrained to positive values only, than the curvature κ given by equation (2) would be bounded

$$\kappa \in \left[-\frac{1}{b}, \frac{1}{b} \right] \text{ if } v_l, v_r \in [0, V_m] \quad (3)$$

and the linear speed $u > 0$ should be constrained to $u \leq V_m/2$ to let κ span its full range $[-1/b, 1/b]$ [8].

Our simulation environment considers the path following controller designed in [17]: given a Serret-Frenet frame $\{F\}$ moving along the planar path, call P

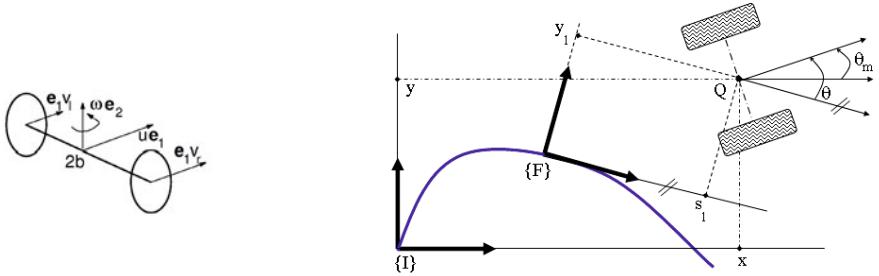


Fig. 4. Left: Differential drive. Right: The used Frenet frame

its origin having curvilinear abscissa s with respect to an arbitrary path point (origin of the curvilinear abscissa). Call $\{I\}$ a fixed inertial frame and Q the mid point of the differential drive robot axis such that Q has coordinates (s_1, y_1) in $\{F\}$ and (x, y) in $\{I\}$ (cf. Fig. 4, right). Calling κ_r the curvature of the reference path, the kinematics of Q in $\{I\}$ would be given by the unicycle model

$$\dot{x} = u \cos \theta_m \quad \dot{y} = u \sin \theta_m \quad \dot{\theta}_m = \omega \quad (4)$$

and by

$$\dot{s}_1 = -\dot{s}(1 - \kappa_r y_1) + u \cos \theta \quad (5)$$

$$\dot{y}_1 = -\kappa_r \dot{s} s_1 + u \sin \theta \quad (6)$$

$$\dot{\theta} = \omega - \kappa_r \dot{s} \quad (7)$$

in $\{F\}$ (refer to [17] for details) being θ_m and θ the vehicle's heading in $\{I\}$ and $\{F\}$ respectively. Following [17] we consider the Lyapunov candidate function

$$V_1 = \frac{1}{2} (s_1^2 + y_1^2) + \frac{1}{2\gamma} (\theta - \delta(y_1, u))^2 \quad (8)$$

for some positive γ to derive the closed loop, time invariant and globally stable control law:

$$\dot{\theta} = \dot{\delta} - \gamma y_1 u \frac{\sin \theta - \sin \delta}{\theta - \delta} - k_2 (\theta - \delta) \quad (9)$$

$$\dot{s} = u \cos \theta + k_1 s_1 \quad \text{with } k_1 > 0, k_2 > 0 \quad (10)$$

The above control law yields the commanded wheel velocities as

$$v_r = u + b (\kappa_r \dot{s} + \dot{\theta}) \quad v_l = u - b (\kappa_r \dot{s} + \dot{\theta}) \quad (11)$$

in which the maximal set values can be incorporated [9], preserving the property of a closed loop, time invariant and globally stable controller.

5 Experiments and Results

The performance of our developed algorithms was analyzed on our testing bench, the simulator. The testing was based on 5 different maps, which are presented in Fig. 5. For each of the five maps we created 10 different random starting states for the robot. All four algorithms were applied to each testing case.

Fig. 6 shows the resulting trajectories of 4 algorithms on the simple map. We notice that in all the cases where the exploration algorithm implied no full stop for a 360° scan, i.e. Scan-replanning-go strategy, and Continuously-replanning-go strategy, the path comes out to be longer and jittery. In Fig. 6 (second), one can see that the robot, even has a tendency to go in a circular trajectory. This behavior is due to the fact that the robot does not stop, and it's doing planning and scanning while moving. Hence, it might be the case, that while reaching a goal point on the map, the environment around is still barely explored. A full scan is done in 5 seconds, which means every time step the robot does a move, it covers only 72° of the environment. Similar results have been obtained in the other maps, see Fig. 7 and Fig. 8.

Exploration path lengths for all strategies are given in Fig. 9 and the corresponding exploration time in Fig. 10. Please notice the scaling of the y -axis. An additional observation is that once the office map contains clutter, i.e., obstacles, the total path length increases significantly for the non-stopping algorithms. The reason is that the obstacles, being of different geometrical forms, block the laser beams from reaching the entire room, and as a consequence the robot has to wander around more, before it explores one entire room.

The large scale maps (Fig. 5 bottom) emphasize our observation that in all the cases where the exploration algorithm implies no full stop for a 360° scan, i.e. Scan-replanning-go strategy, and Continuously-replanning-go strategy, the path

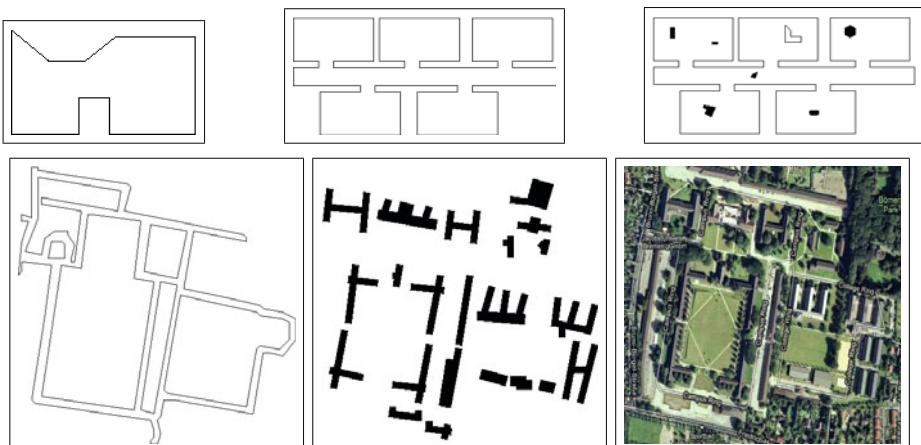


Fig. 5. Five maps used for evaluation: Simple map, empty office map, office map with obstacles, campus roads and campus buildings. Bottom right most map is a satellite view of the campus

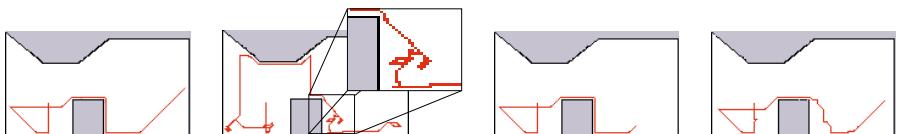


Fig. 6. Results in the simple map. From left to right: Stop-scan-replanning-go, Scan-replanning-go, Continuously-replanning-with-stopping, and Continuously-replanning-go strategy.

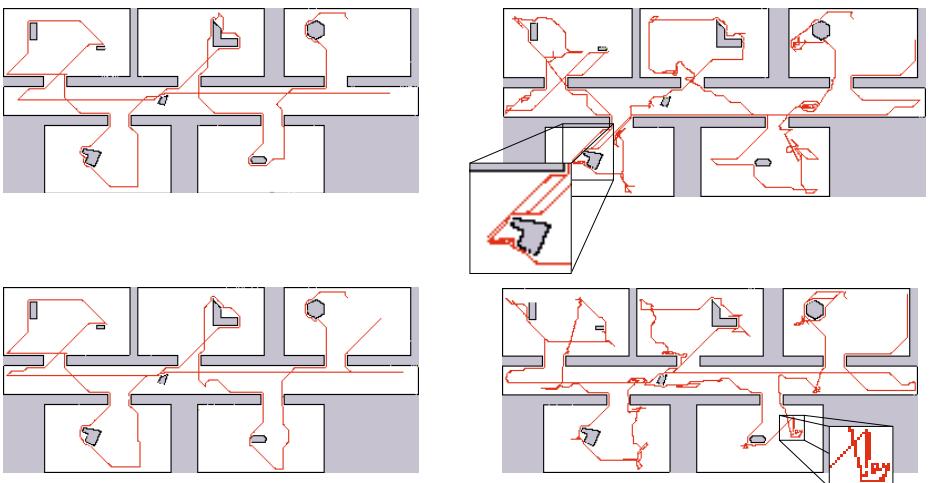


Fig. 7. Results in the office map with clutter. From left to right: Stop-scan-replanning-go, Scan-replanning-go, Continuously-replanning-with-stopping, and Continuously-replanning-go strategy.

comes out to be very long and jittery. On the other hand, the overall time for the exploration was much shorter than for the other two algorithms. Moreover Stop-scan-replanning-go and Continuously-replanning with stopping strategies seem to do similarly good in total path length, but the latter one seems to have an advantage in the total time spent for the entire exploration.

Furthermore, for the large maps, we see a considerable gap between the total path length of the strategies with stopping compared to the ones without. Since both of the maps appear relatively uniform, and very big, statistically speaking, there are no more favored starting positions. Any position the robot will start from will cover a very small portion of the map, and every stop cleans up a small circle area. Hence the robot will more or less go through many small areas,

A video of the exploration strategies can be found under the following links:
<http://plum.eecs.jacobs-university.de/download/simpar2010.mpg> and
<http://plum.eecs.jacobs-university.de/download/simpar2010-cmp.mpg>

An additional finding is that the paths, that have been computed by the planner, were most of the time feasible by the closed loop, time invariant and

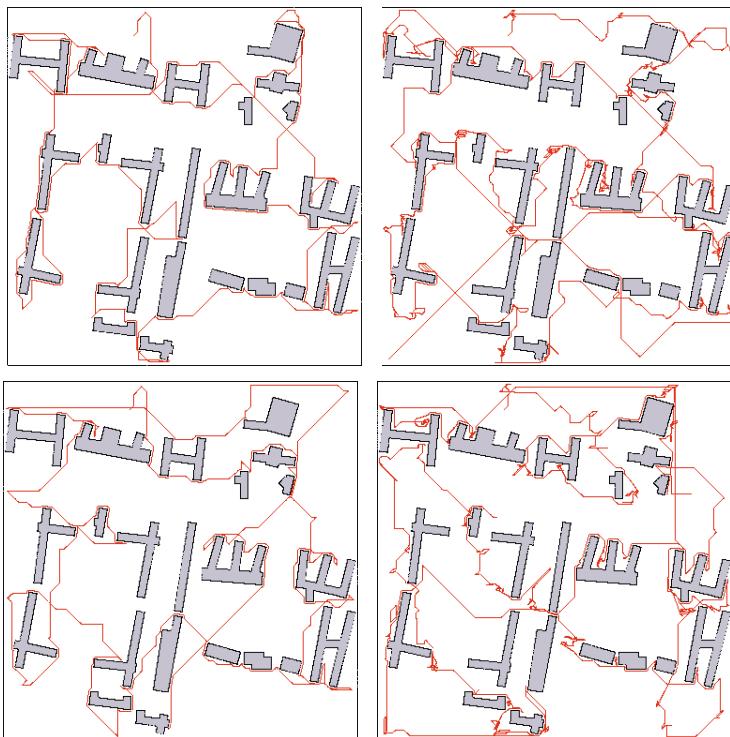


Fig. 8. Results in the large outdoor map. From left to right: Stop-scan-replanning-go, Scan-replanning-go, Continuously-replanning-with-stopping, and Continuously-replanning-go strategy.

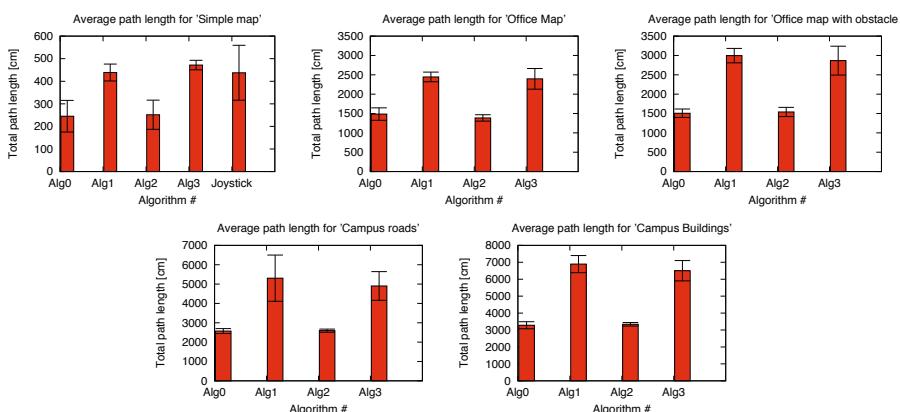


Fig. 9. Average and deviation of the path lengths for the four exploration strategies in our five maps. From top left to bottom right: Simple map, empty office map, office map with obstacles, campus roads and campus buildings. The visualization for the simple map contains in addition the path length for exploration by an operator.

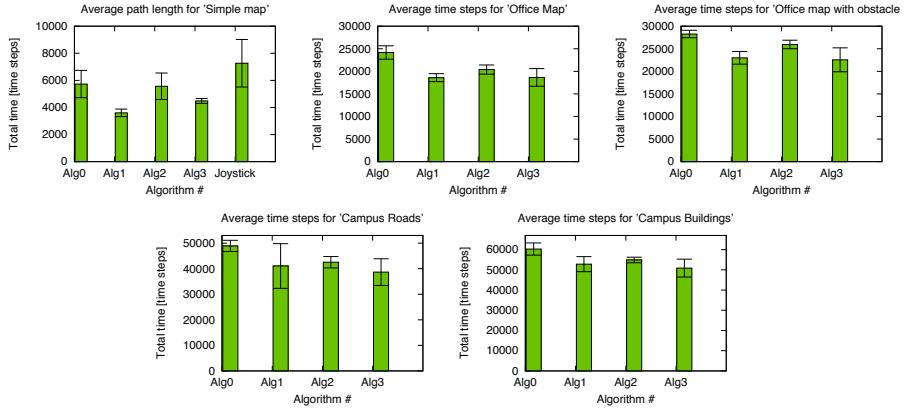


Fig. 10. Average and deviation of exploration times for the four strategies in our five maps. From top left to bottom right: Simple map, empty office map, office map with obstacles, campus roads and campus buildings. The visualization for the simple maps contains in addition the time needed for exploration by an operator.

globally stable control law with bounded wheel velocities. However, sometimes the robot control has to revert to a much simpler path following strategy, where the robot is allowed to turn on the spot.

6 Conclusion and Future Work

Exploration is an ongoing research area in robotics. The emerging technology of 3D sensing devices results in additional challenges exploration algorithms must handle. Most of the literature cover the theoretical analysis of different *stop and scan* methods with a scanner that has a fixed field of view and which is attached on the top of the robot. Our aim was to take usage of a constantly rotating 3D scanner, and to consider its advantages by using *scanning on the fly* methods.

We developed and tested in simulation four different exploration strategies which are based on the frontier approach combined with an extension of flood fill algorithm. Two of the algorithms involve stopping at frontier points to take full 360° scans of the environment, and the other two implied constant movement until the entire map is covered.

In order to test the soundness and consistency of our exploration methods, we have implemented our own small simulator. The robot is simulated by a differential drive model, with a constantly rotating scanner attached to it. The scanner does a full 360° scan of the environment in a certain time, and updates the map accordingly.

Furthermore, the proposed small simulator enables us to test kinematic robot control laws, such as the presented de Wit's control. This enables us to quickly determine control parameters, without using real hardware. This paper has shown that for certain problems there is no need for a sophisticated simulation

environment. A home-brew small simulator enables the roboticist to study important issues of a real robotic system.

In future work, we will test the proposed exploration strategies in combination with the closed-loop path following on the real robot Irma3D. All application scenarios that require 3D robotic mapping benefit from the proposed study. In addition, we'll work on the theoretical foundations of the exploration method and aim at finding the competitive ratio of exploration with a continuously rotating sensor.

References

1. Brenneke, C., Wulf, O., Wagner, B.A.: Using 3D laser range data for SLAM in outdoor environments. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003), Las Vegas, USA (October 2003)
2. Bresenham, J.E.: Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1), 25–30 (1965)
3. Fekete, S.P., Klein, R., Nüchter, A.: Online searching with an autonomous robot. *Computational Geometry: Theory and Applications (CGTA)* 34(2), 102–115 (2006)
4. Fekete, S.P., Schmidt, C.: Polygon exploration with time-discrete vision. *Computational Geometry: Theory and Applications (CGTA)* 43, 148–168 (2010)
5. Fleischer, R., Kamphans, T., Klein, R., Langetepe, E., Trippen, G.: Competitive online approximation of the optimal search ratio. In: Albers, S., Radzik, T. (eds.) *ESA 2004*. LNCS, vol. 3221, pp. 335–346. Springer, Heidelberg (2004)
6. Hoffmann, F., Icking, C., Klein, R., Kriegel, K.: The polygon exploration problem. *SIAM Journal on Computing (SICOMP)* 31, 577–600 (2001)
7. Holz, D., Basilico, N., Amigoni, F., Behnke, S.: Evaluating the efficiency of frontier-based exploration strategies. In: Proceedings of the Joint Conference of the 41st International Symposium on Robotics (ISR 2010) and the 6th German Conference on Robotics (ROBOTIK 2010), Munich, Germany (2010)
8. Indiveri, G., Corradini, M.L.: Switching linear path following for bounded curvature car-like vehicles. In: Proceedings of the IFAC Symposium on Intelligent Autonomous Vehicles (IAV 2004), Lisbon, Portugal (July 2004)
9. Indiveri, G., Nüchter, A., Lingemann, K.: High speed differential drive mobile robot path following control with bounded wheel speed commands. In: Proceedings of the IEEE International Conference Robotics and Automation (ICRA 2007), Rome, Italy (April 2007)
10. König, S., Tovey, C., Halliburton, W.: Greedy mapping of terrain. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2001), Seoul, Korea, pp. 3594–3599 (May 2001)
11. Riegl Laserscanner (2010), <http://www.riegl.co.at/>
12. Lee, D., Lin, A.: Computational complexity of art gallery problems. *IEEE Transactions on Information Theory* 32(2), 276–282 (1986)
13. Nagatani, K., Matsuzawa, T., Yoshida, K.: Scan-point planning and 3-d map building for a 3-d laser range scanner in an outdoor environment. In: Field and Service Robotics (FSR 2009). Springer Tracts in Advanced Robotics, vol. 62, pp. 207–217 (2010)
14. Nüchter, A., Surmann, H., Hertzberg, J.: Planning robot motion for 3D digitalization of indoor environments. In: Proceedings of the 11th International Conference on Advanced Robotics (ICAR 2003), Coimbra, Portugal, pp. 222–227 (June 2003)

15. O'Rourke, J.: Art Gallery Theorems and Algorithms. Oxford University Press, New York (1987)
16. Pierce, D., Kuipers, B.: Learning to explore and build maps. In: Proceedings of the twelfth national Conference on Artificial Intelligence (AAAI 1994), Seattle, W.A., U.S.A. American Association for Artificial Intelligence, pp. 1264–1271 (1994)
17. Soetanto, D., Lapierre, L., Pascoal, A.: Adaptive, Non-Singular Path-Following Control of Dynamic Wheeled Robots. In: Proceedings of the 42nd IEEE Conference on Decision and Control (CDC 2003), Maui, Hawaii U.S.A. pp. 1765–1770 (December 2003)
18. Yamauchi, B.: A frontier-based approach for autonomous exploration. In: Proceedings of Computational Intelligence in Robotics and Automation, CIRA 1997, pp. 146–151 (July 1997)

Validating an Active Stereo System Using USARSim

Sebastian Drews, Sven Lange, and Peter Protzel

Chemnitz University of Technology, 09126 Chemnitz, Germany
`{sebastian.drews,sven.lange,peter.protzel}@etit.tu-chemnitz.de`

Abstract. We present our ongoing work on autonomous quadrotor UAV navigation using an active vision sensor. We motivate the usage of the active vision sensor instead of standard laser range finders and present the underlying technical and theoretical mechanisms. Initial results on ICP based SLAM obtained with a prototype of this sensor, give rise to a deeper analysis of the active vision sensor in a simulation environment. Therefore, the sensor model is integrated in USARSim. Adaption of the sensor model and results on the active vision SLAM algorithm in the idealized environment are discussed and consequences for the real world application are inferred.

1 Introduction

Our research focuses on enabling autonomous, mobile systems to be applicable in a variety of civil applications, mainly in the areas of emergency response, disaster control and environmental monitoring. These scenarios require a high level of autonomy, reliability and general robustness from every robotic system, regardless of whether it operates on the ground or in the air.

While the main research interests over the past years were focused on mobile ground vehicles, recently multirotor unmanned aerial vehicles (UAVs) became more popular. They are now available in a variety of forms and sizes and can be purchased at affordable prices. The systems can be divided into commercially available systems and open source based UAVs which are often based on the *Mikroopter* [3] design. Among the commercial systems those from AscTec are widely used for research purposes, for instance the *Hummingbird* [6]. In contrast to ground vehicles these UAVs offer greater agility and are independent of terrain surface quality. Due to their small size they can also be used indoors, which constitutes a benefit regarding rescue scenarios.

While UAVs work well in environments with available GPS measurements, autonomous navigation without GPS remains a challenge. This is why our research interests are focused on reaching the desired level of autonomy in scenarios where GPS signals are not available. We already developed a method for autonomous landing of our quadrotor system (Fig. 1) [1] and are now working on a robust navigation within cluttered indoor environments. We believe this can be done by using a so-called active stereo system utilizing structured light. Such a system

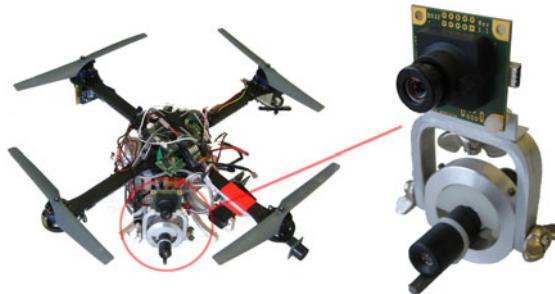


Fig. 1. The image shows our quadrotor system and the active stereo vision prototype

can make use of the already available camera aboard the UAV and therefore does not take up much more of the valuable payload.

Our goal is to construct a reliable map of the environment including a simplified 3D view detailed enough for a human operator to work with, while using low-cost and lightweight components.

In order to support our work we wanted to simulate an active stereo system with the aid of a versatile simulation environment as this would put us in a position to evaluate our approach without the influence of any deficiencies of the underlying hardware components. Encouraged by promising experiences we already made regarding the development of image processing algorithms in combination with the Unified System for Automation and Robot Simulation (USARSim) [4], we decided to choose this simulation environment for our purpose.

2 Related Work

Several research groups are working on UAVs with focus on indoor navigation using different kinds of sensor configurations of which a compact laser scanner is the most similar approach compared to our efforts.

The authors of [5] for example use a Hokuyo URG laser range finder (LRF) aboard a self made quadrotor UAV. The focus of their work was to transfer localization and mapping capabilities of ground vehicles to the UAV while using identical algorithms. With respect to the additional degree of freedom in translation, they use a mirror to deflect some of the laser rays towards the ground. A similar sensor configuration is used in [7] for path planning in information space. The objective is to find a path with good sensor measurements for robust localization. In contrast to [5] their UAV is a commercially produced *Hummingbird* quadrotor.

While the mentioned authors present working prototypes, one major drawback remains. Using the Hokuyo LRF results in a significant consumption of payload which constrains the possibilities to extend the system by additional sensors or computing power. Furthermore, the system's possible operation time will decrease. The authors of [1] compensate this problem by building a more

powerful UAV. Their quadrotor is specially designed for carrying a stereo camera, a Hokuyo LRF with a range of $30m$ and a $1GHz$ Intel PC. However, the high payload of about $500g$ is traded against a shorter flight time of about 10 minutes.

We believe that the use of an LRF is not feasible because it requires too much payload. In our opinion the issue can be solved by using an active stereo system. A combination of a camera and a laser line projection would give information similar to the Hokuyo LRF by only integrating a lightweight laser into the system. The needed camera is often already onboard the UAV and can be used for both, measurement acquisition and live view.

The idea of using a laser line projection combined with a camera to measure distances is also used in [9] and [8]. The authors wanted to use their active stereo camera to aid people who suffer visual impairments and for obstacle detection in front of a driving car.

3 Active Stereo System

In general we use the term *active stereo system* to describe a non-contact 3D measurement system which at least consists of one light projector and a camera. Widespread applications for active stereo systems are industrial 3D measurement or reconstruction tasks. A short summary of various systems with references to additional literature is given in [16]. Besides a short description of different light patterns, a variety of methods for calibration of such systems is given.

Active stereo systems can be most beneficially applied in indoor scenarios. Specifically, there is no need to find correspondences between two pictures like in passive stereo systems which is hard to achieve in rooms with monochromatic or low-textured walls. Moreover, the active stereo system is less dependent on well illuminated environments, which is advantageous because insufficient illumination could lead to blurred images in passive stereo systems due to the UAV's motion. In addition, the active stereo system will also work in complete darkness.

3.1 Theory of Operation

In the previous section, we saw that active stereo systems consist of a camera and a light source for generating a pattern. Analogous to conventional stereo cameras, both components will be arranged in a certain distance to each other, called baseline. Alongside with camera resolution and focal length, it mainly influences the possible distance resolution and therefore the measurement range of the system. Distance measurements can be achieved by triangulation which will result in distance-dependent resolutions as shown in Fig. 2. The diagram shows the relation between the disparity and the resulting range measurement of our prototype, based on the following equation:

$$u(z) = f \frac{B}{z} \quad (1)$$

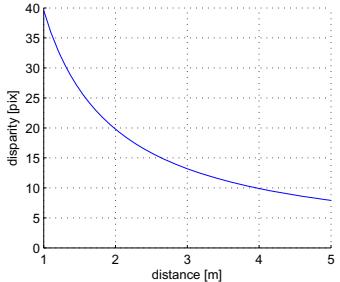


Fig. 2. Disparity depending on distance

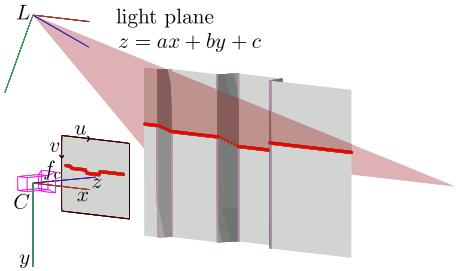


Fig. 3. Schematic view of an active stereo system with laser line projection

Here, u is the disparity in pixels, f the focal length in pixels, B the baseline in meters, and z the distance in meters.

For the special case of projecting a horizontal line of light, the line can be described as a plane of light located in the camera frame, as shown in Fig. 3. Depending on the extrinsic parameters of such an active stereo system, the projected laser line will translate along a specific direction within the image when changing the distance to an object. A reasonable configuration for our application would be a vertical arrangement of camera and laser line generator. By choosing a horizontal orientation of the laser line, the resulting distance measurements will be generated for each column of the image by finding the position of the line. Basis for the distance computation is the following equation [10]:

$$\mathbf{p}^C = \frac{-c}{(a(u - c_u) + b(v - c_v) - f)} \cdot \begin{pmatrix} u \\ v \\ f \end{pmatrix} \quad (2)$$

Where a, b and c are parameters of the light plane $z = ax + by + c$. The focal length is denoted f , u and v are the image coordinates, c_u and c_v label the image center coordinates and \mathbf{p}^C is the resulting 3D point within the camera frame C .

3.2 Prototype

After discussing theoretical aspects of active stereo systems, we will continue with the description of our assembled prototype. As an imaging component we used an IDS uEye color camera with a resolution of 752x480 and 60 degrees horizontal field of view (FOV). For the structured light projection we mounted a sleeve at 6cm distance for clamping in a laser pointer for line projection. The laser is a low-cost laser with 650nm wave length, 5mW output power and a cylindrical lens to generate a line with 90 degrees fan angle. This version can be seen in Fig. 1.

3.3 Calibration

For getting useful measurements, a calibration of the active stereo camera system is needed. In case of a system using a line as structured light source, the calibration process consists of describing the light plane inside the camera frame and finding the correct intrinsic parameters of the camera. We assume the camera is calibrated already, so the intrinsic parameters are known.

For finding the correct plane description, we need at least two images from different distances to a calibration pattern as basis to calculate the orientation of the plane of light within the camera frame. In comparison to a passive stereo camera, there is no need for finding the exact translation and rotation between both components of the system. The equation of the light plane is sufficient to reconstruct the 3D points along the projected line.

We implemented the calibration procedure as a Matlab script and made use of the well known *Camera Calibration Toolbox* [3]. Images showing the laser line upon a calibration pattern are used as input. The user manually chooses 8 points upon the line which will be approximated along a line using least squares to minimize the effect of inaccurate point selections. Now, the *Camera Calibration Toolbox* is used to extract the position and orientation of the checkerboard plane within the camera frame. Along with the selected points upon the laser line, 3D positions can be calculated by finding the intersection between the checkerboard plane and the reprojected line through the camera center point and the selected 2D image points.

After repeating the steps for calculating 3D positions of points along the laser line for all the collected images, the final parameters can be calculated. This is done by fitting all extracted 3D points onto a plane by using a least squares method. In addition, the 3D coordinates of the selected points can be derived with the new calibration parameters and compared to those calculated with the help of the calibration pattern.

3.4 Prototype Validation

In this section we want to present some measurements acquired by our laser line prototype in combination with a specific mapping application. A standard ICP (Iterative Closest Points) algorithm is the basis of our experiments. It is a well known algorithm and is used extensively in combination with laser scan measurements. As we compare our active stereo system to laser scan measurements, it appears convenient to use the ICP algorithm for processing the measurements. For carrying out our experiments, we used the *Mobile Robot Programming Toolkit (MRPT)* [2] which includes an ICP algorithm, different SLAM applications and other useful tools for handling the acquired measurements.

We used a mobile platform without any internal sensors and mounted a Hokuyo LRF and the active stereo system with the laser line projector on it. The Hokuyo LRF is used for comparing the results with our active stereo system so we can determine the achieved accuracy of the system. The environment in which we collected the measurements was a room with a size of about 3 by 4 meters.

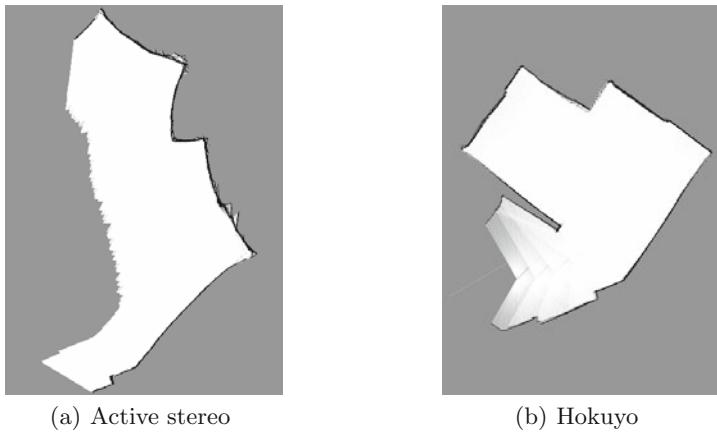


Fig. 4. Occupancy grid maps generated by the ICP-SLAM algorithm

Using the measurements as input for an ICP-SLAM system is the application we have in mind for further work with the quadrotor UAV. Therefore, we used the ICP-SLAM system provided by the MRPT libraries in combination with the acquired active stereo system measurements for building an occupancy grid map and localizing inside the map. At first, we used the acquired Hokuyo LRF measurements as input for the ICP-SLAM application. As expected, the accurate measurements lead to an adequate reconstruction of the room's layout. The result can be seen in Fig. 4(b). Now we took the synchronously acquired measurements of the active stereo system as input data. As can be seen in Fig. 4(a), the results are not as good as with the Hokuyo LRF. A connected wall was generated, but not in the right angles among the piecewise straight wall segments and in an outwards curved way.

3.5 Error Analysis

There are several possible reasons for the unexpected result of the active stereo system. Inaccurate measurements appear mainly in outer image regions, so they may be corrected with better calibration and a more optimized line extraction algorithm. The usage of a color camera in combination with a red laser reduces the effective pixel resolution of the camera because of the bayer pattern. Moreover, the characteristics of the active stereo system are different compared to those of the LRF. Especially the growing uncertainty of the active stereo system states a problem for the ICP algorithm, since it is commonly used for LRF measurements which have nearly the same noise within the whole sensor range. Additionally, the small field of view of the camera complicates the matching process of the ICP algorithm.

4 Simulation

The results that we achieved with our active stereo system encouraged us to emulate the sensor within a simulation environment. This way we can eliminate errors that result from inaccurate calibration or other hardware generated errors.

As mentioned in the introduction, USARSim was chosen to simulate the active stereo system. It is an open-source high-fidelity simulation of robots, sensors and environments based on the popular middleware Unreal Engine which is used in many commercial games. Since its introduction in 2002 it has been evolved from a plain solution for search and rescue scenarios into a versatile research platform for autonomous robots [17]. The simulation is used in the yearly *RoboCup Rescue Simulation League* as well as the *Virtual Manufacturing Automation Competition* which was held in conjunction with the *IEEE International Conference on Robotics and Automation (ICRA) 2010*.

Since the most recent version of the simulation which is based on Unreal Engine 3 is still under heavy development, we used the final version of USARSim 2004 which is based on the Unreal Engine 2.5. As the main difference between both engines is the new physics engine PhysX, which replaces the previously used Karma engine, this should not be a disadvantage.

The Unreal Engine includes an object-oriented programming language called Unreal Script that is similar to C++. The programming language allows the creation and modification of different actuators and sensor models. Currently the simulation environment contains more than 30 different robots and sensor models including for instance the well known Pioneer P2AT robot or the SICK Laser Measurement Sensor. Most of the included sensor models are range sensors. Apart from the model of an ideal camera the emulation of an omnidirectional camera [18] is the only further optical sensor.

4.1 Modelling of an Active Stereo System in USARSim

USARSim does not provide any optical sensors that would fit our requirements. This is why we had to create a new model for that purpose. The Unreal Script programming language does not include functions to emit targeted light but it offers several debug functions to draw simple shapes within the virtual world. These functions can be used to emulate the projection of a laser line.

The basic principle of our model is based on traces along a given field of view and a given angular resolution originating from the center of the sensor. We make use of the `Trace()` function which traces a given path and, in case of a collision, returns the object which has been hit as well as the location of the collision within the world. Whenever `Trace()` hits an object, the sensor marks the location of the collision by drawing a square around it on the underlying surface considering its normal vector. Then it continues with the next trace. At the same time the sensor examines whether consecutive points lie on a line without any barrier in between which indicates that they are located on the same object. Whenever such a line gets interrupted by a new point, the sensor draws the line by connecting its start- and endpoint using the `DrawStayingDebugLine()` function. Of course this

method only works as long as the angular resolution is high enough. When the angular difference between two consecutive traces is too large, the sensor might connect points which lie on a line and on a surface with the same orientation within the world but are located on different objects. On the other hand, the maximal resolution of the sensor is limited by the computational power of the hardware.

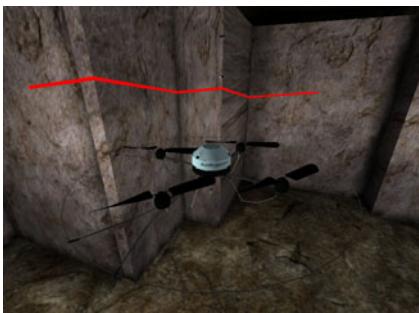
Unfortunately, hue and saturation of such a line stay constant without regarding the optical characteristics of the surface it gets drawn on. Although it would be possible to vary both based on the distance between the line and the camera, it is not possible to include the surface texture because its appearance is not described within the Unreal Engine.

The source code of the sensor model will be contributed to the USARSim project [19].

4.2 Measurement Procedure

To create our data sets we mounted the described laser line projector and an LRF on a quadrotor UAV called AirRobot and flew a certain route in a simple map only containing straight walls. The LRF was parameterized like the Hokuyo URG-04LX 2D LRF we used in the real experiment. The measurements we did with the LRF were only used to validate the calculated distances. The data sets consisted of images from the camera as well as a log file with the measured ranges of the LRF. The images were captured with a resolution of 800x600 pixels at a frame rate of approximately 20Hz.

To simulate the Hokuyo LRF we modified the existing basic laser sensor that USARSim provides. We used a FOV of 240°, an angular resolution of $\approx 0.36^\circ$ and a maximal range of 4 meters. Additionally we added gaussian noise to the measured distances. The noise was generated using the Box–Muller transform. The standard deviation was reasonably estimated at 10 millimeter [12]. Regarding the accuracy of the LRF, one has to consider that in the simulation angles are converted to integer values (so called unreal units - *uu*) between 0 (0°) and



(a) The AirRobot with the attached laser line projector.



(b) The laser line as seen by the camera.

Fig. 5. Simulation of the laser line projector

65536 (360°). But the angular resolution of the real Hokuyo LRF of 0.36° is not a multiple of $360^\circ/65536 = 0.005493^\circ$. That is why we increased the FOV of the simulated LRF to 240.007324° ($43692uu$) and set the angular resolution to 0.362548° ($66uu$) resulting in 663 measurements.

For the laser line projector we chose the same FOV as for the camera. It was set to 90.0° . The angular resolution was specified with 0.109863° ($20uu$). The resulting laser line of this configuration is shown in Fig. 5.

Originally the roll- and pitch-axis rotation angles of the AirRobot vary depending on its current linear and lateral speed including influences of the environment. As a result, the calculated 3D points have to be rotated to represent the real points in the world. Since the images were captured directly from the OpenGL back buffer of the running Unreal Engine whereas the current orientation of the robot was provided by the ground truth sensor within the simulation, it is problematic to synchronize both and associate a certain image with a given orientation. Especially because the performance of the simulation decreases rapidly when both sensors are mounted on the robot which leads to varying scan intervals of all sensors. Furthermore, the Unreal Engine 2.5 does not provide a system time in milliseconds, so we had to generate timestamps for each sensor data set after it has been read by an external program which results in another possible source of error. For these reasons we decided to set the AirRobots orientation within a constant xy-plane. That way it was not necessary to rotate the calculated 3D points.

To avoid tilting of the AirRobot including the attached laser line projector while moving we did not apply torque to the roll- and pitch-axis within the simulated model. To ensure that possible collisions with the environment will not affect the orientation of the robot we also enabled the `StayUpright` option of the physics engine (`bKStayUpright=True`, `StayUprightStiffness=200`, `StayUprightDamping=20`) so that the robot will keep its orientation upright throughout the measurement.

4.3 Line Detection

For calculating distances, the collected camera images have to be processed in terms of finding the location of the laser line. Because of the ideal characteristics of the simulated laser line projector, we can simply use the maximum values for each column of the red image channel. Calculating a mean value over the found locations gives the center of the detected laser line. Due to weak resolution in larger distances, all detected maxima are grouped into lines with a simple split-and-merge algorithm, as described for example in [15]. Afterwards the points are processed for each group in terms of a least squares optimization to get a better estimate for the line and a better resolution for each point upon it.

5 Results

Using our developed sensor model we did a similar experiment as with the real sensor. As before, we used the active stereo system in comparison with a Hokuyo

LRF. This time both devices were simulated within USARSim. The results of the subsequent ICP-SLAM processing are plotted in Fig. 6. As can be seen, the generated map based on LRF measurements is more accurate compared to the one generated with the active stereo system measurements. However, in contrast to the real world results (Section 3.4), the layout and the spacial structure of the map generated using the active stereo system corresponds closely to the map based on the Hokuyo LRF.

Again, there are two main disadvantages compared to the LRF. The field of view of 90° is more than two times smaller and the uncertainty grows for larger distances. Additionally, the decreasing resolution in larger distances also introduces errors in the line detection algorithm. Corners for example are not detected correctly, because the laser line does not bend enough and both line segments are detected as one resulting in a straight wall, as shown schematically in the top view in Fig. 7. Such errors could be decreased by using a higher resolution or a bigger baseline. Both variants are complicated. On the one hand, there is only limited space onboard the quadrotor and on the other hand, a higher resolution produces more processing load.

For improving the SLAM system, we plan to implement a probabilistic scan matching algorithm as described in [14]. The idea is to include the uncertainties of the range measurements as well as the position uncertainty of the sensor. This can be realized by replacing the error metric within the minimization step of the ICP algorithm. Instead of using the Euclidean distance, the Mahalanobis distance is used. In the case at hand, the influence of measurements with a high amount of noise will be reduced. Furthermore, the probabilistic model of the scan matching algorithm makes it possible to integrate the matching process into a probabilistic filter framework.

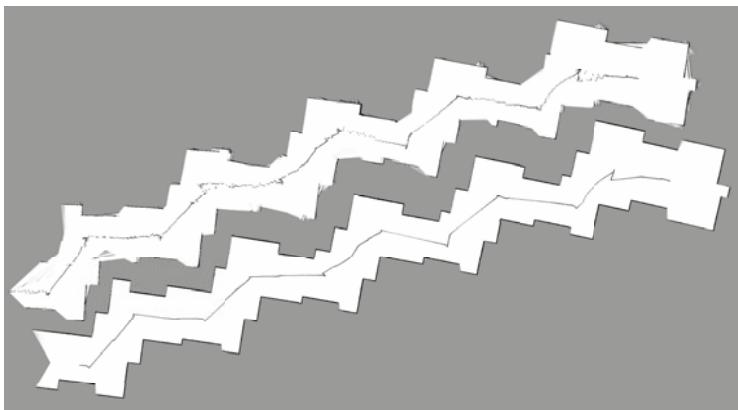


Fig. 6. Occupancy grid maps of ICP-SLAM with active stereo system (top) and with a Hokuyo LRF (bottom)

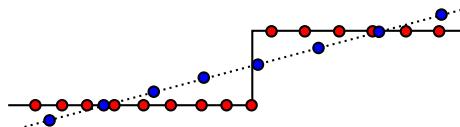


Fig. 7. Schematic measurement of a corner (black line) from short (red) and long distance (blue). Distant measurements result in a straight line without the corner.

6 Conclusions and Further Work

The achieved results from the simulation surpass those from the real scenario considerably and thus prove the feasibility of our structured light measurement system. In future work we will use the simulation to adapt the real measurement process and to improve the ICP-SLAM method referencing the special properties of our active stereo system. As an example, further research will be done regarding the used ICP-SLAM system to incorporate the growing uncertainty in larger distances. However with our work we showed that the robot simulation USARSim can also be used to emulate active vision systems.

Acknowledgement. This work has been partially funded by the European Union with the European Social Fund (ESF) and by the state of Saxony.

References

1. Achtelik, M., Bachrach, A., He, R., Prentice, S., Roy, N.: Autonomous Navigation and Exploration of a Quadrotor Helicopter in GPS-denied Indoor Environments. In: First Symposium on Indoor Flight Issues (2009)
2. Blanco, J.-L.: The Mobile Robot Programming Toolkit (MRPT) (April 2010), <http://www.mrpt.org>
3. Bouguet, J.-Y.: Camera Calibration Toolbox for Matlab. Website (2009), http://vision.caltech.edu/bouguetj/calib_doc
4. Carpin, S., Lewis, M., Wang, J., Balakirsky, S., Scappertow, C.: USARSim: a robot simulator for research and education. In: Proc. of IEEE International Conference on Robotics and Automation, ICRA 2007, Roma, Italy, pp. 1400–1405 (April 2007)
5. Grzonka, S., Grisetti, G., Burgard, W.: Towards a Navigation System for Autonomous Indoor Flying. In: Proc. of IEEE International Conference on Robotics and Automation, ICRA 2009, Kobe, Japan (2009)
6. Gurdan, D., Stumpf, J., Achtelik, M., Doth, K.-M., Hirzinger, G., Rus, D.: Energy-efficient Autonomous Four-rotor Flying Robot Controlled at 1 kHz. In: Proc. of IEEE International Conference on Robotics and Automation, ICRA 2007, Rome, Italy (April 2007)
7. He, R., Prentice, S., Roy, N.: Planning in Information Space for a Quadrotor Helicopter in a GPS-denied Environments. In: Proc. of the IEEE International Conference on Robotics and Automation, ICRA 2008, Los Angeles, CA, pp. 1814–1820 (2008)
8. Ilstrup, D., Elkaim, G.H.: Low Cost, Low Power Structured Light Based Obstacle Detection. In: ION/IEEE Position, Location, and Navigation Symposium, ION/IEEE PLANS 2008, Monterey, CA, pp. 865–870 (2008)

9. Ilstrup, D., Elkaim, G.H.: Single Frame Processing for Structured Light Based Obstacle Detection. In: ION National Technical Meeting, ION NTM 2008, San Diego, CA (2008)
10. Jiang, X., Bunke, H.: Dreidimensionales Computersehen - Gewinnung und Analyse von Tiefenbildern. Springer, Heidelberg (1996)
11. Lange, S., Sünderhauf, N., Protzel, P.: A Vision Based Onboard Approach for Landing and Position Control of an Autonomous Multirotor UAV in GPS-Denied Environments. In: Proc. of the International Conference on Advanced Robotics, ICAR 2009, Munich, Germany (2009)
12. Caprari, G., Kneip, L., Tache, F., Siegwart, R.: Characterization of the compact Hokuyo URG-04LX 2D laser range scanner. In: Proc. of International Conference on Robotics and Automation, ICRA 2009, Kobe, Japan (May 2009)
13. MikroKopter. Website (2010), <http://www.mikrokopter.com>
14. Montesano, L., Minguez, J., Montano, L.: Probabilistic Scan Matching for Motion Estimation in Unstructured Environments, pp. 3499–3504 (2005)
15. Nguyen, V., Martinelli, A., Tomatis, N., Siegwart, R.: A Comparison of Line Extraction Algorithms using 2D Laser Rangefinder for Indoor Mobile Robotics. In: Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2005, Edmonton, Canada. IEEE, Los Alamitos (2005)
16. Ribo, M., Brandner, M.: State of the Art on Vision-Based Structured Light Systems for 3D Measurements. In: International Workshop on Robotic and Sensor Environments, ROSE 2005, pp. 2–6 (October 2005)
17. Balakirsky, S., Carpin, S., Lewis, M.: Robots, Games, and Research: Success stories in USARSim. In: Workshop Proc. of the International Conference on Intelligent Robots and Systems, IROS 2009, St. Louis, Missouri, USA (2009)
18. Schmits, T., Visser, A.: An Omnidirectional Camera Simulation for the USARSim World. In: Proc. of the 12th RoboCup International Symposium, Suzhou, China (2008)
19. USARSim. Unified System for Automation and Robot Simulation (2010), <http://sourceforge.net/projects/usarsim/>

Performance Analysis for Multi-robot Exploration Strategies*

Sebastian Frank, Kim Listmann, Dominik Haumann, and Volker Willert

TU Darmstadt, Control Theory & Robotics Lab, Darmstadt, Germany

{klistman, dhaumann, vwillert}@rtr.tu-darmstadt.de

<http://www.rtr.tu-darmstadt.de>

Abstract. In this note, we compare four different exploration strategies and analyze the performance in terms of exploration time and amount of exploration per time step. In order to provide a suitable reference for comparison, we derive an upper bound for the theoretically achievable increase of explored area per time step. The comparison itself is done using a comprehensive empirical evaluation resulting in statistically significant performance differences.

Keywords: multi-robot exploration, coordination, distributed optimization.

1 Introduction

One of the major research topics in autonomous robotics is the exploration of an unknown environment – often referred to as the *exploration problem*. The goal of the exploration problem is to gain new information of the environment. For instance, a well-known application is a search and rescue mission where the environment is dangerous or even inaccessible to humans. In order to gain new information of the environment, an autonomous robot has to decide on where to move next. An intuitive solution is given by the *frontier-based exploration* [15]. Here a robot always moves to the boundary (the *frontier*), separating explored from unexplored space. Reaching a *target location* on the frontier, the robot pushes back the boundary. This continues until the entire environment is mapped, which is equivalent to the completion of the exploration task.

While the idea of frontier-based exploration is simple, the choice of the next target location is not. During the exploration progress the length of the frontier can span large distances. Hence, lots of potential target locations exist and it is not obvious where to move next to maximize the information gain and minimize the overall time required for the entire exploration. The task of assigning appropriate target locations becomes even more important in scenarios where multiple robots jointly explore the environment. Here, effective coordination of

* This research has been supported by the German Research Foundation (DFG) within the Research Training Group 1362 “Cooperative, Adaptive and Responsive Monitoring in Mixed Mode Environments”.

the group of robots is vital, as the robots have to explore different regions of the environment to finish exploration quickly.

To this end, it is not clear which of the vast number of possible strategies to prefer for exploring the environment. Thus, empirical studies increase insight to the strengths and weaknesses of the strategies compared. Together with a strategy selection, a suitable performance metric must be used for comparison. Generally, shortest-time/path, minimum-energy metrics or maximum-information-under-time/energy-limitations are available and the choice strongly depends on the scenario regarded. The former are objective only if the environment is fully explored, while the latter need to measure information-gain [4].

Related to our approach, the work in [10] evaluates different single-robot exploration strategies based on experiments. Simple, reactive strategies are compared to more sophisticated, planned behaviors with the outcome that a hybrid of both might be the best solution. Later, [1] reported in simulation that strategies mixing utility and cost tend to perform better.

In this paper, we investigate several frontier-based multi-robot exploration strategies in simulation. Hence, we extend previous work to the multi-robot domain. The exploration strategies differ in terms of how to choose appropriate target locations and how to coordinate the group of robots. A special focus is set on distributed coordination by using a partition of the environment. In the simulation we assume ideal measurements and thus neglect the localization problem to simplify the setup. Further, only convex, obstacle-free environments are regarded.

This paper is organized as follows: Section 2 introduces multi-robot exploration in terms of a distributed optimization problem and considers the theoretical optimum. Section 3 details how to do task allocation based on a partition of the environment as well as lists the used exploration strategies. In Section 4 we provide a statistical study and the results. Finally, we conclude in Section 5.

2 Multi-robot Exploration

Multi-robot exploration strategies can be classified into centralized and distributed approaches. In centralized approaches, one team leader is assigned the task allocation, i.e., a single robot coordinates all other robots and the robots then choose the control laws to fulfill the task. A variant toward distributed coordination is the locally centralized approach. Here, robots close to each other build groups that again act in a centralized fashion.

In this paper, we focus on distributed approaches: each robot *autonomously* decides what to do. However, the robots still act as a team when considering the overall behavior. In distributed approaches robots only use *locally available information*, which includes sensing data and communicated data from robots in the (local) neighborhood. Hence, the coordination is distributed. The next section details how to transform multi-robot exploration into a distributed optimization problem that meets the mentioned requirements of distributed approaches.

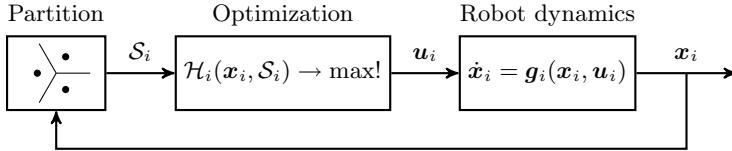


Fig. 1. Feedback loop for distributed multi-robot exploration

2.1 Representation as Optimization Problem

To transform multi-robot exploration into a distributed optimization problem we first have to find a mathematical notation. To this end, let \mathbf{x}_i be the state vector of robot i for $i \in \{1, \dots, N\}$. Further, define $\mathcal{Q} \subset \mathbb{R}^2$ as the environment and $\mathcal{S} \subseteq \mathcal{Q}$ as the explored area. Next, we introduce an objective function $\mathcal{H}(\mathbf{x}_1, \dots, \mathbf{x}_N, \mathcal{S})$ that we want to optimize with respect to the state vectors \mathbf{x}_i :

$$\mathcal{H}(\mathbf{x}_1, \dots, \mathbf{x}_N, \mathcal{S}) \longrightarrow \text{max!} \quad (1)$$

Eq. (1) represents a global optimization problem suited for centralized exploration strategies. In order to perform the optimization of (1) distributively, we decompose the objective function \mathcal{H} into N independent objective functions

$$\mathcal{H}(\mathbf{x}_1, \dots, \mathbf{x}_N, \mathcal{S}) = \sum_{i=1}^N \mathcal{H}_i(\mathbf{x}_i, \mathcal{S}). \quad (2)$$

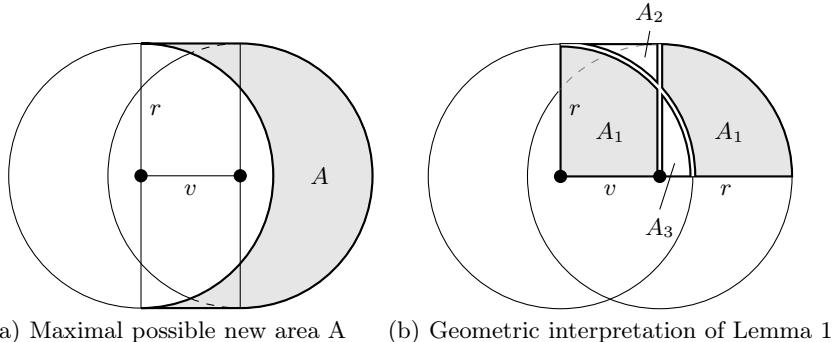
The difference to (1) is that in (2) each robot creates and optimizes its objective function autonomously, i.e., the robots perform the optimization distributively.

So far, each robot relies on the entire information in \mathcal{S} , meaning that all robots have to synchronize their map data, which equals using a globally shared map of the environment. To circumvent this problem, we define a *partition of the environment* such that each robot i is assigned the information \mathcal{S}_i with $\mathcal{S} = \cup_{i=1}^N \mathcal{S}_i$. Hence, we get a distributed optimization problem [5] of the form

$$\mathcal{H}(\mathbf{x}_1, \dots, \mathbf{x}_N, \mathcal{S}) = \sum_{i=1}^N \mathcal{H}_i(\mathbf{x}_i, \mathcal{S}_i) \longrightarrow \text{max!} \quad (3)$$

This optimization problem requires each robot to only have local knowledge \mathcal{S}_i of the environment that can be acquired by local communication with robots in the neighborhood. Fig. 1 visualizes the properties of (3) as closed feedback loop [6].

The basic idea works as follows: By using a partition \mathcal{S}_i of the available information each robot i autonomously creates and optimizes its objective function \mathcal{H}_i to derive an appropriate control input vector \mathbf{u}_i . The control input vector changes the state vector \mathbf{x}_i according to the robot dynamics. This in turn changes the partition and the process begins anew.

(a) Maximal possible new area A (b) Geometric interpretation of Lemma 1**Fig. 2.** Time-optimal exploration with vision radius r and traveled distance v

2.2 Time-Optimal Case

In order to evaluate the effectiveness of an exploration strategy, it is useful to use the time-optimal exploration strategy as reference. In the time-optimal case, we assume that a single robot is able to map the maximal region it is able to scan within a traveled distance $v \in \mathbb{R}$. For simplicity, we assume that a robot uses a 360° scanner to map the environment within a given vision radius $r \in \mathbb{R}$, see Fig. 2(a). Then, the following lemma holds.

Lemma 1. *Given a robot i at position $\mathbf{x}_i \in \mathbb{R}^2$ with 360° sensing capabilities limited by the vision radius $r \in \mathbb{R}$, a maximal area of*

$$A = 2rv \quad (4)$$

can be explored by the robot for a traveled distance v .

Proof. The case $v \geq r$ is trivial. Hence, let $v < r$. According to Fig. 2(a) the half $A/2$ of the maximal area A to be explored can be divided in two sub-areas A_1 and A_2

$$\frac{1}{2}A = A_1 + A_2. \quad (5)$$

The area A_1 is present two times in Fig. 2(b), because it is given as $A_1 = \frac{1}{4}r^2\pi - A_3$. Therefore, area A_2 is given by the rectangle area rv and A_1 and reads

$$A_2 = rv - A_1. \quad (6)$$

Inserting (6) into (5) leads to the maximal area $A = 2rv$. \square

Given an environment \mathcal{Q} , the minimal amount of iterations needed to explore the entire environment in the single-robot case is given by $\#it = \mathcal{Q}/(2rv)$. In the multi-robot case, we assume that all robots behave according to (4). Hence, as mentioned in [6], the minimal amount of iterations for N robots jointly exploring the environment is given by

$$\#it = \frac{1}{N} \frac{\mathcal{Q}}{2rv}. \quad (7)$$

The amount of iterations given in [7] is time-optimal from a purely theoretical perspective. In reality, the amount of time needed depends on the initial robot positions, the robot dynamics, the (usually) non-convex environment and communication and sensing capabilities. Still, the time-optimal case is suitable for reference, as it provides a lower bound for the time needed to explore the entire environment.

3 Distributed Optimization

3.1 Partitioning Policies

As aforementioned, multi-robot exploration is seen as a fully distributed optimization problem. In order to facilitate this, the exploration task must be split nicely among the members of the group. This task allocation is done using particular partitions of the known space. Hence, every robot is assigned a region of dominance where the optimization is conducted. If in addition, this dominance region is convex, collision avoidance comes as a byproduct of the partitioning and does not need to be regarded anymore.

One such partition is given by the *Voronoi diagram* $\mathcal{V} = \{\mathcal{V}_1, \dots, \mathcal{V}_N\}$ [3]. Each *Voronoi cell* \mathcal{V}_i is constructed by

$$\mathcal{V}_i = \{\mathbf{q} \in \mathcal{Q} \mid \|\mathbf{q} - \mathbf{x}_i\| \leq \|\mathbf{q} - \mathbf{x}_j\|, j = 1, \dots, N\}, \quad (8)$$

with the generator points \mathbf{x}_i (in our case these are the robot positions). Given a convex environment, every Voronoi cell is also convex. Moreover, the Voronoi diagram can be computed using local knowledge of the Voronoi neighbors [5].

Another partition offering the same convexity feature but more degrees of freedom is a *Power* or *Laguerre diagram* $\mathcal{L} = \{\mathcal{L}_1, \dots, \mathcal{L}_N\}$ [2]. This time, each *Power cell* is given by

$$\mathcal{L}_i = \{\mathbf{q} \in \mathcal{Q} \mid \|\mathbf{q} - \mathbf{x}_i\|^2 - w_i \leq \|\mathbf{q} - \mathbf{x}_j\|^2 - w_j, j = 1, \dots, N\}. \quad (9)$$

Here, w_i denotes a weight associated to the generator point \mathbf{x}_i . Taking only positive weights $w_i \in \mathbb{R}^+$ an interpretation is as follows: Every Power point (\mathbf{x}_i, w_i) can be seen as a circle of radius $\sqrt{w_i}$ around \mathbf{x}_i . Given an arbitrary point $\mathbf{q} \in \mathcal{Q}$, $\|\mathbf{q} - \mathbf{x}_i\|^2 - w_i > 0$ and $\sqrt{\|\mathbf{q} - \mathbf{x}_i\|^2 - w_i}$ is the Euclidean distance between \mathbf{q} and a tangent point to the circle, such that \mathbf{q} lies on the tangent as well. In contrast to Voronoi cells, Power cells might be empty (i.e., a generator does not have a cell) and the generators can lie outside their own cell. Further, Power diagrams can also be computed distributively [11].

Generally, by assigning specific weights, Power diagrams offer an additional degree of freedom to change the size of the dominance regions by changing the weights of the robots. This can be used to partition the environment more efficiently among the robots compared to the use of Voronoi cells. However, if all weights are chosen to be equal, a Voronoi diagram is recovered.

3.2 Primary Exploration Strategies

After splitting the optimization problem among the robots, we need to specify the primary exploration strategies used for later comparison. Each will be explained in short together with an appropriate objective function \mathcal{H}_i .

MinDist. This simple strategy seeks the closest point (with respect to the Euclidean distance) on the frontier to each robot. Hence, the search is restricted to the frontier cells. It is a standard approach in robotics motion planning and control [9] and the objective function together with the optimization is given by

$$\mathcal{H}_{\text{MinDist},i}(\mathbf{x}_i) = -\frac{1}{2}\|\mathbf{q} - \mathbf{x}_i\|^2 \Rightarrow \arg \max_{\mathbf{q} \in \partial \mathcal{S}_i} \mathcal{H}_{\text{MinDist},i}(\mathbf{x}_i). \quad (10)$$

MaxArea. This strategy lets every robot move into the direction where the (estimated) unexplored area is maximal. It is similar to the approaches [12,13], as large distances are penalized. Further, unexplored parts hidden behind the borders of the environment should not be regarded.

In order to estimate the newly explored area, the convex hull \mathcal{C} connecting and including the two circles around the robot position \mathbf{x}_i and one point \mathbf{q} on the frontier in the respective dominance region of the robot is regarded. The circles are identified with the vision radius of each robot, respectively. Then, the newly explored area is given by the intersection of the convex hull \mathcal{C} with the unknown space $\mathcal{U} = \mathcal{Q} \setminus \mathcal{S}$. Therefore, the objective function and the optimization problem are given by

$$\mathcal{H}_{\text{MaxArea},i}(\mathbf{x}_i) = \|\mathbf{q} - \mathbf{x}_i\|^{-2} \int_{\mathcal{C} \cap \mathcal{U}} d\mathbf{q} \Rightarrow \arg \max_{\mathbf{q} \in \partial \mathcal{S}_i} \mathcal{H}_{\text{MaxArea},i}(\mathbf{x}_i). \quad (11)$$

DisCoverage. This recently developed strategy is explained in detail in [6]. In short, an (artificial) orientation δ_i is introduced for each robot and optimized to find a suitable exploration direction.

In contrast to the previous strategies, this time all the points on the frontier of the respective dominance region will influence the overall result instead of only a single one. The objective function, combining distance and orientation costs, and the optimization problem are stated as

$$\begin{aligned} \mathcal{H}_{\text{DisCoverage},i}(\mathbf{x}_i, \delta_i) &= \int_{\partial \mathcal{S}_i} \exp \left(-\frac{\alpha^2}{2\theta^2} - \frac{\|\mathbf{q} - \mathbf{x}_i\|^2}{2\sigma^2} \right) d\mathbf{q} \\ &\Rightarrow \arg \max_{\delta_i \in [-\pi, \pi]} \mathcal{H}_{\text{DisCoverage},i}(\mathbf{x}_i, \delta_i), \end{aligned}$$

with the angle $\alpha = \triangle(\mathbf{q} - \mathbf{x}_i) - \delta_i$. As can be seen, the objective function is a Gaussian distribution of the orientation with standard deviation θ , denoted as the opening angle, and of the distance with standard deviation σ .

Potential fields. The last strategy interprets the robots as (positively) charged particles in an electrical field generated by the (negatively) charged frontier cells. In the following it is referred to as the E-Field strategy. Following [8] such methodologies posses inherent limitations but their straightforward implementation justifies the inclusion in this comparison. For simplicity, all the charged particles are taken to be point charges, so that the superposition principle holds. Moreover, every particle is only influenced by the point charges within its dominance region. Consequently, the objective function and the optimization problem read as

$$\mathcal{H}_{\text{E-Field},i}(\mathbf{x}_i) = \frac{e^+}{4\pi\varepsilon_0} \sum_{\mathbf{q} \in \partial\mathcal{S}_i} \frac{1}{\|\mathbf{x}_i - \mathbf{q}\|} \quad \Rightarrow \quad \arg \max_{\mathbf{q} \in \partial\mathcal{S}_i} \mathcal{H}_{\text{E-Field},i}(\mathbf{x}_i). \quad (12)$$

Here, ε_0 is the permittivity of the free space and e^+ denotes a positive charge.

4 Simulation of Different Exploration Strategies

4.1 Simulation Environment

The simulation of different exploration strategies was realized in Matlab. For complexity reasons we assume the following: Firstly, all robots are identical and able to explore their environment inside a predefined sensor radius exactly. Inaccuracies during localization and map generation are neglected and the environment will always be convex and obstacle-free. Furthermore, we assume a fully connected communication network between the robots and new information about the environment is stored in a common global map. Note that this is not necessary using our approach as simple connectedness of the communication network is sufficient but this eases up the computations done.

A grid is used to represent the environment, whereby each grid cell can have one of the following states: i) *unknown*: cell not explored yet (initial state), ii) *free*: cell explored and not occupied by an obstacle, iii) *occupied*: cell explored and occupied by an obstacle and iv) *frontier*: cell not explored yet, but separates explored from unexplored space. Since our approach considers obstacle-free environments only, occupied cells always belong to the boundary of the environment.

In our simulation runs five identical robots with a predefined maximum speed (maximum step size per iteration) and panoramic view with a pre-specified radius are used to explore an unknown area. At the beginning of a simulation each grid cell is marked as unknown. Each grid cell that is, in the course of simulation, totally covered by the sensor of a robot, will be marked as free or occupied. An unknown grid cell that is only partially covered by the sensor of a robot is marked as a frontier cell.

4.2 Empirical Study

In order to compare the different exploration strategies, a comprehensive study for solving the multi-robot exploration problem was performed. All exploration

strategies were tested in 20 different maps each with the size of 100×100 units. The maps include 17 randomly generated and three special cases a circle, a square and a narrow rhombus. For every map, 50 different simulation runs with randomly generated robot start positions and orientations were done, respectively. This gives a total number of 1000 scenarios each strategy was tested in.

The goal of a simulation run was to completely explore the respective area. For further analysis and as a primary measure of performance, the number of iterations needed to achieve this goal was used. As the resolution of the grid map was chosen to be 1.0, the robots had to explore 100×100 grid cells, respectively. The maximum speed was limited to 2 units per iteration and for all sensors, radii of 5 units were chosen.

4.3 Results

The objectives of the study were to compare the performance of the different primary exploration strategies and analyze the effect of different influence factors, e.g. the partitioning policy or secondary strategies, on the exploration behavior.

To this end, in the first part of our study we compared the four primary exploration strategies. In all four exploration strategies we used a partitioning policy based on Voronoi diagrams. Fig. 3 visualizes the distribution of our test measure (the number of required iterations) in the form of boxplots [14]. Such plots are particularly helpful in comparing data sets. Note that the box contains all data from the lower to the upper quartile (the 25th and 75th percentile) and the line shows the median (the 50th percentile). Outliers are displayed by crosses outside of the whiskers (dashed).

Apart from a considerable large amount of outliers, we see in Fig. 3(a) that, except for strategy no. 4, the performance of all strategies barely differs. The best performance is achieved with the DisCoverage strategy that required 137.38 iterations on average in case of a complete exploration. This is equivalent to an average performance of 43.69%. The worst performance is achieved with the E-field strategy that needed 161.81 iterations on average being equivalent to a performance of 37.29%. But as we can see in Fig. 3(b) and (c) the situation

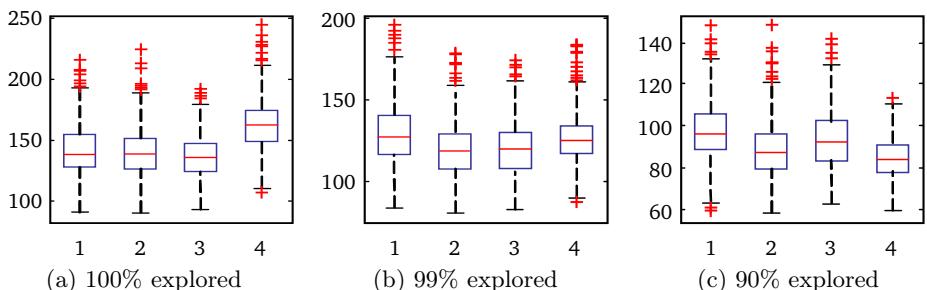


Fig. 3. Performance of the exploration using primary exploration strategies 1) MinDist, 2) MaxArea, 3)DisCoverage and 4) E-Field. In this plot, *the-lower-the-better* holds.

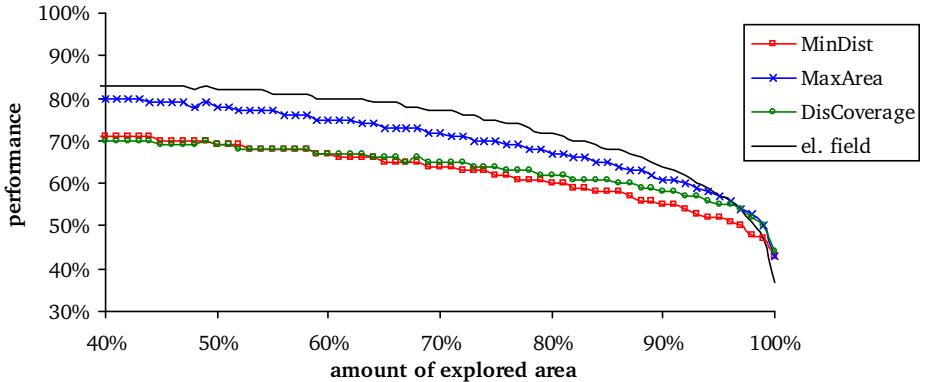


Fig. 4. Plot of the evolution of the performance (in percent of the theoretical optimum) of the exploration process over the explored area. In this plot, *the-higher-the-better* holds.

changes when the goal is an exploration of only 99% or 90% of the whole environment. Surprisingly, the E-field strategy achieves the best performance in case of a required degree of exploration of 90%, depicted in Fig. 3(c). Therefore, it is worth looking at the evolution of the performance over the explored area as given in Fig. 4. It can be seen that using the MaxArea or the E-field strategy results in a very good performance over a wide spectrum of the degree of exploration. In particular for the very simple E-field strategy this is a remarkable result. Only within the last 10% the performance drops significantly. This is due to the generation of many small, unexplored islands in the environment. Further, this shows that the choice of an exploration strategy should depend on the degree of exploration one is willing to achieve or that exploration strategies should be changed throughout the exploration process.

Furthermore, we analyzed the influence of the partitioning on the exploration performance. To this end, we compared three different partitioning policies, namely Voronoi diagrams, Power diagrams and using no partitioning, together with one primary exploration strategy, MinDist. For the case of no partitioning our solution uses a collision avoidance feature similar to the one proposed in [7] but employing a simpler sensor model. The results in Fig. 5 indicate that, using a partitioning policy yields better exploration performance compared to the case without any partitioning. The best results are achieved by using Power diagrams with 137.96 (43.76%) iterations on average in case of a complete exploration. This almost recovers the speed of the DisCoverage strategy when Voronoi partitions are used. The weight of each Power cell is computed by $w_i = k_i d_{\min}$, where $d_{\min} = \min \|p_i - p_j\|, \forall j = 1, \dots, N, j \neq i$ and $k_i \in]0, 1]$ is given by

$$k_i^{(k+1)} = \begin{cases} k_i^{(k)} + \frac{1}{10} & \text{if } \frac{NA_{u,i}}{A_u} < \frac{1}{2}, \\ k_i^{(k)} - \frac{1}{10} & \text{if } \frac{NA_{u,i}}{A_u} > \frac{3}{2}, \\ k_i^{(k)} & \text{else} \end{cases} \quad (13)$$

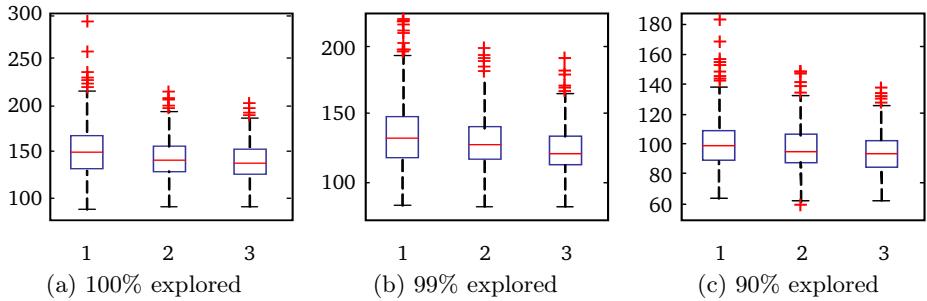


Fig. 5. Performance of the exploration of the MinDist strategy using different partitioning policies 1) none, 2) Voronoi diagrams and 3) Power diagrams

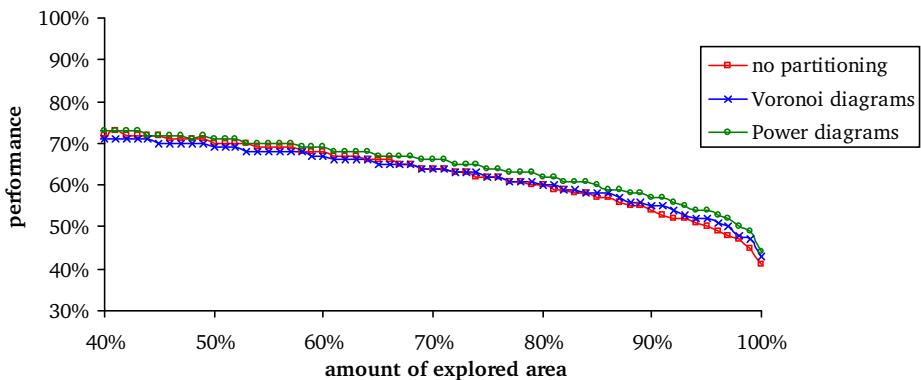


Fig. 6. Plot of the evolution of the performance (in percent of the theoretical optimum) of the exploration process using the MinDist strategy with different partitioning policies

with the number of robots N and the unknown area (in each cell) $A_{u(i)}$. Moreover, Fig. 6 shows that Power diagrams are superior at any time in the exploration process. Hence, using a partition speeds up the exploration process.

Finally, we take the MinDist strategy using a Voronoi partition again as the basic strategy and analyze the influence of two enhancements: Firstly, a secondary strategy is proposed that switches the search for the point with the minimum distance from a local to global one. This is done whenever no frontier cells are in the respective Voronoi cell anymore. Secondly, an information gain near the known border of the environment is modeled. This is used to increase the weight of points that are close to already discovered points of the border. So the optimization result is shifted towards such points, resulting in a “traveling along the border”-behavior of the robots.

Clearly, both additional options have a positive influence on the exploration performance as depicted in Fig. 7. The best results were achieved by using both options in combination with only 122.39 (48.89%) iterations on average for a complete exploration. Fig. 8 shows that the main part of this improvement is due to the information gain modeling near border cells.

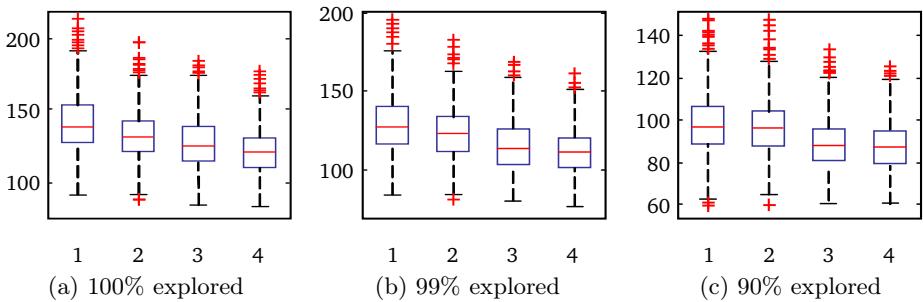


Fig. 7. Performance of the exploration of the MinDist strategy using additional options 1) stand alone, 2) with global search for closest point, 3) with information gain close to border cells and 4) with a combination of 2) and 3)

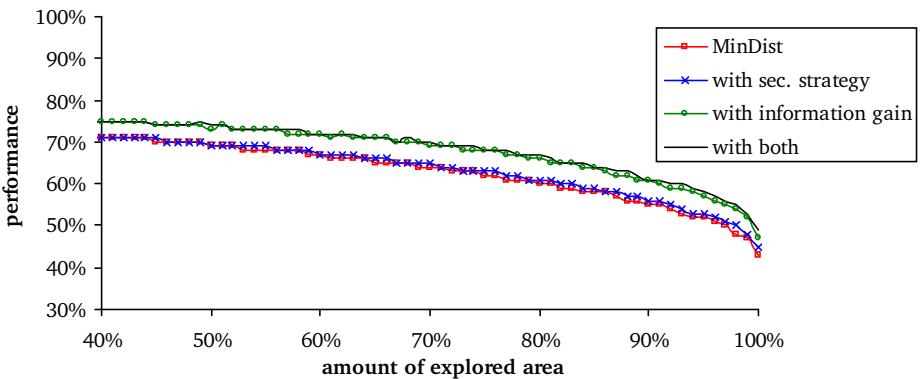


Fig. 8. Plot of the evolution of the performance (in percent of the theoretical optimum) of the exploration process using the MinDist strategy with different additional options

5 Conclusion

In this article a comprehensive study was used to identify important influence factors on the multi-robot exploration problem. To this end, four different primary exploration strategies were compared on 20 different maps with 50 different initial conditions each. The key parameter for the comparison was the number of iterations needed to completely explore each map and the progress of the exploration. Based on rather strong simplifications on the maps and on the robots' sensing abilities, the following results can be concluded: The choice of the method strongly depends on the satisfiable degree of exploration, as some methods performed well up to the point where 90% of the map were explored. Moreover, using a partitioning policy that assigns dominance regions to each robot not only simplifies collision avoidance but also speeds up the exploration significantly.

So far two important influence factors have not been varied – the number of robots and their maximum speed. Identifying the impact of these is left for future work. In addition, it would be nice if some of the simplifying assumptions could be dropped.

References

1. Amigoni, F.: Experimental evaluation of some exploration strategies for mobile robots. In: Proc. 2008 IEEE Int. Conf. Rob. & Autom., pp. 2818–2823 (2008)
2. Aurenhammer, F.: Power diagrams: Properties, algorithms and applications. *SIAM J. Comput.* 16(1), 78–96 (1987)
3. Aurenhammer, F., Klein, R.: Voronoi diagrams. In: *Handbook of Computational Geometry*, ch. V, pp. 201–290. North-Holland, Amsterdam (2000)
4. Calisi, D., Farinelli, A., Iocchi, L., Nardi, D.: Multi-objective exploration and search for autonomous rescue robots. *J. Field Rob.* 24(8-9), 763–777 (2007)
5. Cortés, J., Martínez, S., Karatas, T., Bullo, F.: Coverage control for mobile sensing networks. *IEEE Trans. Rob. Autom.* 20(2), 243–255 (2004)
6. Haumann, A.D., Listmann, K.D., Willert, V.: Discoverage: A new paradigm for multi-robot exploration. In: Proc. 2010 IEEE Int. Conf. Rob. & Autom., pp. 929–934 (2010)
7. Hussein, I., Stipanovic, D.: Effective coverage control for mobile sensor networks with guaranteed collision avoidance. *IEEE Trans. Control Syst. Technol.* 15(4), 642–657 (2007)
8. Koren, Y., Borenstein, J.: Potential field methods and their inherent limitations for mobile robot navigation. In: Proc. 1991 IEEE Int. Conf. Rob. & Autom., pp. 1398–1404 (1991)
9. LaValle, S.M.: *Planning Algorithms*. Cambridge University Press, Cambridge (2006)
10. Lee, D., Recce, M.: Quantitative evaluation of the exploration strategies of a mobile robot. *Int. J. Rob. Res.* 16(4), 413–447 (1997)
11. Pavone, M., Arsie, A., Frazzoli, E., Bullo, F.: Equitable partitioning policies for robotic networks. In: Proc. 2009 IEEE Int. Conf. Rob. & Autom., pp. 2356–2361 (2009)
12. Simmons, R., Apfelbaum, D., Burgard, W., Fox, M., an Moors, D., Thrun, S., Younes, H.: Coordination for multi-robot exploration and mapping. In: Proc. AAAI Nat. Conf. Art. Intell. (2000)
13. Solanas, A., Garcia, M.A.: Coordinated multi-robot exploration through unsupervised clustering of unknown space. In: Proc. IEEE/RSJ Int. Conf. Intell. Rob. & Syst., pp. 852–858 (2004)
14. Tukey, J.W.: *Exploratory Data Analysis*. Addison-Wesley, Reading (1977)
15. Yamauchi, B.: A frontier-based approach for autonomous exploration. In: Proc. IEEE Int. Symp. Comput. Intell., Rob. & Autom., pp. 146–151 (1997)

Dynamic Modeling of the 4 DoF BioRob Series Elastic Robot Arm for Simulation and Control

Thomas Lens, Jürgen Kunz, and Oskar von Stryk

Simulation, Systems Optimization and Robotics Group, Technische Universität
Darmstadt, Hochschulstraße 10, 64289 Darmstadt, Germany
{lens,stryk}@sim.tu-darmstadt.de
www.sim.tu-darmstadt.de, www.biorob.de

Abstract. This paper presents the modeling of the light-weight BioRob robot arm with series elastic actuation for simulation and controller design. We describe the kinematic coupling introduced by the cable actuation and the robot arm dynamics including the elastic actuator and motor and gear model. We show how the inverse dynamics model derived from these equations can be used as a basis for a position tracking controller that is able to sufficiently damp the oscillations caused by the high, nonlinear joint elasticity. We presents results from simulation and briefly describe the implementation for a real world application.

Keywords: flexible joint robot, modeling, control, biologically inspired robotics, series elastic actuation.

1 Introduction

Elasticity in the actuation of robotic arms was for a long time seen as undesirable. When introducing a series elasticity in the joint actuation, reduced torque and force bandwidth and increased controller complexity for oscillation damping and tracking control are the result. Research on series elastic actuators [13] however, showed that mechanical compliance in the joint actuation can simplify force control in constrained situations, increase safety because of the low-pass filtering of torque and force peaks between the decoupled joint and gearbox , and increase performance of specific tasks because of the possibility to store mechanical energy in the elasticity. For example, the increase of performance for throwing was examined in [18]. In [19], an actuation approach with two motors per joint increasing torque bandwith without compromising safety was examined. A classification of elastic joint actuation principles is given in [17].

Flexible link manipulators are also subject to current research. But these systems are even harder to control, especially when dealing with multiple degrees of freedom, and do not introduce significant advantages compared to joint elasticity. An overview over research on flexible joint and link systems with an emphasis on flexible links is given in [6].

The modeling of an elastic joint robot with a reduced model was presented in [14]. The complete model and analysis of the model structure was derived in [15],

and complemented by [7]. The control of elastic joint robots with a controller relying solely on motor-based sensor data was presented in [16]. The use of full state feedback was examined in [1] and [5]. Feedforward/feedback control laws are covered in [3]. A good overview over modeling and control methods for robot arms with joint and link flexibility is given in [4].

2 BioRob Arm Design

In this work, the BioRob arm, an equilibrium-controlled stiffness manipulator, is analyzed. The mechanical design of this robot arm is depicted in Figure 1. The arm consists of a very lightweight structure with rigid links, elastically actuated by DC motors driving the joints by pulleys and cables with built-in mechanical compliances. Alternative actuation concepts such as pneumatic muscles [8] exhibit inherent compliance and omit the need for gearboxes, but are slower, have a restricted range of operation and are suited for mobile applications only to a very limited extent. Electrical motors on the other hand are robust, allow high speeds, exhibit excellent controllability and are well suited for highly mobile applications. The construction and actuation principle is described in more detail in [11,9].

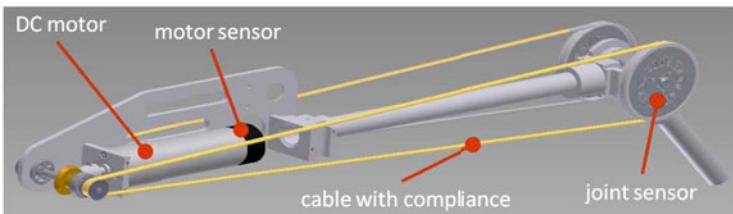


Fig. 1. BioRob robot arm actuation principle

The general properties of series elasticity in the actuation were described in Section 1. The specific properties of the BioRob arm concept compared to other series elastic concepts are reduced link mass and inertia (a total mass of 4kg), reduced power consumption, and a significantly lower joint stiffness (ranging between 4 and 20Nm), in total resulting in increased safety for applications with direct human-robot interaction. As a downside, the use of cable and pulley actuation increases friction and the series elasticity with particularly low joint stiffness demands special efforts regarding oscillation damping. Therefore, an appropriate controller structure is needed.

3 Kinematics Model

The BioRob robot arm consists of four elastically actuated joints. To model the kinematics of the robot arm, it is sufficient to model the rigid link structure,

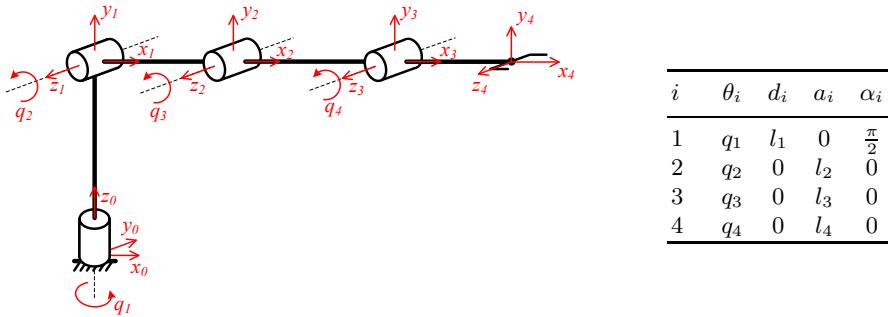


Fig. 2. BioRob 4 DOF robot arm kinematic structure and table with DH parameters

because the joint elasticity has no effect on kinematics when using joint sensors. Link elasticity is negligible for loads not exceeding the nominal load. Thus, the same modeling methods as used for rigid link robots without elasticity in the actuation can be used, as can be seen in picture 2, where the DH-parameters are listed. As mentioned above, the kinematics model depends only on the joint angular positions q_i and not on the motor angular positions θ_i .

Especially for control, the equilibrium positions of the joints are important. These are the joint positions q_i and motor positions θ_i where the elasticity between motor and joint produces no torque. Normally, the equilibrium position of the joints corresponds to the current position of the motors. This is not the case, however, if the motor is mounted neither on the link it actuates nor on the previous link. Then an additional deflecting pulley becomes necessary. As depicted in Figure 3, the motor driving joint four is fixed to link two and coupled with a deflection pulley (radius r_{d3}) in joint three to joint four. Because of the kinematic coupling between link four and three, the equilibrium position of this motor not only depends on q_4 , but also on q_3 , because the cable wraps around the guiding pulley. The amount of cable that wraps around the pulley on joint three is equal to the amount of cable that unwinds from the pulley driving joint four:

$$r_{d3} q_3 = -r_4 \Delta q_4 \quad \Rightarrow \quad \Delta q_4 = -\frac{r_{d3}}{r_4} q_3 . \quad (1)$$

This correction term has to be considered when calculating the equilibrium positions. The resulting joint equilibrium position vector of the manipulator for given angular motor positions $\boldsymbol{\theta}$ and joint positions \boldsymbol{q} is:

$$\hat{\boldsymbol{q}}(\boldsymbol{\theta}, \boldsymbol{q}) = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 - \frac{r_{d3}}{r_4} \cdot q_3 \end{pmatrix} = \boldsymbol{\theta} - \boldsymbol{\alpha}_c(\boldsymbol{q}) . \quad (2)$$

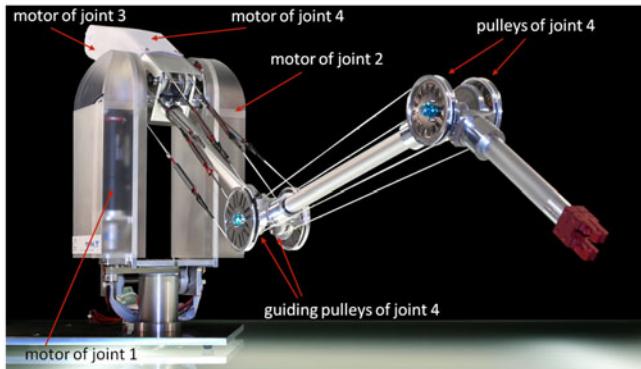


Fig. 3. Position of motors on the BioRob arm

4 Dynamics Model

The dynamics model is used for simulation and controller design. In simulation, the behavior of the model can be studied without the need to perform time-consuming experiments, also avoiding wear of the hardware. It is especially useful for examining scenarios such as collision detection, which are difficult to perform with the hardware. In simulation, it is also possible to provide additional data, that would be difficult to measure, such as collision reaction forces.

However, only effects can be studied that are modeled with sufficient accuracy. The most important effects are the robot arm dynamics consisting of the dynamics of the rigid structure and the joint elasticity, described in Section 4.2. The dynamics model of the motors (Section 4.1) is required to be able to take actuator saturation into account. It also allows to simulate the torque loads and peaks caused by collisions.

The primary requirement for the dynamics model is steady state accuracy, which is important for the controller design. Besides the steady state equations and parameters of motors and robot arm, the nonlinear joint elasticity is to be modeled accurately, shown in Section 4.2, due to the low elasticity and therefore large spring deflection. The second important requirement is the accurate modeling of the joint oscillations caused by the joint elasticity. An accurate model of this behavior allows for a better controller design in simulation.

4.1 Motor Model

Motors can be controlled to deliver a desired torque τ_m which can be seen as the input of the system. Instead of using the simple torque source model, a more complete model of the electrical motor dynamics allows for the examination of the motor currents and voltages, which are bounded in reality. The motors used in the BioRob robot arm are DC motors. The effect of the armature inductance

can be neglected compared to the other dynamics of the motor. The electrical dynamics can be described as:

$$u = R_a i + k_v \cdot \dot{\theta} = \frac{R_a}{k_t} \tau_m + k_v \cdot \dot{\theta} \quad (3)$$

with input voltage u , armature resistance R_a , torque constant k_t , speed constant k_v and generated motor torque τ_m , which drives the rotor. The mechanical dynamics of a freely rotating motor are:

$$I_m \ddot{\theta} + d_m \dot{\theta} = \tau_m . \quad (4)$$

When connected to the robot, the mechanical motor model has to be modeled together with the robot arm mechanics to receive the mechanical dynamics equations of the coupled system. This is described in the following section.

In most cases, electrical motors require gearboxes to achieve the desired torques. These can be modeled with a transmission ratio z reducing the speed $\dot{\theta}$ of the motor:

$$\dot{\theta}^* = \frac{1}{z} \dot{\theta} \quad \text{with} \quad |z| > 1 , \quad (5)$$

which increases the torque τ_m of the motor. The gearbox also introduces additional friction d_g and inertia I_g . For a compact model representation, all motor variables and parameters will be used with respect to the joint side, as *reflected variables*:

$$\tau_m^* = z \cdot \tau_m - d_g \cdot \dot{\theta}^* \quad (6)$$

$$I_m^* = z^2 \cdot I_m + I_g \quad (7)$$

$$d_m^* = z^2 \cdot d_m + d_g . \quad (8)$$

All variables marked with an asterisk are reflected variables calculated with respect to the joint. Further on, only these will be used and for the sake of simplicity, they will not be asterisked.

4.2 Robot Arm Dynamics

The model of a BioRob joint as shown in Figure 4 can be transformed into a model of a series elastically actuated joint shown in Figure 5. This is possible if the mass of the cables and elastic parts is so small that the kinetic energy of these elements can be neglected compared to the kinetic energy of the other mechanical robot arm parts. In this case, the transformation can be performed by adapting some of the model parameters. The mass of the motor can be added to the link it is fixed to and the transmission ratios of the gearbox and the cable and pulley elements can be multiplied. All variables can then be calculated as reflected variables with respect to the joint side, as described in the former section.

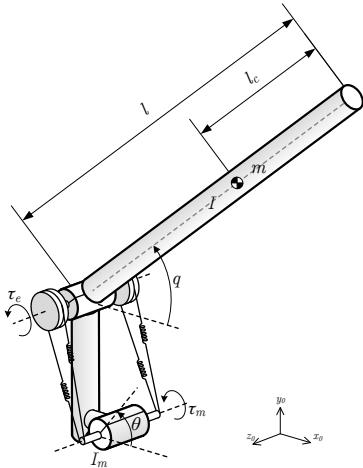


Fig. 4. Single joint of the BioRob arm

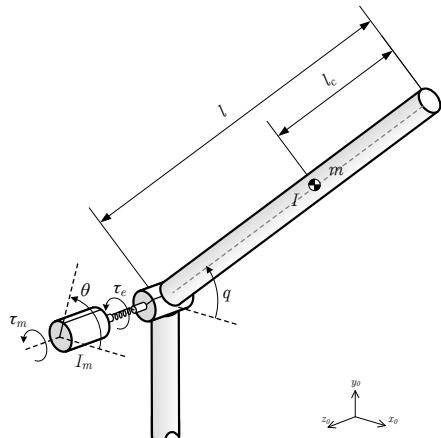


Fig. 5. Joint actuated by a Series Elastic Actuator

The nonlinear joint spring characteristics curve is a function of the deviation of the joint position q_i of its equilibrium position \hat{q}_i , which is normally the motor position θ_i , but can also be dependent of the position of previous joints, as described in Section 3.

$$\tau_e = k_e(\hat{q} - q). \quad (9)$$

Figure 6 shows the spring characteristic of the fourth joint. The joint elasticity of each joint was chosen according to the expected maximum load torque.

The multibody dynamics of the robot arm and the motors can be described by using the reduced model of elastic joint robots. For formal derivation of these equations see [14][16]:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + D\dot{q} + g(q) = \tau_e \quad (10)$$

$$I_m\ddot{\theta} + D_m\dot{\theta} + \tau_e = \tau_m \quad (11)$$

Equation (10) describes the dynamics of the rigid structure with mass matrix M , Coriolis matrix C , gravity torque vector g , and diagonal friction matrix D . Equation (11) describes the dynamics of the motor rotors with the diagonal motor rotor inertia matrix I_m , diagonal friction matrix D_m and motor torque τ_m . The mass matrix consist of the inertia and mass of the links, including the motor masses, which are added to the links where they are mounted (see Figure 3). The mass of the cables and mechanical elasticity is negligible. The diagonal matrix I_m consists of the motor rotor inertias $I_{m_{izz}}$ with respect to the rotor rotating axis. These are the reflected motor inertia values as stated in Section 4.1.

Because the motors are mounted on the first and second joint and therefore moving with low kinetic energy, and because of the large reduction ratios (overall

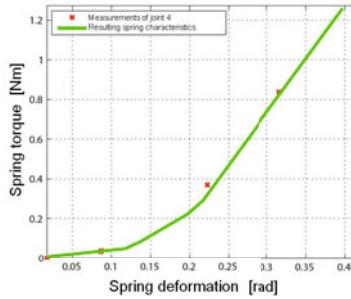


Fig. 6. Nonlinear spring characteristic of joint 4

reduction ratios z_i between 100 and 150), it is possible to neglect the effects of the inertial couplings between the motors and the links, so that the reduced model can be used, as stated in [14]. Otherwise, a more general model would have to be used [15]. Also, the fact that the motors are not located in the joints but mounted on the links, would have to be considered and modeled [12].

5 Inverse Model for Tracking Control

5.1 Inverse Model

For sufficiently high (but still finite) joint stiffness K_e , a singular perturbation model can be used [10]. It consists of a slow subsystem, given by the link dynamics equations, and a fast subsystem describing the elasticity. It can be used as a basis for composite control schemes. The joint stiffness of the BioRob manipulator, however, is too low for this approach. Instead, an inverse model is used for the control law [3].

The calculation of computed torque for a given trajectory $\mathbf{q}_d(t)$ is more difficult with elastic joints, because the desired motor trajectories $\boldsymbol{\theta}_d(t)$ are not known. With the desired link trajectory we calculate the link equilibrium positions from the joint elasticity equation (9):

$$\hat{\mathbf{q}}_d = \mathbf{k}_e^{-1}(\tau_{e,d}) + \mathbf{q}_d . \quad (12)$$

Applying the rigid link dynamics (10) and transforming the equilibrium positions in motor positions with (2) yields the desired motor trajectory that produces the given desired joint trajectory $\mathbf{q}_d(t)$:

$$\boldsymbol{\theta}_d = \mathbf{k}_e^{-1}\left(\mathbf{M}(\mathbf{q}_d)\ddot{\mathbf{q}}_d + \mathbf{C}(\mathbf{q}_d, \dot{\mathbf{q}}_d)\dot{\mathbf{q}}_d + \mathbf{D}\dot{\mathbf{q}}_d + \mathbf{g}(\mathbf{q}_d)\right) + \mathbf{q}_d + \boldsymbol{\alpha}_c(\mathbf{q}_d) \quad (13)$$

The desired motor torques $\boldsymbol{\tau}_{m,d}$ can then be calculated through Equation (11):

$$\boldsymbol{\tau}_{m,d} = \mathbf{I}_m \ddot{\boldsymbol{\theta}}_d + \mathbf{D}_m \dot{\boldsymbol{\theta}}_d + \boldsymbol{\tau}_{e,d} , \quad (14)$$

requiring the second derivative of (13), however. Feedforward laws for the linear joint elasticity case were presented in [3]. This approach would demand a linearization of Equation (6), which would be inaccurate. Therefore, the desired motor trajectory is used for control instead of the desired motor torque.

5.2 Control

The desired link trajectory \mathbf{q}_d and the desired motor trajectory θ_d (13) can be used for a controller as shown in Figure 7. Each elastic joint can be described by an ordinary differential equation of order four, which is the length of the complete state vector. The BioRob arm sensor system measures motor θ_i and joint q_i positions, so that following state variables are chosen: $(q \dot{q} \theta \dot{\theta})$. This representation has the advantage that only the first derivative is needed, which can be obtained by numerical differentiation.

A simplified control structure can be obtained when using only the steady state torque in (13):

$$\theta_d = k_e^{-1}(g(\mathbf{q}_d)) + \mathbf{q}_d + \alpha_c(\mathbf{q}_d). \quad (15)$$

This controller structure uses a global nonlinear calculation of the motor set-point, which linearizes each joint around the current desired position, assuming a sufficiently accurate model. Each joint can than be controlled by a linear controller for all states to receive damped and exact steady state behavior.

In addition to the controller, we also use an approximation of a global gravitational compensation $g(\mathbf{q})$, which is exact in steady state. We assume a motor with controlled torque output (3).

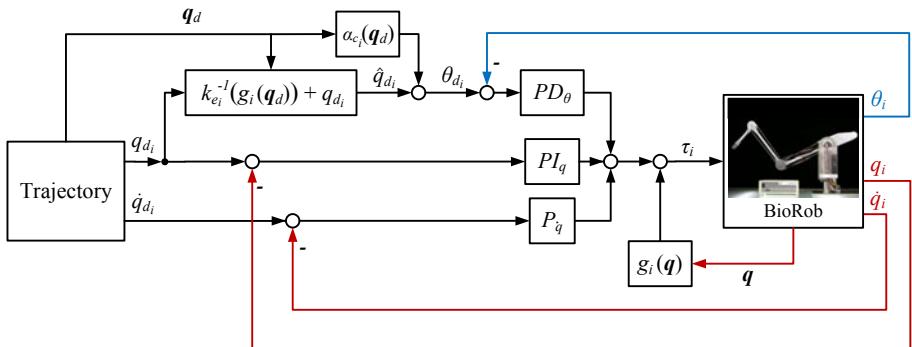
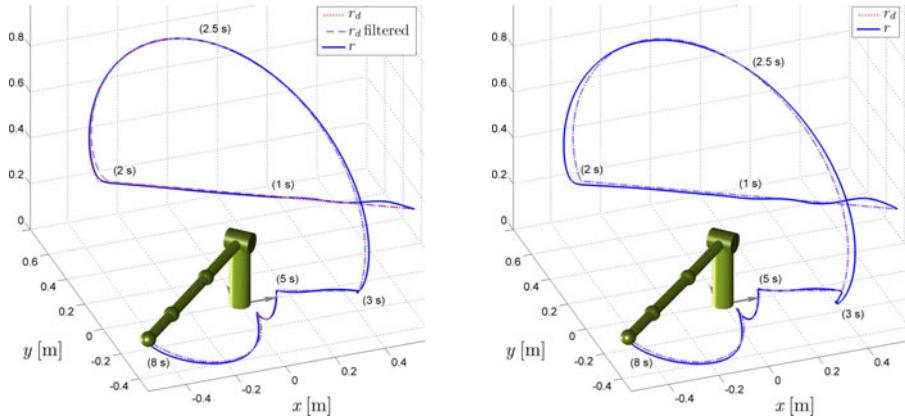


Fig. 7. Control structure for joint i

6 Simulation Experiments

For the evaluation of the presented controller, a simulation model consisting of the models presented in the former chapters was implemented. The values of the model parameters were obtained by identification and optimization with measurements of the real hardware. The joint stiffness ranges from 4 to 20 Nm/rad, the motors are limited to 10 Nm.

The trajectory used for evaluation of the controller is oriented toward a typical application (Figure 10), but also consists of segments of linear motion in Cartesian space. The desired joint trajectories \mathbf{q}_d were piecewise generated by cubic



(a) Low pass filtering of the joint trajectory with time constants $T_1 = T_2 = 0.05$ s.

(b) Without trajectory filtering.

Fig. 8. Effect of the joint trajectory low pass filtering

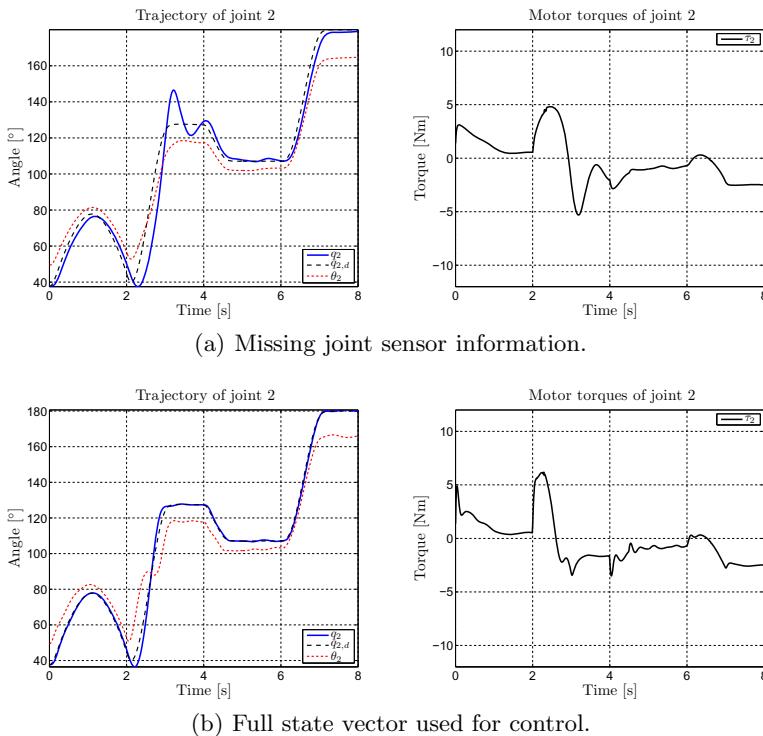


Fig. 9. Comparison of the performance of the full state feedback controller and a reduced controller only using motor sensor information

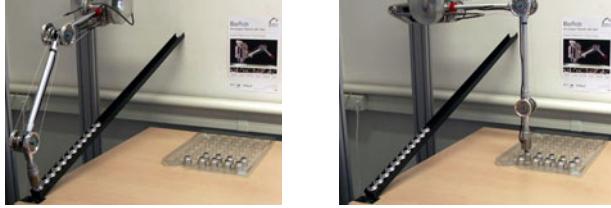
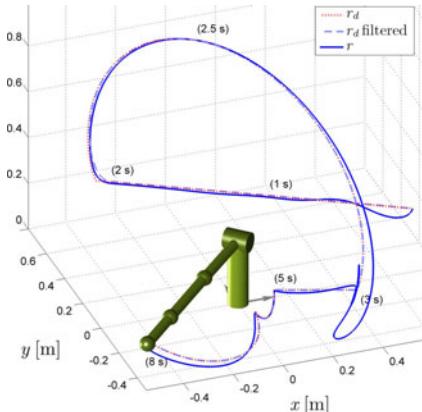


Fig. 10. Pick and place application with the BioRob robot arm [2]

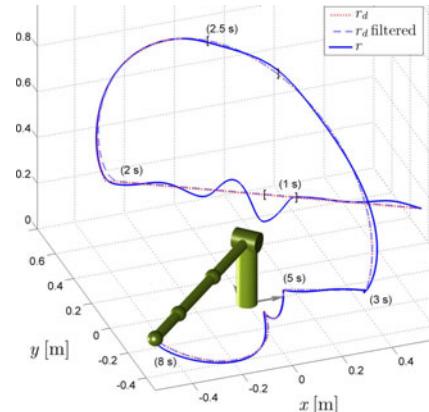
interpolation of joint trajectories calculated by inverse kinematics. These trajectories were then low pass filtered. Figure 8 shows the increased performance when using slight filtering with two time constants. As can be seen, critical trajectory points are smoothed ($t = 2$ s), whereas stationary points are preserved ($t = 3$ s).

Figure 9 shows how the joint sensor information is used for damping the oscillations caused by the joint elasticity. As can be seen in Figure 9(a), good steady state accuracy can be obtained with an accurate steady state model even if no joint information is available.

To evaluate the robustness of the controller design with respect to modeling errors and external disturbances, model parameters of the simulation model were altered (Fig. 11(a)) and external forces (Fig. 11(b)) were applied to the robot arm. The additional weight on the end effector effectively doubles its weight. The controller is not able to prevent overshoot at high accelerations, as can be



(a) Parameter changes: fourth link weight (doubled), joint and motor friction (doubled).



(b) External forces (marked with square brackets): 5 Nm applied on the second joint from 0.9 to 1.1 s and the on first joint from 2.5 to 2.6 s.

Fig. 11. Effect of modeling errors and disturbances

seen at $t = 3$ s, but the overall performance is still good and robust. This is also the case for external disturbing forces.

For videos of the implementation of the pick and place application (Figure 10) with the BioRob arm see [2].

7 Discussion

The presented controller is based on the steady state model. The remaining dynamics are seen as disturbances on joint level, which are compensated for by a linear full state feedback controller. This design limits the end effector loads and acceleration. With high end effector loads and at high accelerations, the controller is not able to completely damp the oscillations caused by the joint elasticity. The main advantages of the approach are low requirements on the accuracy of the dynamics model parameters. The steady state parameters that are used can be identified with substantial lower effort and higher accuracy than the dynamics parameters.

8 Conclusion and Outlook

This paper presented the kinematics and dynamics models and a position tracking control scheme for a series elastic joint robot arm with cable and pulley actuation. We pointed out how the desired motor trajectory can be calculated for a given joint trajectory and how it can be used for damping and tracking control. The performance of the controller design was evaluated in simulation. Future research concentrates on control structures for fast feedforward movements, requiring a model-based extension of the presented controller.

Acknowledgments. The research presented in this paper was supported by the German Federal Ministry of Education and Research BMBF under grant 01 RB 0908 A. The authors would like to thank A. Karguth and C. Trommer from TETRA GmbH, Ilmenau, Germany for the hardware design and photographs (Fig. 11 and 12) of the BioRob robot arm.

References

1. Albu-Schäffer, A., Ott, C., Hirzinger, G.: A unified passivity-based control framework for position, torque and impedance control of flexible joint robots. *International Journal of Robotics Research* 26(1), 23–39 (2007)
2. BioRob project website, <http://www.biorob.de>
3. De Luca, A.: Feedforward/feedback laws for the control of flexible robots. In: Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA), vol. 1, pp. 233–240 (2000)
4. De Luca, A., Book, W.: Robots with Flexible Elements. In: Springer Handbook of Robotics, pp. 287–319. Springer, Heidelberg (2008)

5. De Luca, A., Siciliano, B., Zollo, L.: PD control with on-line gravity compensation for robots with elastic joints: Theory and experiments. *Automatica* 41(10), 1809–1819 (2005)
6. Dwivedy, S.K., Eberhard, P.: Dynamic analysis of flexible manipulators, a literature review. *Mechanism and Machine Theory* 41(7), 749–777 (2006)
7. Höpler, R., Thümmel, M.: Symbolic computation of the inverse dynamics of elastic joint robots. In: Proc. IEEE Intl. Conf. on Robotics and Automation, ICRA (2004)
8. Klute, G.K., Czerniecki, J.M., Hannaford, B.: Artificial muscles: Actuators for biorobotic systems. *International Journal of Robotics Research* 21(4), 295–309 (2002)
9. Lens, T., Kunz, J., Trommer, C., Karguth, A., von Stryk, O.: Biorob-arm: A quickly deployable and intrinsically safe, light-weight robot arm for service robotics applications. In: Proc. 41st International Symposium on Robotics (ISR)/6th German Conference on Robotics, ROBOTIK (2010)
10. Marino, R., Nicosia, S.: On the feedback control of industrial robots with elastic joints: A singular perturbation approach. Tech. Rep. R-84.01, University of Rome (1984)
11. Möhl, B.: Bionic robot arm with compliant actuators. In: Proc. Sensor Fusion and Decentralized Control in Robotic Systems III (SPIE), pp. 82–85 (2000)
12. Murphy, S., Wen, J., Saridis, G.: Simulation and analysis of flexibly jointed manipulators. In: Proc. 29th IEEE Conf. on Decision and Control, vol. 2, pp. 545–550 (1990)
13. Pratt, G., Williamson, M.: Series elastic actuators. In: IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, vol. 1, p. 399 (1995)
14. Spong, M.W.: Modeling and control of elastic joint robots. *Journal of Dynamic Systems, Measurement, and Control* 109(4), 310–318 (1987)
15. Tomei, P.: An observer for flexible joint robots. *IEEE Transactions on Automatic Control* 35(6), 739–743 (1990)
16. Tomei, P.: A simple pd controller for robots with elastic joints. *IEEE Transactions on Automatic Control* 36(10), 1208–1213 (1991)
17. Van Ham, R., Sugar, T., Vanderborght, B., Hollander, K., Lefeber, D.: Compliant actuator designs. *IEEE Robotics & Automation Magazine* 16(3), 81–94 (2009)
18. Wolf, S., Hirzinger, G.: A new variable stiffness design: Matching requirements of the next robot generation. In: Proc. IEEE Intl. Conf. on Robotics and Automation (ICRA), pp. 1741–1746 (2008)
19. Zinn, M., Khatib, O., Roth, B., Salisbury, J.: A new actuation approach for human-friendly robot design. *International Journal of Robotics Research* 23(4/5), 379–398 (2004)

Static Balance for Rescue Robot Navigation: Discretizing Rotational Motion within Random Step Environment

Evgeni Magid and Takashi Tsubouchi

ROBOKEN - Intelligent Robot Laboratory

University of Tsukuba, Japan

{evgeni, tsubo}@roboken.esys.tsukuba.ac.jp

Abstract. The goal of rescue robotics is to extend the capabilities and to increase the safety of human rescuers. During a rescue mission a mobile agent enters a rescue site and it is manipulated by a human operator from a safe place. Without seeing the robot and the environment, a decision on the path selection is very complicated. Our long term research goal is to provide a kind of automatic “pilot system” to propose an operator a good direction to traverse the environment, taking into an account the robot’s static and dynamic properties.

To find a good path we need a special path search algorithm on debris and a proper definition of a search tree, which can ensure smooth exploration. In this paper we present our results in estimation of the transition possibilities between two consecutive states, connected with a rotation step. Exhaustive simulations were used to analyze data and to remove unsuitable directions of the search from the search tree. Combining together the results of this paper and our previous results on a translation step and estimation of losing balance on purpose within Random Step Environment, we can now build a search tree and continue toward the path planning process.

1 Introduction

The long standing target of autonomous robotics is to help weak and vulnerable humans dealing with the situations which are beyond our natural abilities: exploring other planets, volcano craters and dangerous old mine tunnels deep under the earth surface, working in a high pressure and poisonous environments of nuclear and chemical plants, substituting human teams in scouting, clear mines and rescue operations. Extending the capabilities and increasing the safety of human teams during search and rescue operations, when victims are often buried in unreachable locations, is the main application of rescue robotics. Instead of sending human rescues to a dangerous site of heavily damaged buildings and underground areas, a rescue robot is deployed there for initial exploration purposes. Then a human tele-operator can operate the robot from a safe place, thus avoiding unnecessary risk of human rescue team casualties from secondary disaster. The system consists of a remote operation station (fig. 1(a)) and a mobile robot platform (fig. 1(b)), connected with a wireless LAN.

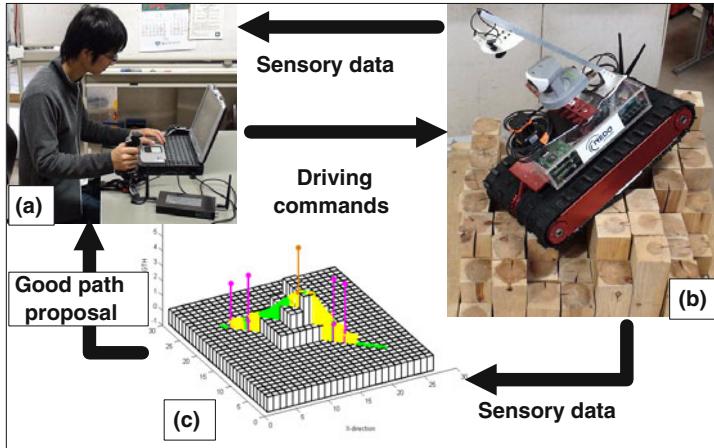


Fig. 1. Standard framework includes operator (a) and rescue robot on RSE (b); we propose to enhance the standard scheme with our “pilot system” (c)

At the moment rescue robots are mainly operated manually by human operators. The operator usually has difficulties to take the decisions based on the limited visual sensory data and choosing a safe robot’s path turns into a complicated time consuming process. Staying inside a crawler-type vehicle, the human operator feel the inclination of the slope and the decision on the traversability of the path becomes easier. Unfortunately, the off-site operator cannot use natural biological sensors and judges on the next move only on the base of the available visual information and his/her previous experience. Since many optional paths may connect start and goal locations, it is hard to make an objective decision on a fairly good and safe path. To propose the operator a good path or several options from the current to the goal location is our main research goal. An automatic “pilot system” will calculate the path with regard to the robot’s static and dynamic properties. Next, the “pilot system” by means of GUI will suggest several options of a good path to the operator (fig. 1(c)). Finally the operator will decide which path to apply it in a real scenario driving the robot.

To find a good path we implement a special path search algorithm on debris. A dangerous and unstable debris site requires the algorithm to keep the robot maximally stable at every step of its path. The real state space of the search is extremely huge. We have to discretize robot’s motion and the state space in order to decrease the number of search directions and make the search feasible. A search algorithm utilizes a search tree [2]; for our problem dynamically created search tree can not be explicitly presented as a skeleton. Using as input arguments $Args$ the robot’s current configuration and a local environment map, we present the search tree with a function $F(Args) = Res$. Its output Res , a set of accessible within one step configurations, will guide the tree search.

In this paper we present the particular part of function F responsible for a rotational step. We estimate the transition possibilities of a crawler type vehicle

between two consecutive states, connected with a rotation step within Random Step Environment(RSE) - a simulated debris environment, proposed by NIST [4]. Previously we presented results on a translation step between two postures [7] and estimation of losing balance on purpose [8] within RSE. This paper is completing our research of the transition possibilities between two consecutive states and affords us to build a search tree and continue toward a path planning process. Our theoretical results were confirmed with exhaustive simulations, removing all unsuitable directions of the search from the search tree. Currently we are in discussing if the verification of the proposed solution with a large set of experiments within different RSE setups is necessary.

2 Static Balance Estimation of a Posture

Urban search and rescue related research assumes that a rescue robot operates in a dangerous unstable debris site of a heavily damaged by some disaster building. One of the most popular simulations of debris environment is a so-called Random Step Environment(RSE) or Stepfield, proposed by NIST - The National Institute of Standards and Technology [4]. RSE is a set of random steps with equal width and length, but different height(fig.1(b)). A big variety of optional assemblies, similar to real debris behavior, easy setup and storage made RSE attractive test arenas for evaluation of the urban search and rescue robots performance [9]. RSE is an obligatory test arena for each RoboCup Rescue competition, where at least several RSEs of different difficulty levels are exploited.

Each rescue group uses its own RSE setup, which is more or less similar to an official RoboCup Rescue version. We assemble our RSE from 85mm × 85mm size wooden blocks of 0, 90, 180 or 270 mm height; 0mm height corresponds to the ground level around the RSE-patch. We assume a simple tractor-like crawler non-reconfigurable robot with the centroidal location of robot's center of mass (CM). This configuration corresponds to the main body of "KENAF" robot(fig.1(b)), consisting of two large tracks with a small gap in between. Table 1 presents "KENAF" specifications - without sensors, front and back pairs of arms - used in experiments and by the simulation "pilot system".

Table 1. Specifications of "KENAF" in basic configuration

Parameter	Measurement
Maximal inclination	
dynamic	60 deg
static	80 deg
Main body length	584 mm
Main body width	336 mm
Track width	150 mm
Height	270 mm
Weight	17.8 kg

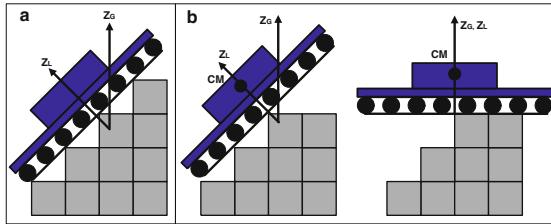


Fig. 2. (a) Green state (b) Orange state: O_1 (left) and O_2 (right)

2.1 Static Stability

The debris site is dangerous and unstable and the main goal of the algorithm is to keep the robot maximally stable at every step of its path in the specific configuration without slipping or turning upside-down. To ensure a stable behavior of the robot while traversing RSE, a continuous satisfaction of static and dynamic constraints [10] is essential. To satisfy a static stability requirement, which is a minimal necessary stability condition [6], the robot's center of mass (CM) must lie above the **support polygon** [1]. For a crawler type vehicle we define a support polygon as a 2D convex hull of all real contact points of the robot's crawlers with RSE at each given posture.

2.2 Static Balance Estimation

In this section we briefly describe six static balance posture types and assign them color labels, presented in details in [6, 7]. **Red state** is statically unstable posture that immediately results in robot's turning upside down if it tries to climb to an impossible steepness. **Magenta state** describes climbing up or sliding down the vertical slope of the environment (fig. 3). **Cyan state** is assigned for a robot's jumping down situation during a rotation motion step (if the posture will change with just one step from fig. 3(d) to 3(a)). Statically stable postures (fig. 2(a)) are described with a high quality balance **Green state** and an average quality balance **Yellow state**; Normalized Energy Stability Margin (NESM) [3] is applied to distinguish those two states. To afford the robot losing the balance on purpose we define **Orange state**. This posture does not result in robot's turning upside down, but do not guarantee a single stable posture since two optional postures exist (fig. 2(b)). Such behavior is necessary when, for example, the robot traverses a barrier, climbing up and going down with losing its balance twice on top. Orange state consists of two sub-states : first sub-state O_1 before the robot loses its balance and second O_2 , which occurs after the robot loses its initial balance, changes its posture discontinuously at that point and obtains a balance again in a different body orientation. Orange state is a very gentle state and should be used with a special care in translation motion step [7, 8] and completely forbidden for a rotational step as we will explain in section 4. Further we denote by R-posture a posture which static balance corresponds to a red state type, M-posture for magenta etc.

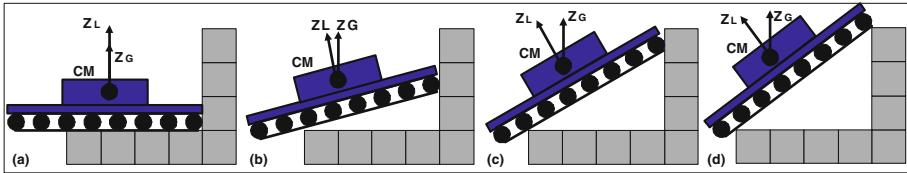


Fig. 3. Magenta state - at climbing up mode robot moves from (a) to (d); at sliding down - from (d) to (a)

2.3 Describing a Posture

To characterize robot's postures qualitatively we use the coloring of the state. To decide on legal transitions between two successive states, we use a combination of 6 following variables [7]. **Steepness** θ_X - the angle, showing the steepness of the RSE at the current robot configuration and rotational moment around robot's transversal Y_L -axis. **Moment** θ_Y - the angle, indicating current rotational moment around robot's lateral X_L -axis. **Contact points quality (CPQ)** depends on the angle θ_{CPQ} between the robot's crawlers and the edges of the RSE cells and affects the robot's ability of climbing the obstacles, losing balance on purpose and going down safely. **Inclination** is the *steepness* angle θ_X sign; with respect to this parameter we specify three groups of posture sets: a climbing up the steps of the RSE posture $G_{U_{inc}}$, a going down posture $G_{D_{inc}}$ and a neutral inclination posture $G_{Z_{inc}}$. **M-sign** is the *moment* angle θ_Y sign; similarly group $G_{P_{MS}}$ is for all postures with positive M-sign, $G_{N_{MS}}$ with negative and $G_{Z_{MS}}$ with neutral postures. Finally, **NESM-stability** - warns about the probability of the robot's turning upside down due to the CM being too close to one of the edges of the support polygon [3].

Inclination and *M-sign* signal about discretization problems, pointing on the missed posture between two successive postures; 4 other variables are emphasized for the experimental work. Combining inclination and M-sign, we specify a neutral Z-posture - a posture with robot's body being parallel to a horizontal patch of RSE : $G_{Z_{inc}} \cap G_{Z_{MS}}$. Further we denote each posture as Col(Inc) where Col is the color, Inc is the *inclination*. For example, $G(Z)$ means a green neutral Z-posture and $G(U)$ means G-posture with U_{inc} .

3 Discretizing Search Space and Building a Search Tree

Search tree function $F(\text{Args})$ is to decide on possible next steps of the robot from a given current location and orientation. In a standard 2D navigation each cell of the state space is "free" or "busy" (obstacle, another moving agent etc.) and a transition between two free cells is always legal [5]. In our case we have "possible" (statically stable) and "impossible" (statically unstable) postures. But even in the case of two adjacent "possible" postures the transition between them is not always possible. To decrease the number of search directions we discretize robot's motion and the state space before the search. Starting from 3 levels of

search space discretization for XY-coordinates of the environment, we concluded that discretizing each 85x85mm cell of RSE into 5x5 cells of the internal robot map with the cell size of 17x17mm is the best choice for our problem solution [7].

KENAF supports two types of motion: translation and rotation. Thus at each node of the search tree the search algorithm opens the 3-neighborhood of the node - go straight or turn left/right - and to proceed the search in the most promising direction. All impossible search directions, different for rotation and translation steps, are immediately cut off from the search tree. **Translation step** is defined as a one cell length step forward in the direction of robot local frame's axis X_L . **Rotation step** is a 5 degrees change in robot orientation θ , rotating clockwise(right) or counter clockwise(left). In this paper we present our results in estimation of the transition possibilities between two consecutive states, connected with a rotation step.

4 Rotation Step Transitions

In practice rotational motion within the simulation and in the real world differ a lot. Quality of the surface, number of real contact points, robot's speed, accumulated error in control system, communication delay etc. could result in a high level of imprecision. To ensure that the simulated path could be repeated by the operator in a real scenario, we forbid any dangerous and unpredictable transitions between two states. Appearance of the O-posture is also a hardly predictable situation, so we forbid it for rotational motion and accept losing balance on purpose only at translational step case. To define a legal rotational transition between two stable postures, we created a set of theoretical hypotheses, based on our experimental experience. Exhaustive simulations for environment existence in MATLAB gave a valuable feedback for our theory and generated branch cutting conditions for the path search algorithm (fig. 4).

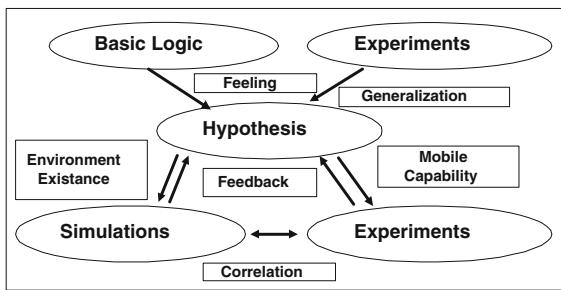


Fig. 4. Theory, simulations and experiments

4.1 Hypotheses

Similarly to a translation step case, described in details in [7], we carried out a set of rotational motion experiments with KENAF robot in several RSEs and

created a set of rules on the rotations (RR) between two successive postures. Some rules are identical for both translation and rotational step. Rules include trivial statements, definitions and assumptions:

(RR1) Neutral posture (Z-posture) is defined if robot's body is parallel to the ground level: $G_{Z_{inc}} \cap G_{Z_{MS}}$

(RR2) Perfect rotation preserves a neutral posture and a vertical coordinate Z_{CM} of robot's CM.

(RR3) The only way to climb up or slide down a vertical slope of RSE is to apply a so-called *M-chain* - a sequence of M postures between two stable postures (start G_{start} shown in fig 3(a) and end G_{end} shown in fig 3(d) postures of the *M-chain*). *M-chain* cannot contain any non-M posture.

(RR4) C-posture appears due to discretization problem. With the infinitely small discretization level it would be substituted by a missed *M-chain*.

(RR5) M-posture has no real θ_X , θ_Y and NESM parameter data, but only an approximation. Also it has no own inclination and M-sign data.

(RR6) Loosing balance on purpose through O-posture during rotational step results into unpredictable next posture of the robot. In a worst case it results into robot's turning upside down.

(RR7) Change of inclination between two postures can occur only through O-posture and its O_1 or O_2 is a Z-posture.

(RR8) Change of M-sign can occur only through O-posture.

(RR9) Any significant change in 3D orientation of the robot may result into robot's turning upside down.¹

4.2 Successive Posture Groups

We divided all possible pairs of postures, connected with a rotational step, into groups. Each group contains theoretically possible or impossible sequence; some of possible sequences are strongly recommended, while some are undesirable. For the rotation case a legal set of colors for each posture is $\{G, Y, O, M, C\}$ and we are supposed to obtain 25 groups of pairs at most. To decrease the number of groups we treat at the simulation level G and Y as a same G color.

Forbidden Sequences

In addition to appearances of R-postures, any pair, containing O-posture, is forbidden with regard to (RR6). If during a 5-degree rotation step robot will lose its balance at O_1 in order to obtain it again at O_2 , real robot's behavior on RSE becomes unpredictable and loosing balance on purpose may result in turning upside down. This implies not only an explicit appearance of O-posture, but also a missed O-postures mentioned in (RR7) and (RR8). Also we forbid any rotational step which results in robot's climbing or jumping up situation. Fig 5 demonstrates the example of jumping up between two successive postures, connected with a 5 degree rotational step: current posture(left) and next posture(right). In most cases it is enough to check the vertical change in CM-coordinate Z_{CM} to

¹ We forbid any change exceeding 4 degrees. For the details see section 4.2.

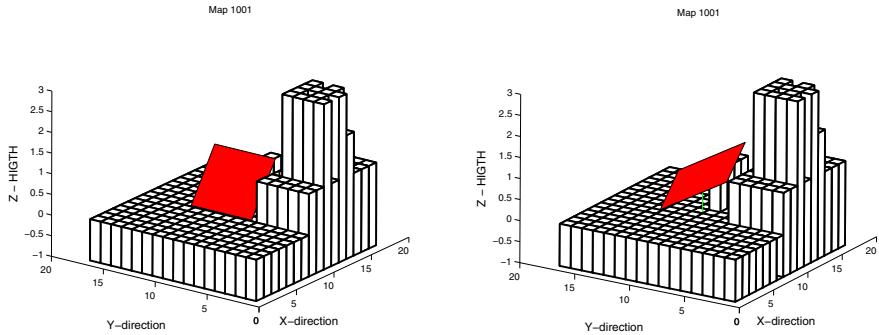


Fig. 5. Jump up case: current(left) and next(right) postures

detect a jump up: any case exceeding $\varepsilon \doteq 4.5\text{mm}$ used to take into an account accumulated numerical error, is physically impossible. No jump up or down between two neutral postures $G(Z) \rightarrow G(Z)$ and no climbing up between G_{start} and G_{end} ends of M-chain can occur. Any jumping down, except C-case, is forbidden as well. C-case permits a jump down within physically acceptable vertical jump of the robot. There is also a number of other postures, which we define as forbidden postures and they will appear later in this section.

Undesirable Sequences

Similar to O-posture case, M-chains and single M-postures are not predictable enough even for the case of sliding down a vertical slope of RSE. Main difficulty is that since we cannot detect precisely where in $[0,5]$ degrees rotational interval of the 5-degrees rotation the robot started to slide down (a start of M-chain), we cannot predict the result of the process. The same explanation, with regard to (RR4), makes any sequence, containing at C-postures, undesirable. Being physically possible, these two postures do not result in robot's turning upside down and do not destroy robot's sensors, the most vulnerable part of the robot, while sliding down. Yet we would like to abstain from their appearance in the path because we can not predict exactly what will be the next posture of the robot. The only case when we permit using undesirable sequences during the path search process is when no other better choice is available. However, if choosing an undesirable sequence, upon arrival to a newly obtained after sliding down posture, we will have to recollect the sensory data about the local RSE patch, localize the robot and restart the path search process starting this new posture.

Perfect Sequence

We distinguish 3 types of possible sequence: perfect, good and fair. Perfect case is a rotation on a flat pattern of RSE with current and next both neutral postures and constant location of CM in vertical direction Z_{CM} , described by (RR2).

Good and Fair Sequences

All other $G \rightarrow G$ cases form good and fair cases group or are added to undesirable and impossible cases groups. To make this more precise division of the cases we apply parameters, defined in section 2.3. For all non perfect $G \rightarrow G$ transitions

Table 2. G → G posture types distribution

GG1	GG2	GG3	GG4	GG5	GG6	GG7	GG8
45%	0.02%	0.02%	0.21%	0.01%	0.15%	2.7%	51.87%

we apply a special contact points checking procedure, which compares expected contact points vertical locations between current and next postures. We forbid the transition if at least one of the expected contact points jumped up.

Combining different inclinations we obtained 8 groups of G → G postures:

$$\begin{array}{lll} (\text{GG1}) \text{ G(D)} \rightarrow \text{G(D)} & (\text{GG2}) \text{ G(D)} \rightarrow \text{G(Z)} & (\text{GG3}) \text{ G(D)} \rightarrow \text{G(U)} \\ (\text{GG4}) \text{ G(Z)} \rightarrow \text{G(D)} & & (\text{GG5}) \text{ G(Z)} \rightarrow \text{G(U)} \\ (\text{GG6}) \text{ G(U)} \rightarrow \text{G(D)} & (\text{GG7}) \text{ G(U)} \rightarrow \text{G(Z)} & (\text{GG8}) \text{ G(U)} \rightarrow \text{G(U)} \end{array}$$

We analized the statistics² for the G → G postures groups appearance, presented in table 2. Subgroups GG3 and GG6 are forbidden with regard to (RR6) and (RR7). GG1 and GG8 cases, being a majority of the groups (45% and 51.87% respectively), were expanded further. Suspicious groups, forming all together 2.94% of the G → G cases, were excluded from further expansion and added to forbidden cases. First of all, if we want to use GG1 and GG8 cases for the path planning, we must remove any dangerous possibilities. Forbidding jump up of $\varepsilon = 4.5\text{mm}$ is not enough; to increase the level of safety, we forbid *any* jump up exceeding zero. Of course, we aware that in many cases we forbid possible cases obtained due to a accumulated numerical error, but we can not take a chance to accept a real jump up. Four no jump up (ok) and jump up (j) sequences have almost the same appearance about 25% of the cases(table 3). Again, we point at the important issue: we take no risk of getting stuck in the real world, accepting suspicious cases in the pilot system at the simulation level.

Table 3. GG1 and GG8 : no jump up (ok) and jump up (j) sequences distribution

GG1(ok)	GG1(j)	GG8(ok)	GG8(j)
25.44%	21.01%	28.37%	25.18%

We are interested to apply the rotations which preserve robot's body 3D orientation and avoid the cases of extreme orientation changes, which in the worst case result into robot's turning upside down. As a measure of this change we use a support polygon's plane normal, which coincides with a Z-axis of a local coordinate frame of the robot Z_L . The change in normal orientation between two successive postures as a measure of orientation change:

$$\Omega = |\widehat{Z_G Z_L}(\text{curr}) - \widehat{Z_G Z_L}(\text{next})| \quad (1)$$

where $\widehat{Z_G Z_L}$ is an angle between the Z-axis of the global RSE frame Z_G and $Z_L(\text{curr})$ and $Z_L(\text{next})$ correspond to the Z-axis of a local frame at current and

² Simulation setup for statistics is described in details in section 5.

next postures respectively. We distinguish 5 distinct subgroups for Ω parameter for each of 2 cases GG1(ok) and GG8(ok):

- (a) $\Omega \in [0,1)$
- (b) $\Omega \in [1,2)$
- (c) $\Omega \in [2,3)$
- (d) $\Omega \in [3,4)$
- (e) $\Omega \in [4,90)$

Voting for those 10 subgroups of GG1(ok) and GG8(ok) sequences showed the following distribution (table 4). Based on the statistics we divide GG1(ok) and GG8(ok) sequences into 3 main groups:

- GG1(a) and GG8(a) form good sequences - they appear in 86.01% cases.
- GG1(b,c,d) and GG8(b,c,d) form fair sequences, where the quality decreases as Ω increases; all together 13.4%.
- GG1(e) and GG8(e) are forbidden sequences with $\Omega > 4$ degrees 0.58%. Even though some of them may be still possible, their removal from the path search procedure will not seriously influence the quality of the path as statistics shows. Currently we are discussing if we need to increase the lower bound of the interval through verification experiments.

Table 4. GG1(ok) and GG8(ok) sequences percentage with regard to parameter Ω

GG1(a)	GG1(b)	GG1(c)	GG1(d)	GG1(e)	GG8(a)	GG8(b)	GG8(c)	GG8(d)	GG8(e)
40.64	4.01	1.21	1.17	0.26	45.38	4.8	1.09	1.13	0.31

4.3 Summary

Summarizing, we group all rotational step cases into 5 clusters:

1. Forbidden - R and O-postures, all jumping and climbing up cases, groups GG2 GG7, suspicious to a jump up subgroups GG1(j) and GG8(j) and big change in 3D orientation subgroups GG1(e) and GG8(e)
2. Undesirable - Sliding down M-chains and jumping down C-postures; they could be included as a part of the path if no better option exists.
3. Perfect - $G(Z) \rightarrow G(Z)$ transition with no vertical change in CM location
4. Good - subgroups GG1(a) and GG8(a), where the change in 3D orientation does not exceed 1 degree
5. Fair - subgroups GG1(b,c,d) and GG8(b,c,d), where the change in 3D orientation is between 1 and 4 degrees

5 Simulations

Only an experimental proof can guarantee that a good theoretical hypothesis will work correctly in a real world as well. A huge number of various cases can happen in a completely random RSE and implementation of such amount of experiments is physically impossible. Impossible in the real world pairs of postures are also

impossible within the simulation. As far as the reverse statement is not true, the simulator cannot replace the experiments, but assists to structure the data and remove the impossible types of sequences, saving time and efforts. Successive transition patterns will be integrated in the search algorithm as a part of neighbor opening and branch cutting function $F(\text{Args}) = \text{Res}$.

For the exhaustive check we created a huge 61x61 cell map (fig.6) which includes all typical obstacles, usually appearing in the RSE: horizontal and diagonal barriers, pairs of parallel barriers, valleys, traversable and non-traversable pikes and holes. All possible pairs of postures connected with a rotation step were proceeded with voting for each group, described in section 4. For a first posture of the pair we placed robot's CM at every node of the grid; next posture was calculated as a 5 degree increment of the robot's heading direction θ , rotating right. The simulation included 86 initial robot orientations $\theta \in \{0, \frac{\pi}{180}, \frac{2\pi}{180}, \dots, \frac{84\pi}{180}, \frac{85\pi}{180}\}$. We assume a symmetrical behavior for rotating left/right and for other 3 quarters of the 360-degree rotation range. In addition to pointing at the impossible (empty) cases the simulation reveals the rare cases and the most often cases.

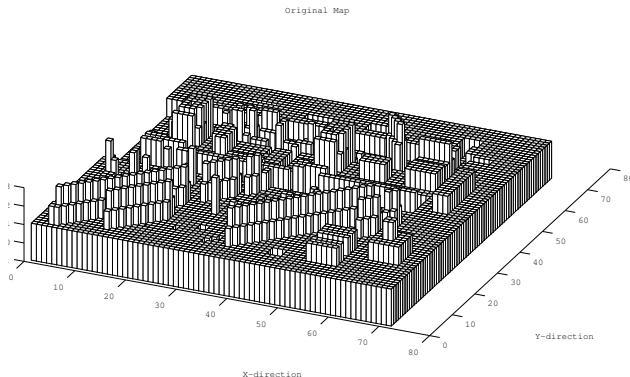


Fig. 6. A 61x61 cells RSE, covering all main types of the environment obstacles

The total number of posture pairs was more than 6 millions. The results of the simulation are summarized in table 5. Among them O-posture appeared in 1.58% of the pairs explicitly and in 0.06% as a missed O-posture. Impossible pairs with a significant jumping up/down behavior form together 1.12% cases. Excluded pairs, which were later also added to forbidden cases, are 0.94%. Together with all additional forbidden posture sequences, forbidden group reaches 23.59%. Un-desirable M and C-postures with sliding down behavior are only 2.36%. Perfect rotations are the most significant amount of the cases, 56.64%; good rotations score 15.71%, while fair rotations sum up to 0.76% only. This statistics matches well with our expectations based on the operational experience - rotational step is also a hard challenge for the operator and in the simulation only 73.11% of the pairs were detected to be more or less suitable as the path points; 2.36% of the pairs are hardly suitable and the rest 24.53% are completely useless.

Table 5. Main, undesirable and forbidden transitions

Perfect	Good	Fair	Undesirable	Forbidden	Excluded
56.64	15.71	0.76	2.36	23.59	0.94

6 Conclusions and Future Work

To provide an assistant “pilot system” for an operator of a rescue robot, which will be able to propose the operator a good path from the current to the target location, is our main long term research goal. After obtaining data from the environment a robot creates an internal world model and the path selection within the internal model should be done. Since usually there exist more than just a single path, a good instrument to evaluate the quality of each path is important for the path search algorithm.

The search algorithm within the graph requires a proper definition of neighboring states to ensure smooth exploration of the search tree. In this paper we presented our results in estimation of the transition possibilities between two consecutive states, connected with a rotation step. It is our final step toward a definition of a search tree neighborhood function $F(Args) = Res$, where arguments $Args$ are the robot’s current configuration and the environment and output Res is a set of accessible within one step configurations. Our theoretical basis for function F was confirmed with exhaustive simulations, removing unsuitable search directions. Next we consider to verify our results with experiments. Combining together the results of this paper and our previous results on a translation step and estimation of losing balance on purpose within RSE, we can now build a search tree and continue toward the path planning process.

Acknowledgments

This research has been partially supported by NEDO Project for Strategic Development of Advanced Robotics Elemental Technologies, High-Speed Search Robot System in Confined Space.

References

1. Bret, T., Lall, S.: A Fast and Adaptive Test of Static Equilibrium for Legged Robots. In: IEEE ICRA 2006, pp. 1109–1116 (2006)
2. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 2nd edn. The MIT Press/McGraw-Hil (2001)
3. Hirose, S., Tsukagoshi, H., Yoneda, K.: Normalized energy stability margin: generalized stability criterion for walking vehicles. In: Proc. of 1st Int. Conf. On Climbing and Walking Robots, Brussels, pp. 71–76 (1998)
4. Jacoff, A., Messina, E., Evans, J.: Experiences in Deploying Test Arenas for Autonomous Mobile Robots. In: Proc. of the 2001 PerMIS Workshop, in association with IEEE CCA and ISIC, Mexico City, Mexico (2001)

5. Latombe, J.C.: *Robot Motion Planning*. Kluwer Academic Publishers, USA (1991)
6. Magid, E., Ozawa, K., Tsubouchi, T., Koyanagi, E., Yoshida, T.: Rescue Robot Navigation: Static Stability Estimation in Random Step Environment. In: Carpin, S., Noda, I., Pagello, E., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2008. LNCS (LNAI), vol. SIMPAR, pp. 305–316. Springer, Heidelberg (2008)
7. Magid, E., Tsubouchi, T.: Static Balance for Rescue Robot Navigation: Translation Motion Discretization Issue within Random Step Environment. In: Proc. of ICINCO, Madeira, Portugal, pp. 305–316 (2010)
8. Magid, E., Tsubouchi, T., Koyanagi, E., Yoshida, T.: Static Balance for Rescue Robot Navigation: Losing Balance on Purpose within Random Step Environment. In: Proc. of IEEE IROS, Taipei, Taiwan (accepted, 2010)
9. Sheh, R., Kadous, M., Sammut, C., Hengst, B.: Extracting Terrain Features from Range Images for Autonomous Random Stepfield Traversal. In: IEEE Int. Workshop on Safety, Security and Rescue Robotics, Rome, pp. 1–6 (September 2007)
10. Shoval, S.: Stability of a Multi Tracked Robot Traveling Over Steep Slopes. In: IEEE ICRA 2004, vol. 5, pp. 4701–4706 (2004)

Discovery, Localization and Recognition of Smart Objects by a Mobile Robot

Emanuele Menegatti, Matteo Danieletto, Marco Mina, Alberto Pretto,
Andrea Bardella, Andrea Zanella, and Pietro Zanuttigh*

University of Padova, Dep. of Information Engineering (DEI), via Gradenigo 6/B,
35131 Padova, Italy
Tel.: +39 049 827 7840; Fax: +39 049 827 7826
emg@dei.unipd.it

Abstract. This paper presents a robotic system that exploits Wireless Sensor Network (WSN) technologies for implementing an ambient intelligence scenario. We address the problems of robot object discovery, localization, and recognition in a fully distributed way. We propose to embed some memory, some computational power, and some communication capability in the objects, by attaching a WSN mote to each object. We called the union of an object and of a mote, a *smart object*. The robot does not have any information on the number nor on the kind of objects in the environment. The robot discovers the objects through the radio frequency communication provided by the WSN motes. The robot roughly locates the motes by performing a range-only SLAM algorithm based on the RSSI-range measurements. A more precise localization and recognition step is performed by processing images acquired by the camera installed on the robot and matching the descriptors extracted from these images with those transmitted by the motes. Experiments with eight smart objects in a cluttered office environment with many *dummy* objects are reported. The robot was able to correctly locate the motes, to navigate toward them and to correctly recognize the smart objects.

Keywords: Object-Recognition, SIFT, Wireless Sensor Network, mobile robot, ambient intelligence.

1 Introduction

Several research and industrial projects are focusing on service and personal robotics. At the beginning, most of the efforts were driven by the idea of developing robots with large knowledge and very skilled acting in passive environments. In the last years, a new paradigm gained momentum: the idea of simple robots connected among them and connected to distributed sensors and distributed actuators. Depending on the research field, this paradigm is known also as: ubiquitous robotics or networked robotics or ambient intelligence [19][6][8][15]. In this approach intelligent and complex behaviors are achieved through the cooperation of many simple robots and sensors.

* This work was partially supported by the University of Padua under the project "RAMSES2" and partially supported by Fondazione Cassa di Risparmio Padova e Rovigo under the project "A large scale wireless sensor network for pervasive city-wide ambient intelligence".

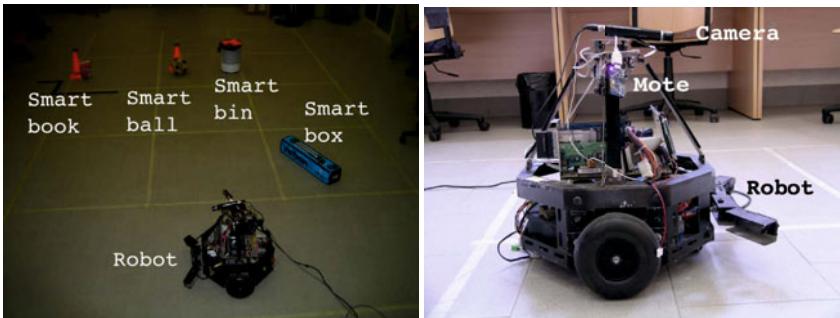


Fig. 1. (Left) An overview of the experiment setup: the robot and the movable smart objects. (Right) The modified Pioneer 2 robot platform. On the top of the robot is a camera with a 90 deg. field of view. On the camera support is latched a TMote Sky to communicate with the Wireless Sensor Network constituted by the smart objects.

In this paper, we focus on the problem of object location and recognition by an autonomous mobile robot. A personal or a service robot will need to interact with and manipulate objects in our houses or in office environments. Thus, it needs to be able to recognize the different objects and locate them in the environment. Several approaches have been proposed for object detection and recognition, most of which perform visual recognition of the object by using a dataset of object-models stored on the robot [32][18]. These approaches, however, require the robot to know the object models beforehand, thus limiting the applicability and the scalability of the solution. We propose a system that overcomes these limits by attaching to the objects small wireless devices (called motes) that provide communication, processing, and sensing capabilities. An object equipped with such a device, hence, acquires some intelligence, becoming a so-called *smart object*. In our solution, the robot does not have any prior knowledge about the shapes of the objects, their positions and their number. Therefore, we can deal with any object from the simplest to the most complex. We can deal with movable (or moving) objects (like the ones in Fig. I or chairs, plants, other robots, etc.). We can deal with a very large variety and multiplicity of objects, because of the distributed approach and, most important, fact that the robot does not need to know the objects beforehand.

Unfortunately, to achieve this flexibility, it is not enough to use localization approaches based on the [signal] strength of the radio signals transmitted by the motes. It is well known that the most common localization techniques based on radio signal strength (RSS) do not provide a precise geographical location of the motes [19]. This is particularly true in indoor environment, because of interference and multiple reflections of the radio signals. Currently, the best algorithm can locate the motes with respect to the robot with a precision of a meter or so (relying only on standard radio signals emitted by standard motes). This is not enough for a reliable localization of the object based only on the object ID. We hence propose to store locally, in the memory of the smart object, the appearance of the object. Thus, the object can transmit its appearance to the robot when the robot requests to interact with that object. The robot can create a map with a coarse localization of the smart objects of interest obtained via radio and move

toward them. When the robot is in the room where the object of interest is located, it can seek for its appearance in its camera image. In this way, the robot can locate much more precisely the object and navigate toward it in order to perform the desired interaction.

The application scenarios for this type of systems include industrial and home applications. For instance, the system may be used in room storages to make it possible for robots to find and retrieve objects on request. The system does not require to store in the robot any information concerning the position or the actual appearance of the objects. Therefore, the robot may be instructed to find an object and the robot will be able to recognize that object in a group of others by using its vision capabilities in conjunction with the information provided by the object itself upon request. Moreover, the proposed approach is scalable also in the number of robots. Indeed, every time a smart object transmits its appearance, more than one robot could listen to the description. Distributing the knowledge in the environment (i.e., in the objects) makes it possible to seamlessly work with one single robot or with a team of cooperating robot. In a home scenario, the same system can be used to tied-up the play-room of kids. At the end of the day, a robot can locate all toys in the room and store them in the right containers. Both the toys and the containers are smart objects in the sense explained above.

One of the principal sources of inspiration for our work was the PEIS-Ecology (i.e., *Ecology of Physically Embedded Intelligent Systems*) approach [15]. In fact, in the PEIS framework a large number of sensors are attached to objects and can transmit to the robot useful information [16] (i.e., the position of objects, the light in the ambient, etc.).

This framework has been already described in rather general terms in our previous seminar works [12] [20]. This paper introduces into our previous framework several new elements that increase the reliability and the flexibility of the system. In this work we presented a unified system for mote localization and object recognition based on the interaction between autonomous mobile robot and WSN Technology. In Section 2 we explain how we implemented the smart objects. In Section 3 we present the new implementation of a range-only SLAM performed with measures of distance obtained from the RSSI of the messages sent by the motes in which the initialization problem of our previous work is solved by introducing a particle filter for implementing a delayed initialization of the Extended Kalman Filter. In Section 4 we present the new object appearance descriptors we adopted to solve the problem of motion blur in the camera image that cased missed matched in the previous implementation of the recognition module. In Section 5 we present experiments in which the robot performs all the steps to discover and approach the smart objects. In particular we show new preliminary experiments on the robustness to motion blur and on the delayed initialization based on the particle filter.

2 Smart Objects

We propose the use of WSN technologies for implementing the smart objects. This allows to relay on hardware and software already developed by the WSN community decreasing the cost of the smart tags to be attached at the objects and exploiting already found solutions for energy saving and efficient communications. The smart objects and

the robot will also constitute a WSN that can perform (and transmit) measures both on the internal state of the objects (like object temperature, level of filling, inclination, weigh, deterioration, etc) and on the state of the environment (temperature, pollution, humidity, and so on). Moreover, smart objects may establish a multi-hop communication network to relay messages to the robot in case direct communication is not available. As written above, each mote attached to the object enables the robot to reliably recognize and locate the object in the environment and the proposed approach is scalable to the number of objects. This is achieved by storing on the motes a compact description of the appearance of the object. We choose a description that can fit in the small memory on board of the motes and that could be created by the manufacturer of the object. This is important in our idea of a totally distributed approach. It is not the robot programmer or robot installer that create the descriptions of the smart objects. It is the object manufacturer that store the object appearance in the mote, so the object manufacturer and the robot provider do not need to interact, but just to follow the same standard.

2.1 The Communication Protocol

The smart objects and the robot are equipped with motes that shares the same communication protocol. In our experiments we used Berkeley TmoteSky which are IEEE 802.15.4 compliant. The communication protocol was implemented in NesC exploiting the TinyOS primitives.

In order to allow the unambiguous identification of each object, each mote is identified by a unique ID, which is also used for labeling each packet transmitted by the mote. The communication protocol is connectionless and unreliable, i.e., no acknowledgement is required to confirm the correct reception of descriptor data packets. This choice is motivated by energy saving and packet traffic management considerations.

3 Object Discovery and Localization

As said, the number of smart objects and their position in the environment is unknown to the robot. The robot needs to discover and locate the objects in the environment. The robot can discover the smart objects when their motes are in the communication range of the robot. At the beginning, when the robot enters the environment, it sends a *HELLO messages* and waits for *HELLO* packets from the motes attached to the objects. The robot reads the ID of the replying objects (i.e., of the smart objects in the communication range) and starts a SLAM procedure (i.e., the Simultaneous Localization of the robot And of the Mapping of the wireless nodes in the environment).

The robot can estimate the distance to nearby nodes of the WSN by measuring the Received Signal Strength Indicator (RSSI) of the received radio messages. Since the RSSI can be calculated directly by most radio transceiver, it allows for a (rough) range estimation without the need for specialized hardware, with an advantage in terms of device's cost and size. Another advantage of RSSI-based ranging is that it does not require the node to be in line of sight with the robot, since the RF signal passes through obstacles as persons, furniture or even walls. Unfortunately, the RSSI measure is very

noisy, especially in an indoor environment due to interference and reflections (multipath). However, our approach proved to be successful in such a challenging environment. We adopted an Extended Kalman Filter SLAM algorithm to integrate RSSI measurements from the different nodes over time, while the robot moves in the environment. In a previous work [12], we realized that the main source of error in the mote localization is the initial guess of the motes position, which is necessary to initialize the Extended Kalman Filter. In this paper, after introducing the RSSI-based SLAM method, we extended it with a new mote position initialization technique that uses particle filters and resulted to be much more accurate of the initialization based on trilateration, which was used in previous works.

3.1 RSSI-Based Range-Only SLAM

To estimate the distance between two motes by measuring the RSSI, it is required to know the signal propagation law. Unfortunately, the characteristics of the radio channel are largely affected by the environment and are difficult to capture by a unique model. In this work, we refer to the standard path-loss model, which is generally considered reasonable for a large number of scenarios [7]. Accordingly, we estimate the distance d_i of a node whose radio signal is received with power P_i as:

$$d_i = d_0 \cdot 10^{\frac{A - P_i}{10n_p}} \quad (1)$$

where

- P_i radio signal power [dBm]
- d_0 is the far-field reference distance ($d_0 = 1.071467$ m)
- A is the nominal received power at the reference distance d_0 ($A = -45$ dBm).
- n_p is the path loss coefficient ($n_p = 2$)

The power measure $P[\text{dBm}]$ can be obtained through a linear conversion of the RSSI measurements returned by the node transceiver. We observe that the distance estimate given by [1] will be affected by an error due to the noisy RSSI readings. This error is taken into account in the filtering matrices of the KF, as detailed below.

EKF SLAM is a well-known technique that recursively solves the online SLAM problem where the map is feature-based. It estimates at the same time the position of the robot and the position of the features: in our case the features are the motes. Thus, the state vector of the system that should be estimated is:

$$q_k = [x_k, y_k, \theta_k, x_{m_1}, y_{m_1}, \dots, x_{m_n}, y_{m_n}]^\top \quad (2)$$

where $[x_k, y_k, \theta_k]^\top$ is the robot position at time k and $[x_{m_i}, y_{m_i}]^\top$ is the location of the i -th mote.

During the EKF SLAM prediction phase, the state (i.e., its mean and covariance) at time k is updated according to the *motion model*:

$$\mu_k^- = f(u_k, \mu_{k-1}) \quad (3)$$

$$\Sigma_k^- = A_k \Sigma_{k-1} A_k^\top + B_k Q_k B_k^\top \quad (4)$$

where μ_{k-1} and Σ_{k-1} are the mean and covariance of the state q at time $k - 1$.

When the robot receives a new radio message from the i th node, it reads the RSSI value and estimates the distance $d_{k,i}$ using [1] we can perform the correction phase of the EKF. Given h the function that maps the current state in an expected measurement $\hat{d}_{k,i}$ given the mote m_i , we can update the mean and covariance of the state:

$$\mu_k = \mu_k^- + K_k (d_{k,i} - h(\mu_k^-, i)) \quad (5)$$

$$\Sigma_k = (I - K_k H_k) \Sigma_t^- \quad (6)$$

where H_k is the Jacobian of the function h and K_k is the Kalman gain. Note that thanks to the unambiguous ID associated to every mote that is embedded in every sent packet, in our case the data association problem is solved. For a more detailed description please refer to [12].

3.2 Delayed Initialization

As shown in [12], the largest part of the residual error in the estimation of the mote positions is due to a wrong initialization of the motes in the Kalman filter at the beginning of the SLAM procedure. In this paper, we propose the use of a particle filter for implementing a delayed initialization in the mote positions. In our previous work [12], the initialization of the Extended Kalman Filter was performed with the trilateration method. The robot collected three measures of its distance to a mote, from three non-aligned positions. Three circumferences centered in each robot position and with radius the corresponding robot-mote distance are traced. The intersection of the circles corresponds to the mote position. Unfortunately, RSSI range measurements are very noisy, so the measure of distances are better represented by a mean distance averaged over several measures with an associated standard deviation, hence, an annular ring with a Gaussian distribution with the peak at the mean distance radius, see Fig. 2. The estimated mote position now lays somewhere in the intersection of the three annular rings (or better in the intersection of the three Gaussian distributions associated to the three distances), see the grayish ellipse in Fig. 2. We modeled all this, instancing for each mote a particle filter in which the samples are hypotheses on the position of the mote drawn from a normal distribution. In Fig. 2(Right), the robot received the first message from a mote and calculates the mote-robot distance r from the RSSI using [1]. The samples are uniformly distributed around a ring of radius r and standard deviation σ , previously calculated with a calibration table in which at each measured distance is associated a standard deviation. The weight associated to every particle is given by the Gaussian distribution. When the robot receives new messages from the mote and calculates new estimations of the mote distance, these correspond to new annular probability distributions centered in the corresponding robot position. We apply a Sampling Importance Resampling (SIR) algorithm each time a new distance estimation is available, making the particle to condensate toward the real mote position, see Fig. 4 in Section 5. To solve the problems of the frequent outliers, that can arise when estimating the distance from an RSSI measure, especially in the first steps of the initialization, we also distributed uniformly in the environment a certain percentage of the particles (namely

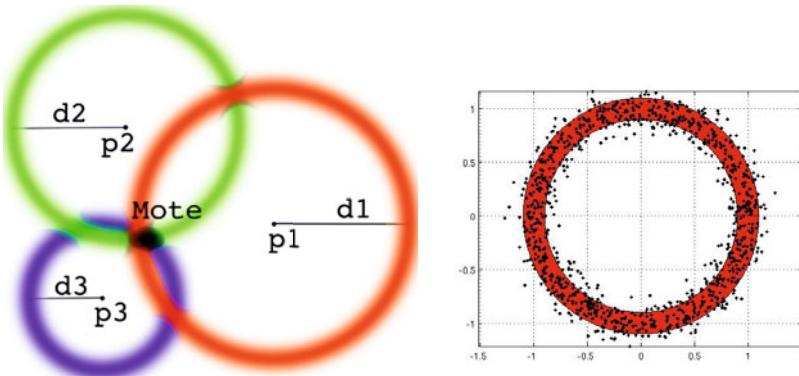


Fig. 2. (Left) An example of trilateration. P_1 , P_2 , and P_3 are robot positions along the path of the robot where the corresponding distances robot-node d_1 , d_2 , and d_3 were estimated using the RSSI. (Right) The anular ring and the samples drawn from the underlying Gaussian distribution associated to a distance estimated by the robot measuring the RSSI of the messages received by a mote.

the 5 %). This enables the particle filter to recover from a totally wrong estimation as in the case of the well-known “*kidnapped robot problem*” [17].

4 Object Recognition and Reaching

Even if, the delayed initialization using the particle filter improves the results of the RSSI-based localization, the residual error still is the order of a meter or so. This is too much if we want the robot to be able to approach and manipulate the object, especially when in the same room there are many smart objects. Our proposal is to perform the precise localization of the smart objects by recognizing their appearance in the robot’s camera. Operatively, when the robot is in the surrounding of the object of interest it switches to *Discovery Mode*. A *Sleep* message is sent to non-interesting motes and the *Descriptor Request* packets are sent to the mote of interest. This is done to minimize the battery energy consumption of the sensors and to avoid interference with other active motes. Received descriptors are passed to the object identification module. The robot controller executes the following steps:

- Grab an image with the on-board camera;
- Calculate the descriptors of this image;
- Match a subset of these descriptors with those arrived from object’s mote;
- Return object presence and position to the robot navigation module.

In the last years, many works of visual object recognition was successfully implemented exploiting Scale-Invariant Feature Transform (SIFT) descriptors. The SIFT descriptors, introduced by David Lowe in 1999 [10] [11], extract features from an input image to find a match on a set of different reference images. However, in indoor environment,

most of the time the ambient is dim and the light is not enough for grabbing clear images. In the case of a mobile robot moving around, seeking for the objects of interest, this means that the images grabbed by the robot will likely be affected by motion blur. In our first implementation [12] of a recognition system based on SIFT feature, the system was missing correct matches when the robot moved too fast. We solved this problem by using, instead of SIFT, a new feature detector scheme called MoBIF (Motion Blur Invariant Feature detector) we originally developed for humanoid robots [13]. MoBIF descriptors proved to perform similarly to SIFT and SURF on standard datasets and outperformed SIFT and SURF in images affected by motion blur or in dim images. Like SIFT, MoBIF descriptors are particularly robust to affine transformation and occlusion. For further details please refer to [13][14].

4.1 Object Appearance Description

The MoBIF descriptors provide a compact representation of the appearance of the object and they can be stored in the tiny mote's memory. A single MoBIF descriptor, named MoBIF block, occupies approximately 128 bytes. Since the IEEE 802.15.4 protocol data unit (PDU) has a payload of 28 bytes only, each MoBIF block needs to be fragmented in approximately 7 PDUs for transmission. Each MoBIF block is assigned a 2 bytes signature (MoBIF Identifier, SID) that marks all the fragments of that block, thus making it possible to recognize the fragments of the same original MoBIF block at robot side. Beside the SID field, each PDU also carries a sequence number (SN) field, of only 3 bits, that specifies in which order the fragments have to be reassembled. Furthermore, we dedicated the first two bytes of the payload to carry the SID field, leaving the remaining 26 bytes for the MoBIF data fragment.

4.2 Descriptor Clouds

Unfortunately, MoBIF descriptor (like SIFT and SURF) are not robust to large perspective transformations nor to large rotations of the object along z-axis that occurs when the robot might observe the objects from very different points of view. Another problem regards the scale invariance of SIFT, SURF, or MoBIF. These descriptors show a good invariance to the scale only until a certain limit (in the order of a couple of meters for the size of our objects).

We addressed these two problems taking many pictures of the object from different points of view, separated by 20-30 degrees and at several distances (1 m, 3 m, and 5 m). We extracted the descriptors from each image. The descriptors coming from all the pictures are merged in a single *descriptor cloud* and the object matching process is made only once on this cloud. During the building of the cloud, the descriptors coming from each picture are added to the cloud in an incremental way. If a descriptor is too similar to another descriptor already present in the cloud, it is rejected.

Let define $Z = \{Z_1, Z_2, \dots\}$ as the set of reference pictures. The descriptor cloud is D_c .

1. Set $D_c = \emptyset$.
2. For each reference image Z_i in Z do:

- (a) For each descriptor Y_j extracted from Z_i :
 - if $\exists Y_s \in D_c$ so that Y_j is similar to Y_s then discard Y_j
 - else add Y_j to D_c

The set of descriptors resulting from this process completely describes the external surface of the object. Searching for objects in a frame is hence done by looking for correspondences on this set.

It should be noted that this union of descriptors potentially similar due to the fact that the reference images might share common regions does not compromise the recognition process. In fact the redundant descriptors are joint together in a single sample which is sufficiently different from any other descriptor of the cloud. Thus its selectivity is preserved.

On the contrary, considering each reference picture one by one, the time required to evaluate the correspondences would drastically increase, because the total number of descriptors increases quickly. On the contrary, merging the information coming from all the pictures and deleting redundant descriptors allows to recognize the object from any point of view maintaining reasonable computational requirements.

4.3 Object Recognition

The object identification process begins by processing the images Im_1 taken by the onboard camera in order to extract a list I_1 of MoBIF descriptors. This set of descriptors is compared with the list of descriptor I_2 associated to the reference image Im_2 of an object. If there is a minimum number of descriptors of I_1 fairly similar to I_2 , we can assume the object appears in the image.

The recognition of the object in the image acquired by the onboard camera encompasses three steps, which are repeated for each descriptor x of I_1 :

1. Find the descriptors y_1 and y_2 of I_2 that are closer to x (using the Euclidean norm as distance);
2. If the ratio between the distance of y_1 and y_2 from x is less than a certain threshold (in this case we used 0.49 as suggested by Lowe [1]) the correspondence between y_1 and x is accepted;
3. The process is repeated until I_1 is empty or the number of correspondences is greater than a threshold. In this case, it is likely that the target object is present in Im_2 .

The object identification process does not require the association of all MoBIF components of the object with all MoBIF components extracted from the current frame. This makes the system robust to packet loss. The proposed method allows to precisely locate the object without prior knowledge of an approximate position information as required by registration techniques like the ICP algorithm that can fall into a local minimum [4]. The solution found by our method can be further refined using the ICP algorithm.

5 Experiments

The robot is a custom-built wheeled differential drive platform based on the Pioneer 2 by MobileRobots Inc, see Fig. 1(Right). The robot is equipped with only a camera and



Fig. 3. Example of correct matches obtained with MoBIF descriptors (left) a zoom of an image grabbed by the robot without motion blur, (right) an image grabbed with motion blur

the odometers connected to the two driven wheels. The mote is mounted on top of the robot and acts as a gateway to the WSN. The experiment¹ in Fig. 4 is performed in outdoor scenario and the robot is equipped with more sensors like compass and gyroscopes to improve the odometry. Then the experiment is finalized to localize all the sensor in the environment.

5.1 Discovering and Mapping New Objects

When the robot receives from the mote the MoBIF-based object descriptors, it is able to look for this object in the surrounding environment. Once the object is discovered, the mote is labeled in the map with this descriptor.

In our experiments, we use several types of smart objects with well defined appearance (i.e., with distinctive shape and/or distinctive pattern, see for example Fig 1). In all the experiments we performed, the robot, in addition to correctly localize itself and build a map of the perceived motes, correctly recognized the smart objects and associated them to the right mapped mote. In Fig. 3 an example of correct recognition even in presence of motion blur is presented. The red dots are the MoBIF descriptors.

Table 1. Landmark's mean error position

Mote ID	2	3	4	5	6
x_{mean} (cm)	22.17	-31.11	-18.91	1.29	20.77
y_{mean} (cm)	15.93	-48.80	9.01	-39.54	101.92

5.2 SLAM with Delayed Initialization

We successfully tested our delayed initialization strategy with some dataset where our previous range-only SLAM approach [12] based on initialization with trilateration, failed due to a wrong initial guess of the mote positions. In such a cases the Extended Kalman Filter may quickly diverge, making impossible to reconstruct the right map of the mote positions in the environment and the robot path. With the proposed delayed strategy based on a particle filter, the initial mote position is estimated over several

¹ Watch the simulation on web site: www.dei.unipd.it/~danielet/video

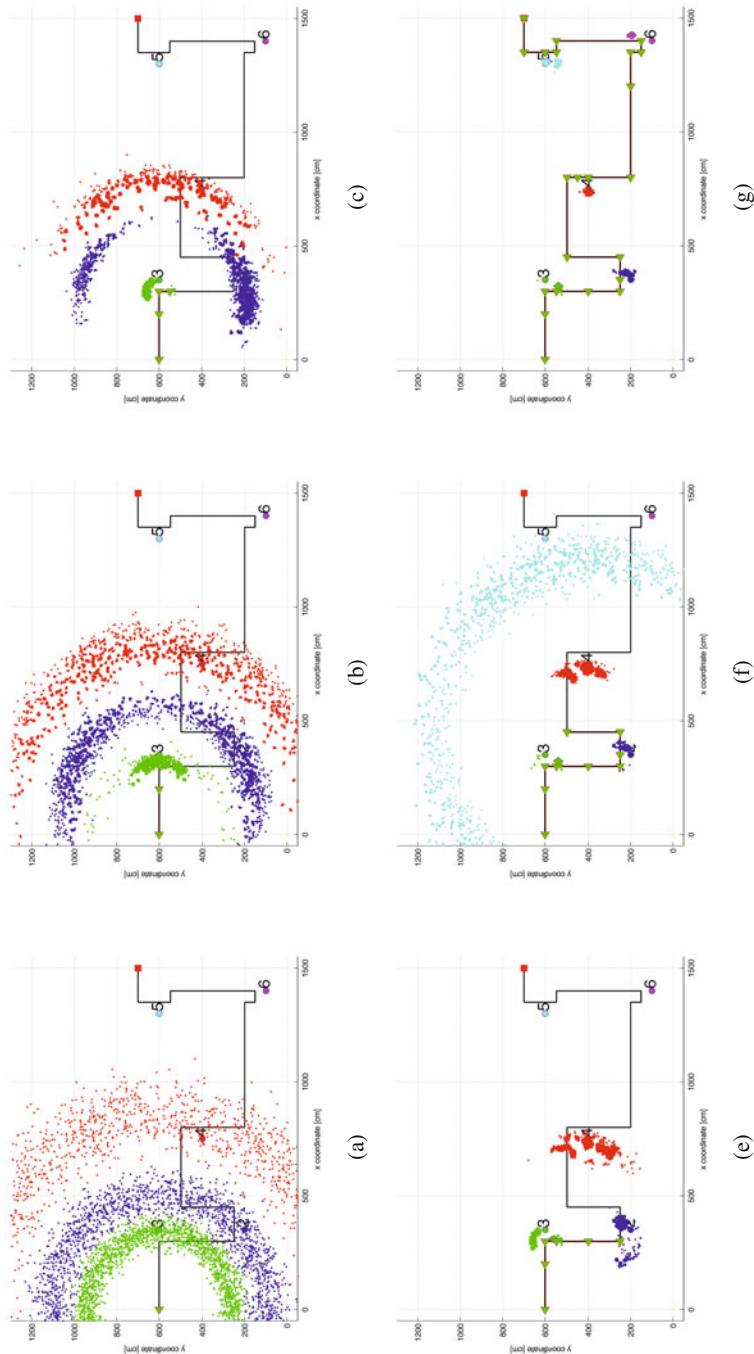


Fig. 4. The delayed initialization with the particle filter in action while the robot (the green triangle) is moving in the environment. (First row)(a) The robot received the first distance RSSI thus the annular rings are created around the robot and it encompasses the motes; (b) The robot moved; new measures are inserted in the particle filter and the samples cluster around the mote position; (c) the particle filter converged and the node is initialized in the Extended Kalman Filter. (Second row) More complex situation along the robot path with several showing the convergence of multiple particle filters (d), (e), (f).

consecutive range measures, filtering out in a probabilistic way the outliers. An example of SLAM convergence with the proposed approach is depicted in Fig. 4. Before incorporating a new mote inside the map, the robot maintains and updates for some steps a set of particles for every new detected mote: the mote is incorporated once its location uncertainty decreases below a threshold. Despite the wide noise in the RSSI range measurements, our approach make it possible to well recover the robot trajectory and roughly estimate the motes positions (Fig. 4 (g)). The accuracy of the proposed algorithm in localizing the different smart objects is shown in Table 1.

6 Conclusions and Future Work

In this paper, we presented the implementation in a single framework of a system of localization and recognition of objects for home or service robots, based on the concept of Intelligent Ambient and Wireless Sensor Network. The robot does not have any a priori information on the objects it needs to locate. It is equipped with the same mote that is attached to the object to make them “*smart*”. We presented preliminary experiments in which the robot is able to discover, locate, and recognize the object by exploiting WSN technologies, a range-only SLAM algorithm, and appearance descriptors robust to motion blur. In the future, we want to integrate in the SLAM filter also the localization information one can extract from the robot’s camera images, once the camera is precisely calibrated. The next step will be to map not only the motes, but also the rest of the environment by integrating, with the system presented in this paper, the Visual SLAM algorithm we are developing for humanoid robots. Thus, the smart objects will be located in a 3D visual map of the environment. The final step will be to integrate this complete system with a Robotics Brain Computer Interface we are developing in collaboration with IRCCS San Camillo of Venice (Italy) and the University of Palermo (Italy) [5]. The BCI system will enable to select just “*by thinking*” the smart object we want the robot to interact with. This will open, also to people with severe disabilities (like people affected by ALS - Amyotrophic Lateral Sclerosis), the possibility to interact with a domotic house or an intelligent ambient.

References

1. Network robot forum, <http://www.scat.or.jp/nrf/English/>
2. Asfour, T., Azad, P., Vahrenkamp, N., Regenstein, K., Bierbaum, A., Welke, K., Schröder, J., Dillmann, R.: Toward humanoid manipulation in human-centred environments. *Robot. Auton. Syst.* 56(1), 54–65 (2008)
3. Azad, P., Asfour, T., Dillmann, R.: Combining apperance-based and model-based methods for real-time object recognition and 6d localization. In: International Conference on Intelligent Robots and Systems (IROS), Beijing, China. IEEE Institute of Electrical and Electronics Engineers (2006)
4. Besl, P.J., McKay, N.D.: A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.* 14(2), 239–256 (1992)
5. Chella, A., Pagello, E., Menegatti, E., Sorbello, R., Anzalone, S., Cinquegrani, F., Tonin, L., Piccione, F., Prifitis, K., Blanda, C., Buttita, E., Tranchina, E.: A bci teleoperated museum robotic guide. In: International Conference on Complex, Intelligent and Software Intensive Systems, CISIS 2009, pp. 783–788 (March 2009)

6. Dressler, F.: Self-organization in autonomous sensor and actuator networks (2006)
7. Goldsmith, A.: Wireless Communications. Cambridge University Press, Cambridge (2005)
8. Kim, J., Kim, Y., Lee, K.: The third generation of robotics: Ubiquitous robot. Palmerston North, New Zealand (2004)
9. Lee, J., Hashimoto, H.: Intelligent space - concept and contents. Advanced Robotics 16(3), 265–280 (2002)
10. Lowe, D.: Object recognition from local scale-invariant features. In: Proceedings of International Conference on Computer Vision, pp. 1150–1157 (1999)
11. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision 60, 91–110 (2004)
12. Menegatti, E., Zanella, A., Zilli, S., Zorzi, F., Pagello, E.: Range-only SLAM with a mobile robot and a wireless sensor networks. In: IEEE International Conference on Robotics and Automation, ICRA 2009, pp. 8–14 (May 2009)
13. Pretto, A., Menegatti, E., Bennewitz, M., Burgard, W., Pagello, E.: A visual odometry framework robust to motion blur. In: IEEE International Conference on Robotics and Automation, ICRA 2009, pp. 2250–2257 (May 2009)
14. Pretto, A., Menegatti, E., Pagello, E.: Reliable features matching for humanoid robots. In: 7th IEEE-RAS International Conference on Humanoid Robots, pp. 532–538 (January 2007)
15. Saffiotti, A., Broxvall, M., Grittì, M., LeBlanc, K., Lundh, R., Rashid, J., Seo, B., Cho, Y.: The PEIS-ecology project: Vision and results. In: IROS 2008: Intelligent Robots and Systems, pp. 2329–2335. North-Holland Publishing Co., Amsterdam (2008)
16. Sanfeliu, A., Hagita, N., Saffiotti, A.: Network robot systems. Robot. Auton. Syst. 56(10), 793–797 (2008)
17. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics (Intelligent Robotics and Autonomous Agents). The MIT Press, Cambridge (2005)
18. Vincze, M., Schlemmer, M., Gemeiner, P., Ayromlou, M.: Vision for robotics: a tool for model-based object tracking. IEEE Robotics and Automation Magazine 12(4), 53–64 (2005)
19. Zanca, G., Zorzi, F., Zanella, A., Zorzi, M.: Experimental comparison of RSSi-based localization algorithms for indoor wireless sensor networks. In: REALWSN 2008: Proceedings of the Workshop on Real-World Wireless Sensor Networks, pp. 1–5. ACM, New York (2008)
20. Danieletto, M., Mina, M., Zanella, A., Zanuttigh, P., Menegatti, E.: Recognition of smart objects by a mobile robot using sift-based image recognition and wireless communication. In: Proc. of European Conference on Mobile Robotics (ECMR 2009), Mlini/Dubrovnik, Croatia (October 2009)

Simulation for the Optimal Design of a Biped Robot: Analysis of Energy Consumption

Federico Moro^{1,2}, Giuseppina Gini¹, Milos Zefran², and Aleksandar Rodic³

¹ Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy

² Department of Electrical and Computer Engineering,

University of Illinois at Chicago, USA

³ University of Belgrade, Institute Mihajlo Pupin, Robotics Laboratory, Serbia

Abstract. Our first aim is to develop a systematic method to estimate energy consumption of bipedal locomotion. This method is then used to evaluate the performance of materials and actuators that could be used for the design of a biped robot. Moreover, with this analysis we also demonstrated the importance of having good joint trajectories in order to reduce energy consumption. Results collected are then integrated with complementary information about materials and actuators, to finally suggest the best configurations. These indications are meant to be used for future developments of LARP, the biped of Politecnico di Milano. The method adopted, however, is general enough to produce valid results for any robot, and we hope our considerations will help in evaluating design choices for future humanoid robots.

Keywords: model of biped for locomotion simulation, computation of energy consumption, materials and actuators for the design of a robot.

1 Introduction

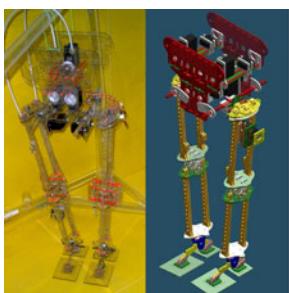
Locomotion is an active research in humanoid robotics. To evaluate the performance of a walking robot the main features to consider are stability and energy-efficiency.

Our first aim is to design an energy-efficient robot. We based our research on LARP [1], the biped robot of Politecnico di Milano, and analyzed the energetic performance of different design solutions. In particular, we focused on the possible materials to use for the structure of the robot, and on the actuators. The results are meant to be used for the improvement of LARP itself, but are general enough to be a valid indication for the development of other humanoid robots.

In the last decade, research on bipedal locomotion has seriously enforced, and the performance of the robots have significantly increased. Among the most skilled robots, *Asimo* [2], produced by Honda, and *Wabian-2R* [3], from Waseda University, deserve a special mention. Asimo is 130 cm tall for 54 kg, and can walk at a speed of 2.7 km/h, or run at up to 6 km/h. Its 34 degrees of freedom, of which 6 in each leg, are actuated by servomotors. Its dynamic stability is guaranteed by a Zero Moment Point (ZMP) controller. Wabian-2R, whose height

is 150 cm and weight with batteries is 64 kg, is the one that performs the most natural and human-like gait, with stretched knees and heel-contact and toe-off motion. These results are achieved by the use of a redundant mechanism with 2 degrees of freedom in the waist. Wabian-2R also has two 1 degree of freedom passive joints in its feet, that sum up with the other 39 active joints in the whole body, actuated by servomotors. ZMP is adopted to ensure stability. Last, in chronological order, is *Petman* [4], of the Boston Dynamics, supposed to be completed in 2011. It is the evolution of the famous BigDog [5], and the challenge is to obtain the same performance in the more difficult context of bipedal locomotion. Another problem concerns the actuators: BigDog is powered by a big combustion engine, which drives 12 hydraulic actuators. Petman, being smaller, will also need smaller motors.

The Light Adaptive-Reactive biPед, referred as *LARP* now on (Figure IIa), is the humanoid robot developed at DEI, Politecnico di Milano. Being focus on locomotion, it was decided to start building only the lower part of the robot. The structure is in polycarbonate, with some small parts in carbon-fiber. The reason of that choice was the good strength to weight ratio, as well as the fact that this material is way cheaper than others. The robot is 90 cm high, has 12 active degrees of freedom, 6 per leg, distributed in 3 in the hip, 1 in the knee, and 2 in the ankle, and 4 passive degrees of freedom, 2 in each foot, representing the heel and the toe. The target of building a light robot is completely satisfied, being LARP weight less than 5 kg. Moreover, dimensions, mass distribution, and range of motion of each joint reproduce those of an average human being. Currently LARP has 12 HITEC HS-805BB servomotors, that provide up to 24.7 kg cm torque each, located in the pelvis. A system of coramide tendons (with a tensile strength of 800 N, for a diameter of 0.5 mm) transmits the torques from the pelvis to the joints. The motors in this position keep high the center of mass of the robot, making it easier to correct the stability. On the other hand, friction generated by the transmission represents a big loss in terms of energy. To control the joint stiffness, each servomotor is equipped with a spring and a damper. A 3D model of the robot, developed in MSC Adams is also available (Figure IIb).



	Measure [cm]
Pelvis width	27.63
Femur length	35.1
Tibia length	37.25
Foot length	21.5
Foot width	10

Fig. 1. (a) Robot LARP, of DEI, Politecnico di Milano, (b) its model in MSC Adams, (c) and its dimensions

Two main features [6] characterize the mechanical design of LARP: the knee and the foot. The knee adopts a solution developed in advanced prostheses, that use a multi-axial mechanism in which the center of rotation is not fixed, as in a pin joint, but moves along a trajectory that depends on the mechanism structure itself. The foot, whose design is a crucial phase to obtain stability and energetic efficiency, is composed by an arc and by 2 passive degrees of freedom, representing the heel and the toe. This solution permits to manage the energy received by the ground reaction force, and doesn't constrain the ankle in one fixed position.

2 Model of the Robot and Simulation

Our main goal is to develop a formal, reproducible method, not constrained by the specific characteristics of LARP, to determine the efficiency of certain configurations. The system architecture is represented in Figure 2. The first step is modelling the robot in a simulation environment. We decided to work in Matlab, exploiting the opportunities provided by the first version of the *Humanoid robots simulation platform* [7], now on referred to as *HRSP*, developed at the *Institute Mihailo Pupin*. We detected those parameters that affect the most the energetic performance of a robot, deciding, hence, to concentrate on the material used for the structure, and the actuators.

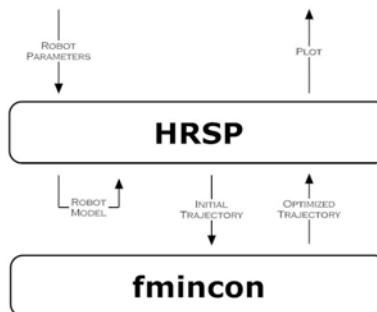


Fig. 2. Schema of the system developed to evaluate the energy efficiency of different design solutions

At this point, we needed a way to evaluate the performance of the different design solutions. That is, determine whether the robot is more efficient when a certain material and a certain kind of actuators are used, with respect with other possible solutions. The plan was to optimize the joint trajectories, and compare the energy consumption necessary to execute it in the different configurations. *fmincon*, a function provided with the *Optimization toolbox*, was the tool of choice.

2.1 The Model of the Robot

Using Denavit-Hartenberg notation with multiple kinematic chains we defined in HRSP the parametric model of a robot that reflects some characteristics of LARP. In this model we didn't reproduce the specific mechanics of the knee and the foot previously described: we just adopted a pin joint knee and a flat foot. The measures of the single parts, reported are in Figure IIc, as well as masses, center of gravity of each link, and inertia matrices, respect those of the real robot.

2.2 The Parameters

First decision to be taken is what material to use to build the structure, that is those parts that behave like bones for humans. The main features of the available materials are in Table II. Candidate materials must satisfy certain requirements: be light, strong, and, possibly, not too expensive.

Table 1. Main characteristics of the materials considered for the structure of the robot

	Density [g/cm ²]	Young's modulus [GPa]	Tensile strength [MPa]	Cost
Polycarbonate	1.2	2	65	Low
Aluminium alloy	2.7	69	310	Medium
Titanium alloy	4.4	110	1000	High
Stainless steel	7.9	193	570	Medium

Polycarbonate, a thermoplastic material widely used in the engineering field, has a high strength to weight ratio, being its density only 1.2 g/cm², a restrained cost, and is easy to fabricate. Unfortunately, experience tells us this material is not strong enough to resist to impacts.

This is the reason why we decided to consider stronger materials. *Young's module* is the most used measure of the stiffness of an elastic material, while the *tensile strength* indicates the edge to pass from an elastic deformation to a plastic deformation. These mechanical parameters describe the performance of a material in terms of its strength. We focused our attention on three materials widely used in high-performance engineering applications.

Tempered 6061 aluminium alloy has a density of 2.7 g/cm², Young's modulus of 69 GPa, and tensile strength of 310 MPa. It contains magnesium and silicon as its major alloying elements, and is one of the most common alloys of aluminium. The high availability of raw material, aluminium is the most abundant metal in the Earth's crust, and the limited processing costs, due to its low melting point, make it an affordable solution.

Titanium 6Al-4V alloy, also known as titanium alloy grade 5, has a density of 4.4 g/cm², Young's modulus of 110 GPa, and tensile strength of 1000 MPa. It has a chemical composition of 6% aluminium, 4% vanadium, and remainder titanium, and is significantly stronger than pure titanium, while having the

same stiffness. Because of these properties it is the most commonly used titanium alloy. Compared to the aluminium alloy it has better mechanical features, though remaining light enough. Its main weakness is cost: both raw material and processing are expensive.

A more affordable material is the 316 stainless steel, a solid solution of iron with alloying elements, main of which are Cr, between 16% and 18%, and Ni, between 10% and 14%. It has a density of 7.9 g/cm^3 , Young's modulus of 193 GPa, and tensile strength of 570 MPa. This means that its mechanical features are comparable or even better than those of the titanium 6Al-4V alloy, though its cost is way lower. Unfortunately this material is heavier than the others, and this would affect the weight of the robot and, consequently, its efficiency, with an incidence that we're going to evaluate with the method we developed.

A second parameter to study is what kind of actuators to use; this choice has a big impact on energy efficiency, in a way that is less predictable than in the materials case. In robotics the solution of servomotors is the most widely adopted. If the system is correctly set, an average efficiency of 80% can be reached. Some advanced robots are actuated by pneumatic or hydraulic linear actuators. These systems, however, are driven by auxiliary components, such as compressors or high pressure cylinders, that are heavy, noisy, and need a lot of space. This turns out to be a big limitation, that reduces the field of use to academics. A more sophisticated solution is given by EAPs actuators [8], polymers whose shape is modified when a voltage is applied to them. Interesting are the linear stack actuators based on dielectric EAPs, or multi-layer dielectric EAPs actuators [9]. The classical structure of dielectric EAPs, where a passive silicone or acrylic elastomer film is coated on both sides with electrodes, is replicated, stacking up several layers of this basic unit. This approach makes it possible to enlarge force and deformation in thickness direction, and, even more interesting, this kind of actuators, controlled by the applied voltage, behave in a way that is really similar to human muscles. These devices are extremely light, and can reach an average efficiency that is major than 90%. Of course, the use of linear actuators to drive rotational joints implies an additional loss, that depends also on the varying angle of the joint, and the simulation model we developed takes it into account.

2.3 The Optimization Module

Once developed a parametric simulation model of LARP in Matlab, we needed to define a way to evaluate the performance of the different solutions analyzed, in order to compare them.

We decided to use fmincon, provided by the Optimization toolbox, that finds the minimum of a constrained nonlinear multi-variable function, to determine which joint trajectories minimize the energy consumption for each material/actuator configuration. We used the active set algorithm [10], which is supposed to be best suited to our problem, and is one of the more efficient and accurate in the state of the art. The fmincon problem is specified in the following way:

$$\min_x f(x) \text{ such that} \left\{ \begin{array}{l} c(x) \leq 0 \\ ceq(x) = 0 \\ A \cdot x \leq b \\ Aeq \cdot x = beq \\ lb \leq x \leq ub \end{array} \right.$$

where

x, b, beq, lb, ub : vectors;

A, Aeq : matrices;

$c(x), ceq(x)$: functions, possibly nonlinear, that return vectors;

$f(x)$: objective function, possibly nonlinear, that returns a scalar;

1. x is a vector containing the concatenation on joint angles at the different frames of a step, that is the minimal periodic sequence in bipedal walking.
2. $f(x)$, the objective function, is the evaluation of energy consumption to achieve one step, following a certain trajectory described by x . We started from the dynamic equation of motion: $H(q)\ddot{q} + C(q, \dot{q})\dot{q} + N(q) = \tau$, where H is the inertia matrix, C includes the Coriolis and centrifugal forces, and Gravity terms are included in the vector N . Velocity and acceleration vectors \dot{q} and \ddot{q} are the first order and the second order derivatives of the vector q , respectively. The mechanical energy is then computed as $M = \int_0^T \dot{q}^T \tau dt$. The efficiency of the different actuation solutions, and the additional losses given by linear actuators are then taken into account to evaluate the electrical energy E necessary to perform the desired trajectory. All of these equations are discretized to be used in Matlab.
3. It was then necessary to define the constraints the optimization problem must satisfy. With lb , lower bound, and ub , upper bound, the possible configurations of the joints are limited into a physical range. With the linear inequalities also velocity and acceleration of the joints are limited, while with the linear equalities the starting and the final position of the robot are fixed. The last constraint, defined with the nonlinear inequalities, is given by the ground, considered that during the swing phase the foot should avoid any contact with it.

This method, unfortunately, turned out to be computationally too complex. The length of vector x is equal to the number of joints multiplied by the number of frames in one step. Even if we set the time step to be long enough, obtaining only 50 frames per step, complexity remains too high. Some test, taking into account just one leg per time, that is 6 degrees of freedom, took months to get to a result. A complete simulation, with all the 12 degrees of freedom of the robot, would require fmincon to find the 600 values of a vector x that minimize the objective function, not violating any constraint, in an enormous solution space; this would mean, reasonably, years of simulation.

We investigated how to use distributed computing. Unfortunately, the basically sequential nature of fmincon makes it impossible to take advantage of parallelism, and makes it necessary to look for alternative solutions. We hypothesized that this alternative could be genetics algorithms: we substituted the fmincon function with the similar *ga* function, provided with the Genetic Algorithms toolbox, but results were not satisfactory. Given the high number of constraints, it takes a big initial population to just find a solution respecting all of them: setting a big initial population, on the other hand, makes the *ga* function even slower than fmincon. We also tried the optimization platform provided by *Tomlab*, but, again, the problem was too complex, and also this solution turned out to be not sufficient.

2.4 The Simplified Optimization Problem

At this point we approached the problem in a different way. We identified some simplifications that wouldn't affect the results we are looking for but reduce enough the complexity to allow fmincon process it in a reasonable time.

We respected a basic assumption: considered that this work aims to define whether certain changes in the LARP design can improve its efficiency, any approximation has to be conservative with respect to the current LARP configuration. The simplifications we adopted are, in a certain way, penalizing the choice of EAPs: thus, in case we obtain that, from our simplified model, they require less energy to perform the robot walking then servomotors, we're sure that, in the real system, their use would guarantee an enhancement that is equal or major than that reached in simulation. For this reason we decided to evaluate only the energy required to perform the swing phase of one leg, with fixed pelvis, when the robot is performing a rectilinear walking. Taking into consideration only one leg per time reduces the degrees of freedom to 6. The weight of the leg depends mainly on the material while the choice of actuators doesn't really affect it. On the contrary, using EAPs definitely reduces the weight of the pelvis. The energy required by the 6 degrees of freedom we're not considering, then, will be higher in the servomotors case. Moreover, the fact that we decided to set the robot walking trajectory as rectilinear makes it superfluous to evaluate all of the 6 degrees of freedom in the swinging leg: most of the work will be done in the sagittal plane, as in the case of planar robots. We can then just consider the 3 degrees of freedom, 1 one in the hip, 1 in the knee, and 1 in the ankle, that permit the motion of the leg in this plane, and reduce the length of vector x to 150. The last approximation is in modelling the linear actuators: for each joint, we set the two application points at the same distance from the joint itself. This solution is not meant to be the best possible, but reduces the weight of the calculation of energy consumption, further reducing the computational complexity.

These approximations provided a model whose energetic performance could be evaluated by the fmincon function in a reasonable time: every simulation took 10 to 14 hours, and we had the results of the 8 possible combinations of the 4 materials and the 2 actuators.

Fmincon, by the way, doesn't guarantee global optimality: in complex problems, as ours is, it is quite typical that the solution found is only a local minimum. However, improvements from the starting point, whatever geometric joint trajectory you might adopt, are relevant; energy consumption with an optimized trajectory is even orders of magnitude less than the starting one. This, as we'll see in the next section, happens because the peaks of energy consumption are eliminated.

3 Experimental Results

We can finally report the results obtained on the simplified problem.

A first observation regards the local minima in the search for the optimal trajectory. Fmincon doesn't ensure results returned are globally optimal. We'll see in Section 3.1 how we handled this possible limitation. But before reporting the results, we introduce the notation adopted to identify the configurations. The actuators are numbered: 1 refers to servomotors, and 2 to EAPs actuators. An alphabetical letter is associated to the materials: *a* polycarbonate, *b* aluminium alloy, *c* titanium alloy, and *d* stainless steel. This means that when, for instance, we write 1*a*, we intend the current LARP configuration, with polycarbonate, and servomotors.

3.1 Energy-Efficiency of the Different Configurations

We performed 8 simulations, one for each configuration. All of them returned new joint trajectories and a value indicating the energy consumption. For the reasons explained in the previous section, these values are not directly comparable, because they may refer to different local minima. Anyway, the trajectories found are way more efficient than the starting one. Hence, we have a pool of 8 good trajectories, and we calculated the energy consumption for the different configurations for all of them. Since these values account only the energy necessary to move the swinging leg in the single support phase, absolute values are not actually significant. We are interested in the ratio between the different configurations, and in particular the ratio between current LARP configuration and the other analyzed. Results are reported in Figure 3.

It is interesting that in the single trajectories the ranking of the performance of the different configurations doesn't change: in particular, the order is always 1*b*, 1*a*, 2*b*, 2*a*, 3*b*, 3*a*, 4*b*, 4*a*. What changes a lot, instead, is how much these parameters influence the performance in the different trajectories. It turns out that in some of these, as for instance trajectory 7, using different configurations doesn't affect to much the efficiency. In some others, as trajectory 1, the influence of the choice of the parameters is clearer. This explains the high standard deviation observed.

From Figure 3a, we also noticed that the ratio between the performance of configurations 1*a* and 1*b* is constant. Whatever trajectory you consider, configuration 1*b* has an energy consumption that is 0.8889 of that of configuration

1a. We verified that also the $2b/2a$ ratio, as well as $3b/3a$ and $4b/4a$, is 0.8889. This means that, fixed the material, the use of EAPs reduces energy consumption, that then will be around the 89% of that of servomotors. This percentage, furthermore, is an upper bound: it's important to remember that all of the approximations adopted to model our robot are penalizing EAPs. Anyway, this observation tells us that the influence of the actuators on the energetic performance does not depend on the material of the structure, and vice versa. Hence, we can continue our analysis of the effects of the choice of parameters, considering them independently.

	Traj 1	Traj 2	Traj 3	Traj 4	Traj 5	Traj 6	Traj 7	Traj 8	Average	SD
1a	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.0000
2a	2.5474	1.5394	1.5613	1.5037	2.4754	1.5596	1.2939	1.3802	1.7326	0.4901
3a	4.3324	2.6448	2.2149	2.0779	4.1892	2.6497	1.7021	1.8209	2.7040	1.0197
4a	8.0527	4.9873	4.1762	3.3427	7.7580	4.9069	3.1683	2.8112	4.9117	2.0155
1b	0.8889	0.8889	0.8889	0.8889	0.8889	0.8889	0.8889	0.8889	0.8889	0.0000
2b	2.2643	1.3684	1.3878	1.3367	2.2003	1.3863	1.1501	1.2268	1.5401	0.4356
3b	3.8510	2.3509	1.9688	1.8470	3.7237	2.3552	1.5103	1.6186	2.4032	0.9068
4b	7.1579	4.4331	3.7122	2.9699	6.8960	4.4417	2.9162	2.4988	4.3657	1.7917

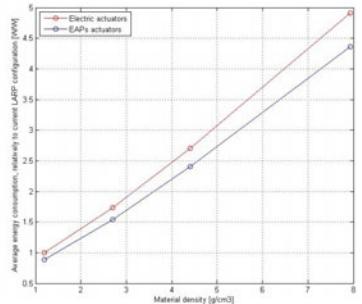


Fig. 3. Energy consumption, relatively to current LARP configuration, (a) in the different trajectories, (b) and graph of average values

About which material to use for the structure, we should avoid polycarbonate, because, though light, it is too fragile, and stainless steel, because too heavy and really inefficient.

The aluminium alloy and the titanium alloy are the most suitable materials: they're a good trade off between strength and efficiency. The choice between them depends mainly on the cost. The structural characteristics of the titanium alloy are definitely better. If this solution is not affordable, the aluminium alloy represents an absolutely valid, and more economic, alternative.

Choice on actuators is in a certain way simpler. In perspective, EAPs have a greater potential than servomotors. Being their structure similar to that of human muscles, the movement they produce is smoother and definitely more natural. They're also more efficient, as shown in the previous sections. But the fact that this is a young technology makes this solution more risky than servomotors.

3.2 Further Considerations upon the Optimization Module

As explained in the previous sections, the fmincon function doesn't guarantee global optimality. Theoretically, this would represent a serious limitation. Experimental data, instead, reveal the goodness of the method we adopted. We checked the energy consumption before and after optimization, for all of the

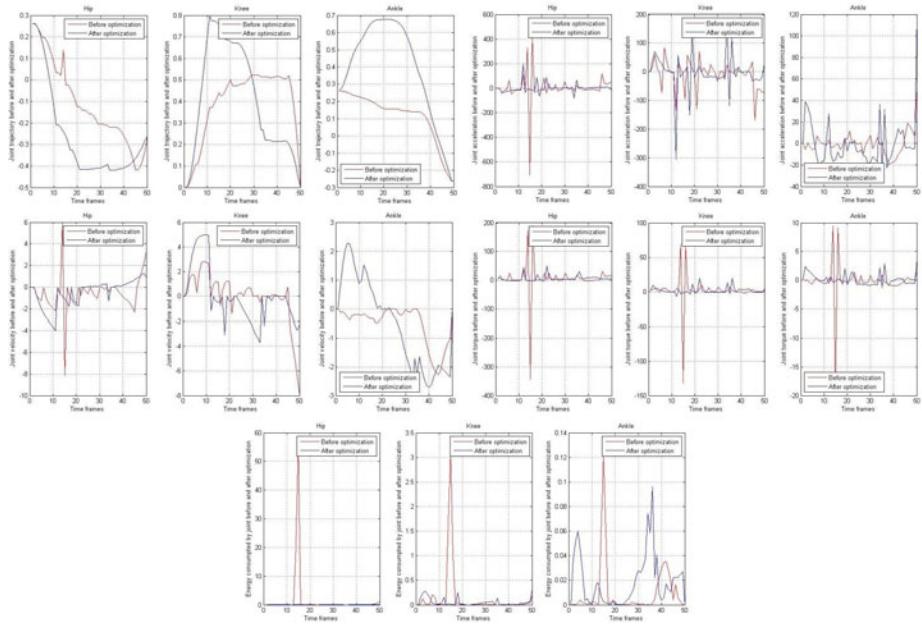


Fig. 4. Comparison of joint (a: top left) trajectories, (b: middle left) velocities, (c: top right) accelerations, (d: middle right) torques, and (e: bottom) energy consumptions (note that the y-axis have different scales) before (red) and after (blue) optimization

eight simulations, and found that there is a consistent improvement of the performance. On average, the optimized trajectories require only the 1.97% of the energy necessary to perform the initial trajectory. In the best case this percentage decreases to 0.65%, in the worst it is 3.61%.

We discuss the simulation of the current LARP. Figure 4 represents the comparison of (a) the trajectory of the joints, of (b) their velocity, of (c) their acceleration, and of (d) the torques required, before and after the optimization. Just looking at the trajectories, it's not easy to understand why their performance is so different. The velocity figure starts helping, but it is in the accelerations and in the torques figures that it becomes clear how the optimization produced smoother trajectories: the highest peaks in accelerations and torques are eliminated, in particular in the hip joint, that is the one that requires more energy to move. In Figure 4e we can see the energy consumption, frame by frame, for each joint. To eliminate the peak around frame 15, the new trajectory might require more energy at some other instants, but the consumption now remains approximatively constant, and the comprehensive result, on the complete step, is excellent.

To conclude, we considered the foot trajectory. The constraints we formulated didn't impose any particular condition on the gait, but its correctness, that is avoiding the contact with the ground while the leg is swinging, and fixing the

joints configuration at the first and the last frames. It is interesting to notice how the returned trajectories generate a gait that really reminds the human one. The foot always remains closer to the ground than with the original trajectory, and the movements are smoother and look a lot more natural. This could also be considered as a confirmation of how energy efficiency is one of the main goals in the human gait.

4 Conclusions and Future Work

Energy optimization is one of the most important objectives of research in autonomous robotics, that need to reach a certain degree of independence from human intervention for a reasonable time.

We have seen how dramatically the definition of the gait influences the energetic performance. Our interpretation of the goodness of our results is that the approach we proposed to generate trajectories itself is different. Usually, trajectories of the end-effector (in this case the foot) are imposed to follow a certain geometric shape. Joints trajectories are then obtained from the foot trajectory using inverse kinematics. These joints trajectories may result to be not smooth at all, and this generates the peaks of energy consumption we've shown in Figure 4e. Our method, instead, imposes only the correctness of the foot trajectory. Attention is then focused on the joints. The contribution of every single joint to the overall energy consumption is decisive for the choice of the trajectory. This approach has proved to be effective, and results obtained exceed expectations. It is also important to notice that the choice of using a very simplified simulation system permitted us to focus only on those parameters we are considering, sure that results are not affected by any other. This wouldn't have been possible if we decided to make use of a more precise, yet complex, 3D simulator. Moreover, the use of the fmincon function has an advantage: it works completely independently from the context of robotics, unaware of the fact that what it is optimizing are joint trajectories. This means that the results we got cannot be affected by a priori considerations. Returned trajectories, though, really remind those adopted in human locomotion, definitely more than the geometric trajectories generated on the basis of theoretical assumptions. This represents an important validation to the results we obtained.

About what material to use for the structure of the robot, as we've seen, there are two solutions that satisfy all the requirements: the tempered 6061 aluminium alloy, and the titanium 6Al-4V alloy. It is possible to achieve a further considerable improvement in the energetic performance of the robot by adopting more sophisticated solutions for the design of the joints, as done for the knee and the foot in LARP, or for the waist in Wabian-2R. About the actuators, a safe solution, for now, is represented by the more consolidated servomotors. In this case, it also becomes important to decide where to locate them. In LARP they have been included in the pelvis structure. If the robot was completed with an upper body, though, it would be possible to locate servomotors closer to the joints they actuate, and reduce the energy loss caused by friction, maintaining

also a mass distribution similar to that of human body. Furthermore, the development of an upper body would improve even more energy efficiency. Although we usually associate arms with manipulation, they also play an important role in human walking. For instance they are used to counteract the rotation of the pelvis. It could then be interesting to think about a whole-body posture control, to reproduce also the way humans maintain stability. In fact, to manage external disturbances, we don't only modify the characteristics of our gait, as the length and width of our steps, but we also change the trunk inclination, and use arms to balance.

References

1. Gini, G., Folgheraiter, M., Scarfogliero, U., Moro, F.: A biologically founded design and control of a humanoid biped. *Humanoid Robots*. Tech education and Publishing, Vienna (2009)
2. Sakagami, Y., Watanabe, R., Aoyama, C., Matsunaga, S., Higaki, N., Fujimura, K.: The intelligent ASIMO: system overview and integration. In: IEEE/RSJ International Conference on Intelligent Robots and System, Lausanne, Switzerland, vol. 3 (2002)
3. Ogura, Y., Aikawa, H., Shimomura, K., Morishima, A., Lim, H.-o., Takanishi, A.: Development of a new humanoid robot WABIAN-2. In: IEEE International Conference on Robotics and Automation 2006, Orlando, Florida (2006)
4. Committee on Full-System Testing and Evaluation of Personal Protection Equipment Ensembles in Simulated Chemical-Warfare Environments, National Research Council: Soldier Protective Clothing and Equipment: Feasibility of Chemical Testing Using a Fully Articulated Robotic Mannequin. The National Academies Press (2008)
5. Raibert, M., Blankespoor, K., Nelson, G., Playter, R.: The Big-Dog Team: Bigdog, the rough-terrain quadruped robot. In: 17th World Congress, The International Federation of Automatic Control, pp. 10822–10825 (2008)
6. Gini, G., Folgheraiter, M., Scarfogliero, U.: New joint design to create a more natural and efficient biped. *Journal of Applied Bionics and Biomechanics* 6(1), 27–42 (2009)
7. Rodic, A.: Humanoid Robot Simulation Platform - HRSP, Matlab/Simulink software toolbox for modeling, simulation & control of biped robots. Robotics Lab, Mihailo Pupin Institute (2009), <http://www.institutepupin.com/RnDProfile/robotics/hrsp.html>
8. Bar-Cohen, Y. (ed.): Electroactive polymer (EAP) Actuators as artificial muscles. Reality, potential, and challenges, 2nd edn., vol. PM136. SPIE Press, San Jose (2004)
9. Carpi, F., Salaris, C., De Rossi, D.: Folded dielectric elastomer actuators: Smart Materials and Structures 16, S300–S305 (2007)
10. Gill, P.E., Murray, W., Wright, M.H.: Numerical Linear Algebra and Optimization, vol. 1. Addison Wesley, Reading (1991)

Efficient Use of 3D Environment Models for Mobile Robot Simulation and Localization

Andreu Corominas Murtra, Eduard Trulls,
Josep M. Mirats Tur, and Alberto Sanfeliu

Institut de Robòtica i Informàtica Industrial
C/Llorens i Artigas 4-6. 08028, Barcelona, Spain
`{acorominas,etrulls,jmirats,asanfeliu}@iri.upc.edu`
<http://www.iri.upc.edu>

Abstract. This paper provides a detailed description of a set of algorithms to efficiently manipulate 3D geometric models to compute physical constraints and range observation models, data that is usually required in real-time mobile robotics or simulation. Our approach uses a standard file format to describe the environment and processes the model using the *openGL* library, a widely-used programming interface for 3D scene manipulation. The paper also presents results on a test model for benchmarking, and on a model of a real urban environment, where the algorithms have been effectively used for real-time localization in a large urban setting.

Keywords: 3D environment models, observation models, mobile robot, 3D localization, simulation, openGL.

1 Introduction

For both simulation or real-time map-based localization, the mobile robotics community needs to implement the computation of expected environment constraints and expected sensor observations, the latter also called sensor models. For all these computations, an environment model is required and its fast and accurate manipulation is a key point for successful results of upper-level applications. This is even more important since current trends on mobile robotics leads to 3D environment models, which are richer descriptions of the reality but harder models to process.

Using computer accelerated graphics card for fast manipulation of 3D scenes has been addressed by robotic researchers from some years ago, specially in the simulation domain [4][2][1], but it remains less explored in real-time localization. In real-time particle filter localization [8][3], expected environment constraints and expected observations are computed massively at each iteration from each particle position, causing the software modules in charge of such computations being a key piece for the final success of the whole system. Unlike simulation

field, there exists few work reporting the use of accelerated graphics cards to implement the online computation of expected observations [5].

Moreover, detailed descriptions on processing 3D geometric models to efficiently compute physical constraints or expected observations are not common in the robotics literature. 3D geometric models have started to become an essential part of mobile robot environment models in recent years, specially for applications targeting outdoor urban environments, and thanks in part to the availability of powerful and affordable graphics cards on home computers. This paper wants to remedy the lack of a detailed description on how to efficiently manipulate such models, presenting in a detailed way the algorithms for an optimal use of the graphics card acceleration through the OpenGL library [7]. Thanks to their fast computation, the presented algorithms are successfully used for real-time map-based localization in a large urban pedestrian environment, demonstrating the potential of our implementation.

The paper begins introducing the 3D environment model in section 2. Section 3 details the algorithms to compute the physical constraints of wheeled platforms within the environment. Section 4 presents the algorithm to compute expected range observations, minimizing the computation time while keeping sensor's accuracy. Section 5 evaluates the performance of the algorithms and briefly describes a real application example. Finally, section 6 concludes the work.

2 Overview of the 3D Environment Model

The goal of our environment model is to represent the 3D geometry of a real, urban pedestrian environment in a useful way for map-based localization and simulation. Thus, the model incorporates the most static part of the environment, such as buildings, floor, benches, and other fixed urban furniture.

Our approach uses the *.obj* geometry definition file format [6]. Originally developed for 3D computer animation, it has become an open format and a *de facto* exchange standard. We use two different maps for our intended application. One map features the full environment, \mathcal{M} , while the other consists only of those surfaces traversable by the robot, $\mathcal{M}_{\text{floor}}$, leaving holes instead of modelling obstacles. Figure 1 shows a view of the full model for the UPC (Universitat Politècnica de Catalunya) campus area and figure 2 shows real pictures of this environment and their approximate corresponding views in the 3D model.

This environment model implicitly defines a coordinate frame in which all the geometry is referenced. Therefore a 3D position can be expressed with respect to the map frame as follows:

$$X_p^M = (x_p^M, y_p^M, z_p^M, \theta_p^M, \phi_p^M, \psi_p^M) \quad (1)$$

where the *xyz* coordinates define the *location* of the position and the $\theta\phi\psi$ coordinates parametrize the position *attitude* by means of the three Euler angles heading, pitch and roll.

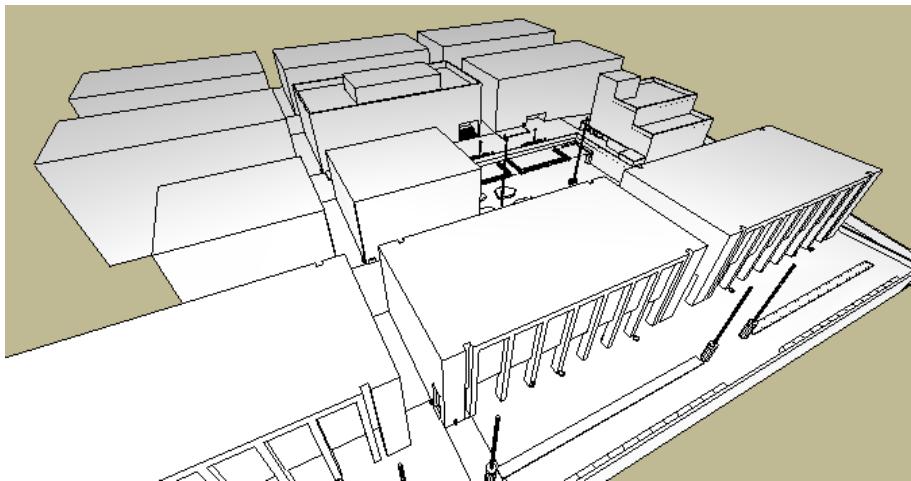


Fig. 1. 3D environment model of the UPC campus area



Fig. 2. Pictures of the UPC campus environment and their approximate corresponding views of the 3D map. The 3D map only models the most static part of the environment.

3 Physical Constraints

Terrestrial, mobile robots have position constraints, due to gravity and the environment. Computing such physical constraints is a basic requirement for simulation but also for map-based localization, since the search space is reduced dramatically, therefore improving the performance of the localization algorithm.

3.1 Gravity Constraints

A wheeled robot will always lie on the floor, due to gravity. For relatively slow platforms in can be assumed as well that the whole platform is a rigid body, so that a suspension system, if present, does not modify the attitude of the vehicle. With these assumptions, gravity constraints for height, pitch and roll can be derived from the environment.

The height constraint sets a height, z^M , for a given coordinate pair (x^M, y^M) . To compute it only the floor map is used. Algorithm 1 outlines the algorithm. The key idea is to renderize the 3D model from an overhead view point, setting a projection that limits the rendering volume in depth and aperture in order to renderize only the relevant part of the model. Afterwards, by means of an OpenGL routine, the algorithm reads the depth of the central pixel.

Algorithm 1. Height constraint algorithm for 3D models

INPUT: $\mathcal{M}_{\text{floor}}, (x^M, y^M)$

OUTPUT: g_z

```

setWindowSize(5,5); //sets rendering window size to 5x5 pixels
setProjection(1°, 1, zmin, zmax) //1° of aperture, aspect ratio, depth limits
Xoverhead = (xM, yM, zMoverhead, 0, π/2, 0); //sets an overhead view point, pitch= π/2
renderUpdate(ℳfloor, Xoverhead); //renders the model from Xoverhead
r = readZbuffer(3,3); //reads depth of central pixel
gz = zMoverhead - r;
return gz;

```

The pitch constraint fixes the pitch variable of the platform to a given coordinate triplet $(x_p^M, y_p^M, \theta_p^M)$. The algorithm to compute the pitch constraint is outlined in algorithm 2. It employs the previous height constraint algorithm to compute the floor model's difference in height between the platform's frontmost and backmost points, g_{zf} and g_{zb} . The pitch angle is then computed using trivial trigonometry, while L is the distance between the above mentioned platform points. The roll constraint can be found in a similar way but computing the height constraint for the leftmost and rightmost platform points. Please note that the roll constraint applies to all wheeled platforms, but the pitch constraint does not apply to two-wheeled, self-balancing robots, such as ours, based on Segway platforms.

Algorithm 2. Pitch constraint algorithm for 3D models

INPUT: $\mathcal{M}_{\text{floor}}, L, (x_p^M, y_p^M, \theta_p^M)$
 OUTPUT: g_ϕ

$$\begin{aligned} x_f^M &= x_p^M + \frac{L}{2} \cos \theta_p^M; //\text{compute the platform's frontmost point} \\ y_f^M &= y_p^M + \frac{L}{2} \sin \theta_p^M; //\text{likewise} \\ g_{zf} &= \text{heightConstraint}(x_f^M, y_f^M); //\text{compute height at frontmost point} \\ x_b^M &= x_p^M - \frac{L}{2} \cos \theta_p^M; //\text{compute the platform's backmost point} \\ y_b^M &= y_p^M - \frac{L}{2} \sin \theta_p^M; //\text{likewise} \\ g_{zb} &= \text{heightConstraint}(x_b^M, y_b^M); //\text{compute height at backmost point} \\ g_\phi &= \text{atan2}(g_{zf} - g_{zb}, L); \\ \text{return } g_\phi; \end{aligned}$$

3.2 Offline Height Map Computation

The constraints presented in the previous section are computed intensively in filtering applications such as map-based localization. To speed up online computations during real-time execution, a height grid is computed offline for the floor map. G_{height} is then a grid containing the height value z^M for pairs (x^M, y^M) , thus being a discrete representation of the height constraint with a xy step γ :

$$G_{\text{height}}(i, j) = g_z(x_p^M, y_p^M) \quad | \quad i = (\text{int}) \frac{x_p^M - x_0^M}{\gamma}, \quad j = (\text{int}) \frac{y_p^M - y_0^M}{\gamma}, \quad (2)$$

where x_0^M and y_0^M are the map origin xy coordinates. The z^M value is computed offline by means of the height constraint (algorithm II) along the grid points. Figure 3 shows the height grid for the UPC campus environment.

Note that this approach is valid for maps with a single traversable z-level, such as ours, and while our algorithms can be directly applied to multi-level maps further work would be required in determining the appropriate map section to compute. Since computing the pitch and roll constraints requires several z^M computations, the height grid speeds up these procedures as well. To avoid discretization problems, specially when computing pitch and roll constraints using

Algorithm 3. Grid version of the height constraint

INPUT: $G_{\text{height}}, (x_p^M, y_p^M)$
 OUTPUT: g_z

$$\begin{aligned} i &= (\text{int}) \frac{x_p^M - x_0^M}{\gamma}; \quad j = (\text{int}) \frac{y_p^M - y_0^M}{\gamma} \\ z_1 &= G_{\text{height}}(i, j); \\ (i_2, j_2) &= \text{nearestDiagonalGridIndex}(); //i_2 = i \pm 1; \quad j_2 = j \pm 1; \\ z_2 &= G_{\text{height}}(i_2, j); //\text{height of a neighbour cell} \\ z_3 &= G_{\text{height}}(i, j_2); //\text{height of a neighbour cell} \\ z_4 &= G_{\text{height}}(i_2, j_2); //\text{height of a neighbour cell} \\ g_z &= \text{interpolation}(z_1, z_2, z_3, z_4, x_p^M, y_p^M); \\ \text{return } g_z; \end{aligned}$$

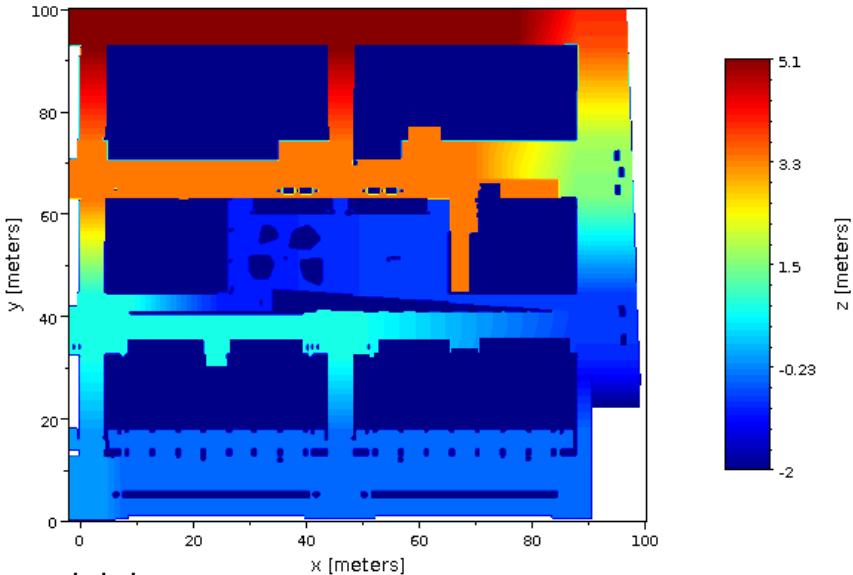


Fig. 3. Height grid, G_{height} , of the UPC campus environment

G_{height} , we use lineal interpolation on the grid. Algorithm 3 summarizes the grid version of the height constraint.

4 Range Observation Model

Another key factor in dealing with environment models is the computation of expected observations, also called sensor models, or simulated sensors. This is also useful for both simulation and real-time map-based localization. This section outlines how, from a given 3D position in the environment, expected 2D range scans and expected 3D point clouds are computed. A common problem in either case is the computation of range data given the sensor position and a set of sensor parameters like angular aperture, number of points and angular accuracy. To compute these observation models, we use again OpenGL renderization and depth buffer reading, but the approach specially focuses on minimizing the computation time without violating sensor's accuracy and resolution. This minimization is achieved by reducing the rendering window size and the rendering volume as much as possible, while keeping the sensor accuracy.

The goal of a range observation model is to find a matrix \mathbf{R} of ranges for a given sensor position X_s^M . Each element r_{ij} of the matrix \mathbf{R} is the range computation following the ray given by the angles α_i and β_j . Figure 4 shows these variables as well as coordinate frames for the map, $(\mathbf{XYZ})^M$, and for the sensor, $(\mathbf{XYZ})^s$.

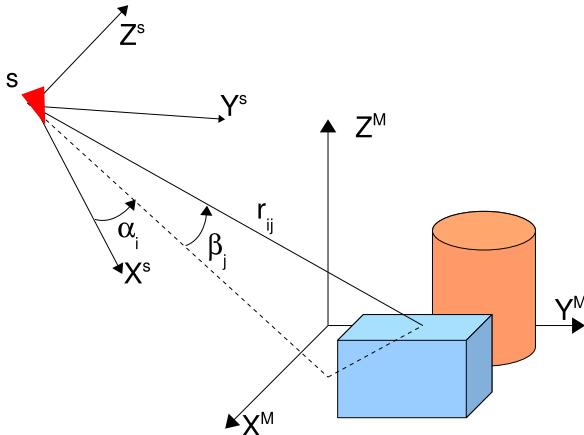


Fig. 4. Model frame, sensor frame, angles α_i and β_j , and the output ranges r_{ij}

The range observation model has the following inputs:

- A 3D geometric model, \mathcal{M} .
- A set of sensor parameters: horizontal and vertical angular apertures, Δ_α and Δ_β , horizontal and vertical angular accuracies, δ_α and δ_β , the size of the range matrix, $n_\alpha \times n_\beta$, and range limits, r_{min}, r_{max} .
- A sensor position, $X_s^M = (x_s^M, y_s^M, z_s^M, \theta_s^M, \phi_s^M, \psi_s^M)$.

The operations to execute in order to compute ranges r_{ij} are:

1. Set the projection to view the scene.
2. Set the rendering window size.
3. Render the scene from X_s^M .
4. Read the depth buffer of the graphics card and compute ranges r_{ij} .

Set the Projection. Before using the OpenGL rendering, the projection parameters have to be set. These parameters are the vertical aperture of the scene view, which is directly the vertical aperture of the modelled sensor, Δ_β , an image aspect ratio ρ , and two parameters limiting the viewing volume with two planes placed at z_N (near plane) and z_F (far plane)¹. These two last parameters coincide respectively with r_{min} and r_{max} of the modelled sensor. The only parameter to be computed at this step is the aspect ratio ρ . To do this, first the metric dimensions of the image plane, width, w , and height, h , have to be found. The aspect ratio will be derived from them:

$$w = 2r_{min}tg\left(\frac{\Delta_\alpha}{2}\right); h = 2r_{min}tg\left(\frac{\Delta_\beta}{2}\right); \rho = \frac{w}{h}; \quad (3)$$

Figure 5 depicts the horizontal cross section of the projection with the associated parameters. The vertical cross section is analogous to the horizontal one.

¹ z_N and z_F are OpenGL depth values defined in the screen space.

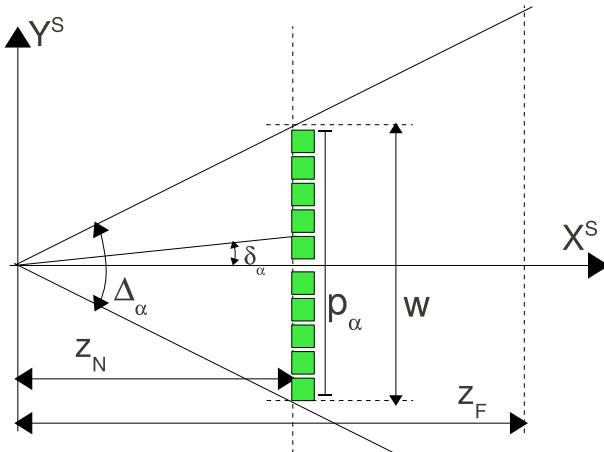


Fig. 5. Horizontal cross section of the projection with the involved parameters. Green squares represent the pixels at the image plane.

Set the rendering window size. Before rendering a 3D scene, the size of the image has to be set. Choosing a size as small as possible is a key issue to speed up the proposed algorithm. Since range sensors have limited angular accuracy, we use that to limit the size of image, in order to avoid rendering more pixels than those required. Given a sensor with angular accuracies δ_α and δ_β , pixel dimensions of the rendering window are to:

$$p_\alpha = (\text{int})2 \frac{\tan(\Delta_\alpha/2)}{\tan(\delta_\alpha)}; \quad p_\beta = (\text{int})2 \frac{\tan(\Delta_\beta/2)}{\tan(\delta_\beta)}; \quad (4)$$

Figure 5 shows an horizontal cross section of the projection and the related variables to compute the horizontal pixel size of the rendering window (the vertical pixel size is found analogously).

Render the scene. The scene is rendered from the view point situated at sensor position X_s^M . Beyond computing the color for each pixel of the rendering window, the graphics card also associates to each one a depth value. Moreover, graphics card are optimized to discard parts of the model escaping from the scene, thus having limited the rendering window size and volume speeds up the rendering step. Renderization can be execute in a hidden window.

Read the depth buffer. Depth values of each pixel are stored in the *depth buffer* of the graphics card. They can be read by means of an OpenGL function that returns data in a matrix \mathbf{B} of size $p_\alpha \times p_\beta$, which is greater in size than the desired matrix \mathbf{R} . Read depth values, b_{kl} , are a normalized version of the rendered depth for each pixel. To obtain the desired ranges, we first have to compute the

D matrix, which holds the non-normalized depth values, that is the depth value of the pixels following the \mathbf{X}^s direction:

$$\begin{aligned} k &= (\text{int})(\frac{1}{2} - \frac{\text{tg}(\alpha_i)}{2\text{tg}(\frac{\Delta_\alpha}{2})})p_\alpha; \quad l = (\text{int})(\frac{1}{2} - \frac{\text{tg}(\beta_j)}{2\text{tg}(\frac{\Delta_\beta}{2})})p_\beta \\ d_{ij} &= \frac{r_{min}r_{max}}{(r_{max} - b_{kl})(r_{max} - r_{min})}; \end{aligned} \quad (5)$$

The last equation undoes the normalization computed by the graphics card to store the depth values. The **D** matrix has $n_\alpha \times n_\beta$ size, since we compute d_{ij} only for the pixels of interest. Finally, with basic trigonometry we can calculate the desired r_{ij} as:

$$r_{ij} = \frac{d_{ij}}{\cos(\alpha_i)\cos(\beta_j)} \quad (6)$$

Figure 6 shows the variables involved on this last step, showing the meaning of the d_{ij} and r_{ij} distances in an horizontal cross section of the scene. Equation 6 presents numerical problems when α_i or β_j get close to $\pi/2$. This will limit the aperture of our sensor model. However, section 5 explains how to overcome this limitation when modeling a real sensor with a wide aperture.

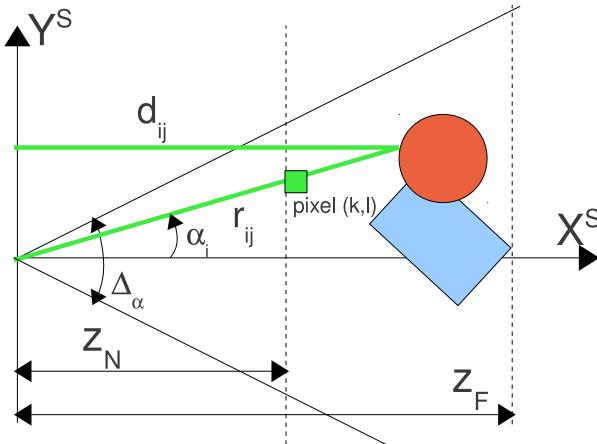


Fig. 6. Horizontal cross section of the projection, with distance d_{ij} and range r_{ij} of the corresponding kl^{th} pixel

The overall procedure is outlined in algorithm 4. Inside the **for** loops, variables k and l are directly functions of i and j respectively, so we can precompute expressions in equation 5 for k and l , and store values in a vector.

5 Experimental Evaluation

This section presents some results that evaluate the performance of our algorithms. Two range models are presented, corresponding to real laser scanners,

Algorithm 4. Range Sensor ModelINPUT: $\mathcal{M}, \Delta_\alpha, \Delta_\beta, \delta_\alpha, \delta_\beta, n_\alpha, n_\beta, r_{max}, r_{min}, X_s^M$ OUTPUT: \mathbf{R}

```

 $w = 2r_{min}tg(\frac{\Delta_\alpha}{2}); h = 2r_{min}tg(\frac{\Delta_\beta}{2}); \rho = \frac{w}{h};$ 
glSetProjection( $\Delta_\beta, \rho, r_{min}, r_{max}$ ); //rendering volume: vertical aperture, aspect ratio, depth limits
 $p_\alpha = (int)2\frac{tg(\Delta_\alpha/2)}{tg(\delta_\alpha)}; p_\beta = (int)2\frac{tg(\Delta_\beta/2)}{tg(\delta_\beta)};$ 
glSetWindowSize(0, 0,  $p_\alpha, p_\beta$ );
glRenderUpdate( $\mathcal{M}, X_s^M$ ); //renders the model from the sensor position
 $\mathbf{B} = glReadZbuffer(ALLIMAGE);$  //reads normalized depth values
for  $i = 1..n_\alpha$  do
     $\alpha_i = \Delta_\alpha(0.5 - \frac{i}{n_\alpha});$ 
     $k = (int)(0.5 - \frac{tg(\alpha_i)}{2tg(\Delta_\alpha/2)})p_\alpha;$ 
    for  $j = 1..n_\beta$  do
         $\beta_j = \Delta_\beta(0.5 - \frac{j}{n_\beta});$ 
         $l = (int)(0.5 - \frac{tg(\beta_j)}{2tg(\Delta_\beta/2)})p_\beta;$ 
         $d_{ij} = \frac{r_{min}r_{max}}{(r_{max}-b_{kl})(r_{max}-r_{min})};$ 
         $r_{ij} = \frac{d_{ij}}{\cos(\alpha_i)\cos(\phi_j)};$ 
    end for
end for
return  $\mathbf{R};$ 

```

and computational time is provided for a testbench 3D scene. Finally, we briefly describe the successful use of all presented algorithms for real-time map-based localization.

5.1 Laser Scanner Models

Two kind of laser scanner models has been used, using the same software. Table 1 summarizes the input parameters of these laser scanner models. Our implementation sets angular accuracies equal to angular resolutions. Please note also that, due to application requirements, we only model part of the scan provided by the Hokuyo laser.

Table 2 outlines the derived parameters of the models. Leuze device has an horizontal aperture greater than 180° and that poses numerical problems on computing equation 6. This issue is overcome by dividing the computation in two

Table 1. Input parameters of the laser scanner models

Input Parameter	Leuze RS4	Hokuyo UTM 30-LX (partial)
$\Delta_\alpha, \Delta_\beta$	$190^\circ, 1^\circ$	$60^\circ, 1^\circ$
n_α, n_β	133, 5 points	241, 5 points
$\delta_\alpha = \Delta_\alpha/n_\alpha, \delta_\beta = \Delta_\beta/n_\beta$	$1.43^\circ, 0.2^\circ$	$0.25^\circ, 0.2^\circ$
r_{min}, r_{max}	0.3, 20 m	0.3, 20 m

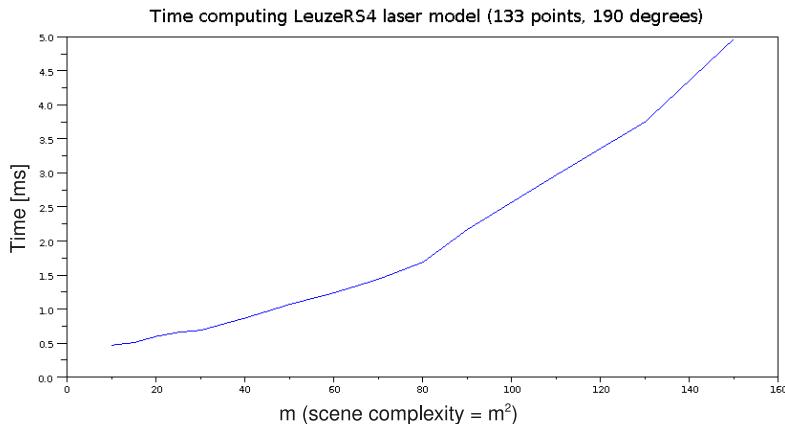


Fig. 7. Time performance versus scene complexity for the Leuze RS4 laser scanner. The sphere object has m^2 elements.

Table 2. Derived parameters of the laser scanner models

Derived Parameter	Leuze RS4 (per scanning sector)	Hokuyo UTM 30-LX (partial)
w	0.655 m	0.346 m
h	0.005 m	0.005 m
ρ	125	66
p_α	88 pixels	265 pixels
p_β	5 pixels	5 pixels

scanning sectors, each one with the half of sensor's aperture, so the parameters given in table 2 in the Leuze column are for a single scanning sector.

To evaluate the computational performance of the proposed implementation while increasing the scene complexity, we have done a set of experiments consisting on computing 100 times the Leuze model against a testbench environment composed of a single sphere, while increasing the number of sectors and slices of that shape. The results are shown in figure 7. For a given m , the sphere is formed by m sectors and m slices, and thus the scene has m^2 elements.

Please note that using the same software implementation, other range models of devices providing point clouds such as time-of-flight cameras or 3D laser scanners can be easily configured and computed.

5.2 Map-Based Localization

Gravity constraints and laser scanner models have been used for 3D, real-time, map-based localization of a mobile platform while it navigates autonomously on the UPC campus area introduced in section 2. The mobile platform is a two-wheeled self-balancing Segway RMP200, equipped with two laser devices scanning over the horizontal plane forward and backward (Leuze RS4), and a third laser

device (Hokuyo UTM 30-LX) scanning the vertical plane in front of the robot. A particle filter integrates data from these three scanners and from the platform encoders and inclinometers to output a position estimate. At each iteration of the filter, for each particle, height and roll constraints are calculated at the propagation phase by means of the grid versions of gravity constraints, so a negligible time is spent during online executions. On the other hand, expected range observations are computed online using algorithm ②. The filter runs on a DELL XPS-1313 laptop at 5 Hz, using 50 particles. This implies that the computer was calculating $5 \times 50 \times (133 + 133 + 241)$ rays per second. The approach has been proved successful as will be documented in future publications.

6 Conclusions

Although efficient computation of 3D range observation models is commonplace in robotics for a wide range of applications, little effort has been put on documenting algorithms to solve this issue. This paper details a set of algorithms for fast computation of range data from 3D geometric models of a given environment using the very well-known OpenGL programming library. Additionally, we show that the same principles can be applied to the computation of physical constraints of terrestrial mobile platforms and demonstrate our approach for a computationally expensive, real-time application.

References

1. Friedmann, M., Petersen, K., von Stryk, O.: Simulation of Multi-Robot Teams with Flexible Level of Detail. In: Carpin, S., Noda, I., Pagello, E., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2008. LNCS (LNAI), vol. 5325, pp. 29–40. Springer, Heidelberg (2008)
2. Laue, T., Spiess, K., Röfer, T.: A General Physical Robot Simulator and Its Application in RoboCup. In: Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y. (eds.) RoboCup 2005. LNCS (LNAI), vol. 4020, pp. 173–183. Springer, Heidelberg (2006)
3. Levinson, J., Montemerlo, M., Thrun, S.: Map-Based Precision Vehicle Localization in Urban Environments. In: Proceedings of the Robotics: Science and Systems Conference, Atlanta, USA (June 2007)
4. Michel, O.: Cyberbotics Ltd - WebotsTM: Professional Mobile Robot Simulation. International Journal of Advanced Robotic Systems 1(1) (2004)
5. Nuske, S., Roberts, J., Wyeth, G.: Robust Outdoor Visual Localization Using a Three-Dimensional-Edge Map. Journal of Field Robotics 26(9) (2009)
6. OBJ file format, <http://local.wasp.uwa.edu.au/~pbourke/dataformats/obj/>
7. OpenGL, <http://www.opengl.org>
8. Thrun, S., Fox, D., Burgard, W., Dellaert, F.: Robust Monte Carlo localization for mobile robots. Artificial Intelligence 128 (2001)

Decision and Coordination Strategies for RoboCup Rescue Agents

Maitreyi Nanjanath, Alexander J. Erlandson, Sean Andrist, Aravind Ragipindi,
Abdul A. Mohammed, Ankur S. Sharma, and Maria Gini

Department of Computer Science and Engineering, University of Minnesota
`{nanjan,gini}@cs.umn.edu`

Abstract. We describe the decision processes of agents in the Robocup Rescue Agent Simulation. Agents have to rescue civilians trapped in buildings and extinguish fires in a city which has been struck by an earthquake. Lack of information and limited communications hamper the rescue process. We examine how effective our strategies and algorithms are and compare their performance against the baseline agents and agents which competed in last year's competition.

1 Introduction

Disaster management and urban search and rescue is an open area of research for AI and multi-agent systems. This research not only has the potential for a huge social impact but also presents plenty of challenges. A search and rescue system has a large number of heterogeneous agents who have to act in real-time in a difficult environment with limited information while confronting problems of logistics, planning and collaboration.

The RoboCup Rescue Agent (RCR) simulation was proposed in [1] to advance research in the area of disaster management and urban search and rescue. It provides a platform for disaster management where heterogeneous field agents (police, fire brigades, and ambulances) co-ordinate with each other to deal with a simulated disaster scenario [2]. Police agents have to clear road blockades to provide access to the disaster sites, ambulance agents have to rescue civilians, and fire brigade agents have to control the spread of fire and extinguish it. The simulator also provides centers, a Police Office, a Fire Station, and an Ambulance Center, to help the field agents coordinate. Centers are critical for co-ordination, because the number and size of messages each agent can send or receive in each cycle are severely limited. The agents are given a map of the city but are unaware of the locations of civilians, blocked roads or burning buildings. They have to search the area to discover emergencies and coordinate with other agents.

Figure 1 shows a screenshot of the simulation environment. It is a portion of Kobe city where an earthquake of magnitude 6.9 on the Richter scale struck in 1995 damaging almost the entire city and causing a considerable loss of life and property. The blue, white, and red circles on the maps represent respectively the police agents, ambulance agents, and fire brigade agents. The bright green, dull green, and black circles represent healthy, hurt, and dead civilians in that order. The yellow, orange, and maroon colors represent the increasing intensity of fire in buildings; black represents completely burned and destroyed buildings. There are two special types of buildings: the

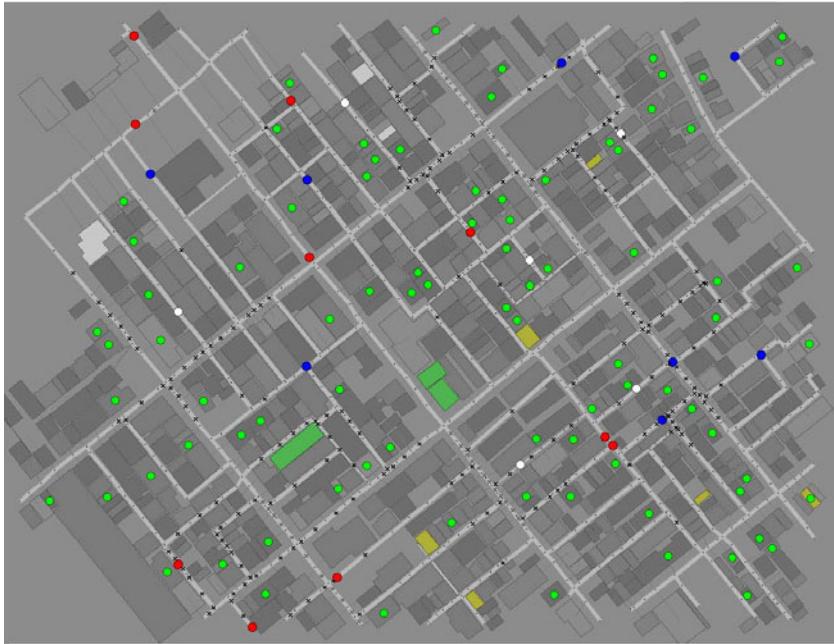


Fig. 1. The map of Kobe in the simulator

green buildings are refuges where saved civilians are taken, and the white buildings are centers. Cross marks on roads represent blockades. The simulation runs for 300 time steps.

The major contributions we make in this paper are: (1) novel algorithms for our team of agents, MinERS (Minnesota Emergency Response Squad). (2) an empirical study of performance of MinERS, which includes comparison with top agents from the 2009 RoboCup competition.

The rest of this paper is organized as follows. In the next section, we describe related work. This is followed by a description of the algorithms we use for each agent type, and experimental results. We conclude by outlining future work.

2 Related Work

In the search and rescue simulation there are many tasks, such as saving civilians, clearing the rubble, and extinguishing fires, but there is a limited number of agents, each with limited capabilities, so task allocation is the core problem.

In [3], three ways to solve the coordination problem are proposed: decentralized mutual adjustment, centralized direct supervision, and environment partitioning. The decentralized approach is more flexible but not always preferable. A hybrid approach between the decentralized and centralized approach is proposed in [4].

Evolutionary reinforcement learning is used by [5] at the ambulance center in order to decide how many ambulances should co-operate to save the civilians buried in a building. LA-DCOP, a low communication distributed constraint optimization algorithm, is used by [6]. Combinatorial auctions are used by [7]. Agents inform the police center about their rubble clearing needs. The police center passes the information to all the police agents, who submit bids. Combinatorial auctions, also used by [8], achieve optimal task allocation but require large computational power and message bandwidth. The paper compares auctions with a distributed mechanism using localized reasoning. According to the authors, the distributed approach achieves satisfactory results with low computation power and minimum messaging.

Both [3] and [7] suggested partitioning the disaster space among agents. In both cases the partitioning is pre-determined and is homogeneous. Such an arrangement can result in partitions that have a drastic difference in the number of roads in each partition. A more powerful partitioning strategy based on the degree of blockades on the roads is proposed in [3], but such approach requires massive real time survey of the environment by all the agents, which is too costly. We also use clustering, but we try to generate uniformly sized clusters.

In [9] coordination is addressed with coalition formation, solving the problem as a distributed optimization problem via DCOP and the Max-Sum algorithm, and including spatial and temporal constraints. However, the approach is not evaluated using RCR.

3 Our Approach to Decision Making and Coordination

A significant portion of the decision making and coordination among our agents is done through the centers. Specifically, the centers provide (1) communications among agents, (2) clustering algorithms, (3) support for auctions for task allocation to police and ambulance agents, (4) computation of probability distributions of presence of buried civilians over the buildings in the city.

3.1 Communications

The simulator limits severely the communications. Several communication channels are provided for the agents to send messages. A message has a maximum length of 256 bytes, so compression is needed to fit in more information into the message. A field agent can receive at most 4 messages per cycle and send a maximum of 4 messages, determined by the channels to which the agent is subscribed. Since messages are broadcast, however, this means that multiple agents sending on a channel will quickly overflow the agents capacity to receive messages. To avoid this we use round-robin to limit the cycles in which agents can send messages.

In addition to buffering messages sent by field agents, we also use message senescence so that the messages that get through tend to be the most important ones. At the start of the simulation, agents have to share a lot of information they sensed, but as the simulation proceeds the amount slowly drops. The buffering helps share information even when the channels available are not sufficient. Messages about cleared roads, fires, and civilians found are passed from the centers to the field agents to keep their local knowledge up to date.

3.2 Clustering Algorithms

The centers partition the city into clusters of roads and buildings based on the Manhattan distance between roads and between buildings, using the k -means algorithm, as implemented in CLUTO (CLUSTering Toolkit). Figure 2 shows an example of partitioning the city for the police agents. For police agents, the number of clusters is half the number of police agents.



Fig. 2. Foligno map with roads divided into clusters for the police agents

The Ambulance Center partitions the city into areas of the same size and assigns each to an ambulance. The number of clusters is determined by the size of the city. Ambulances switch to a different unexplored area as soon as they finish exploring their own area, so spreading out over the city.

Fires start at multiple locations and spread to nearby buildings. This results in the formation of clusters of buildings on fire. The Fire Station partitions known fires into clusters of buildings, with a number of clusters equal to half the number of available fire brigades. This enables sending the fire brigades where they are most needed.

3.3 Auctions

Unlike fire agents where every unattended fire must be put out as promptly as possible, police and ambulance agents need to determine which portions of the city to attend to first to minimize civilians' deaths. They do this using auctions.

We use Sequential Single Item Auctions [10], which have the advantage of providing a better solution than the basic parallel auction [11] that assigns tasks to agents directly. In Sequential Single Item Auctions, a bid for a task accounts for the cost of previous tasks the agent has on its plate. This ensures that no agent is overloaded with all the tasks and ensures that tasks get addressed in a timely fashion.

The major challenge in implementing the auctions is due to the communication restrictions. Since each agent can only send and receive four messages per timestep, auctioning directly to agents would require multiple timesteps during which agents could instead be productive. We work around this by setting up a proxy agent that handles the actual auction process. The proxy is created by the center and it tracks the current and expected positions of the agents at each timestep. It also tracks whether agents are idle or busy at the specific timestep. Using the proxy introduces some error in the bidding process, since an agent may not be at the exact predicted location, but the error is bounded by the time taken by the agent to get back to that location, and is therefore never more than 2 timesteps.

The auction is implemented as follows:

1. Field agents send requests to the center for different tasks as they sense them.
2. The centers sort the requests with the most critical request first.
3. The centers activate the agent proxy with the requests to complete the auction.
4. The agent proxy assigns tasks created from the requests to agents that are not presently at work. Task assignment is done by estimating how long it would take for the agent to travel to that location, and then to perform the task. For ambulances the costs includes the cost of carrying civilians to a refuge.
5. After a task assignment, the proxy saves the cost and uses it when computing the total cost for the next task.
6. When all tasks are assigned, the auction stops and the centers send out the list of assignments to the agents.
7. The proxy continues to monitor agents after assignment, and marks them as idle when their tasks are completed.

3.4 Probability Distributions

The Ambulance Center maintains a probability distribution of civilians over all buildings. This distribution is populated using sense messages received from field agents. The ambulance teams are periodically sent this information so that their local knowledge also gets updated.

We model the process of determining the probability of occupancy of a building as a Markov Process, described by the state and current inputs. Using this, we arrive at the following probability update rule for civilians heard (civilians seen are identified precisely; heard information only provides a range within which the person is probably present, and no other data). Given:

a = a person is in a building nearby

b = a person can be heard, so $P(\neg b|\neg a) = 1$, and it is specified that $P(\neg b|a) = 0.9$

c = a person is in this particular building, so $P(a|c) = 1$.

$P(c)$ is the probability of occupancy for each individual building, which we desire to compute.

$P(a) = 1 - \prod_{i=1}^n (1 - P_i)$, where P_i is the probability of building i having any occupants and n is the number of buildings in hearing range. Then we have the following:

$$\begin{aligned} P(a|\neg b) &= P(\neg b|a) \times P(a) / (P(\neg b|a) \times P(a) + P(\neg b|\neg a) \times P(\neg a)) \\ &= 0.9 * P(a) / (0.9 * P(a) + (1 - P(a))) = 0.9 \times P(a) / (1 - 0.1 \times P(a)) \\ P(c|\neg b) &= P(c|a) \times P(a|\neg b) = 0.9 \times P(c) / (1 - 0.1 \times P(a)) \\ P(c|b) &= P(c|a) \times P(a|b) = P(a|c) \times P(c)/P(a) = P(c)/P(a) \end{aligned}$$

Because of communication limitations, messages may be received by the center out of order. Each time a message arrives out of synchronization, all messages from that point on are reexamined and the probabilities recalculated. By doing this we can treat the sense messages received as dependent only on the state at that time step, given the knowledge available at that time step. We thus satisfy the Markov property by maintaining independent behavior from other time steps, and hence satisfy the requirements for the Bayesian network update rule.

The probability of hearing a civilian per cycle is only 0.1 (10%). However, because there are multiple field agents and their information is pooled by the center, the possible location of any civilian heard can be narrowed down to a few buildings very quickly. Empirically over the course of 300 timesteps, all the living civilians get located.

4 Field Agents Design

Since field agents have limited communications they cannot coordinate efficiently. We rely on the centers to provide coordination functionality.

When the simulation starts, the agents get the map of the city but debris and the resulting blockades are visible only when an agent is close to them, i.e. none of the agents has a comprehensive view of the complete disaster situation. At the field agent level, agents rely on low-level coordination strategies to determine where they need to go. They use their knowledge of the starting map and assign themselves to specific regions to explore, ensuring that there is limited duplication of effort between them. They rely on the centers to provide them with more detailed information on pending tasks. In what follows, we describe the basic design of the field agents.

4.1 Police Agents

Police agents are responsible for clearing blockages on the roads which hamper the movement of ambulance and fire brigade agents. The cost to clear the debris from a road segment is directly proportional to its amount. Debris removed from the road can reappear as the simulation proceeds. While the debris clearance does not directly affect the score in the competition, it affects how quickly fire brigades and ambulances can reach a given area. Thus we need police agents to respond quickly to the needs of the other agents and at the same time to ensure that the city is cleared of blockades.

The city is partitioned into clusters, as shown earlier in Figure 2. Once the partitions are generated, each agent assigns itself, based on its unique id, in a round-robin fashion

to a partition so that each partition has at least one agent. Each agent needs to visit all the roads in its cluster to clear them. We solve this as a Travelling Salesman Problem (TSP). We first transform the road network into a graph where each vertex corresponds to a street segment and each edge to an intersection. We then convert the graph into a complete graph by using the all pairs shortest path Floyd-Warshall algorithm [12], which generates a $n \times n$ distance matrix, consisting of the shortest paths between all n vertices. We use the distance matrix as a complete graph for the Approx-TSP-Tour algorithm [12], which generates a tour whose cost is no more than twice the minimum spanning tree's weight.

In addition to following this tour, the agents respond to help requests from fire brigades and ambulances as and when they arrive. These requests are assigned by the Police Office using auctions. While responding to requests, the police agents save their previous path and return to the portion of the tour they were on after the requested blockade is cleared. This ensures that the agents complete their tour and attend to emergency requests in a timely manner.

4.2 Ambulance Agents

Ambulance agents are responsible for saving civilians that are injured or trapped. Civilians' conditions deteriorate over time, hence they have to be located and rescued as soon as possible.

Ambulance agents have two types of tasks:

1. Gather information. This involves counting the number of civilians trapped, their approximate location, and judging how soon and how much assistance they need.
2. Rescue civilians. Once a civilian is located, the ambulance has to remove the rubble trapping him, load him and take him to a refuge.

Civilians can occasionally be heard when within a hearing range, but locating them takes time because each building in their proximity has to be visited. It is often physically impossible to visit all the buildings in the timeframe given.

The location of a civilian may be known but inaccessible due to a blockade or fires in the area. Ambulances have to weigh waiting for the blockades to be cleared and fires to be put out against saving other civilians who may or may not be in as critical a condition. Coordination with other ambulances can help in speeding up the process.

Since the information available to each ambulance is insufficient to determine which civilians to rescue first, ambulances bid on tasks with the Ambulance Center, as described earlier, and also maintain a local copy of the ambulance center's probability map. Police and fire agents help by exploring buildings when their own tasks are completed. This speeds up the exploration process and allows the ambulances to concentrate on rescues instead.

4.3 Fire Brigade Agents

Fire brigades and the fire station are responsible for getting fires under control. The spread of fire depends upon wind speed and wind direction. Due to fire, buildings get damaged and collapse resulting in blocked roads. Additionally, fire can result in civilian death and agents getting hurt.

A critical responsibility of the fire agents is to maintain an accurate representation of the fire locations and intensities across the city. All field agents of each type report to the Fire Station the id and fire intensity of any building they sense is on fire, as well as the time they discovered it. The Fire Station keeps a list of all known buildings on fire, and updates each building's fire intensity as new data are collected. It uses this information to generate brigade assignments.

Since fires start in separate locations and spread outwards, the buildings on fire tend to form clusters around their respective points of origin. Since smaller clusters of less intense fires are much easier and vital to extinguish, we weigh each cluster as the sum of the squared intensities of the fires for each building in the cluster, and then invert the weights by subtracting them from the highest-weighted cluster. To ensure that each cluster gets addressed, one brigade is allocated per cluster, and the rest are assigned in proportion to the inverted weight of each cluster compared to the total weight of all the clusters. Within each cluster, brigades are assigned to first address the less intense fires using a similar weight inversion method using the square of the fire intensity.

Communication about fire locations and intensity of fires is often delayed, so the extinguish assignments may be based on slightly outdated information. Since fires grow in intensity over time, the delay often causes agents to be directed to extinguish a building where the fire is too intense. To mitigate this problem, when the brigades reach their target building, they search for nearby fires that would be extinguished more easily. This allows the agents to handle their assigned fire cluster without wasting time and water on hopelessly intense fires.

The decision-making process of the fire brigades at each timestep is as follows. The first thing a fire brigade does is look for civilians within its sensing radius. It then sends a message to ambulances indicating whether or not there are civilians in need of saving in the area. The second step is to check if the agent is currently inside a refuge. If so, it checks if its water level is less than full and then stays inside the refuge for this timestep.

The next thing the agent checks is if it is currently located on a road. If so, it checks if it has encountered a blockade, impeding its progress. If the agent has been blocked, and it has a current goal that it is working towards, the D* Lite replanning method is invoked to recompute a new path. If the agent does not currently have a goal, it simply moves into a nearby building for this timestep so it can get out of the way for police agents that may be arriving to clear the blockade.

The fourth check is whether or not the agent is currently sitting inside a building, and furthermore if that building has caught fire. If this is true, the fire brigade immediately attempts to escape from the dangerous situation. Next is a check of the agent's current health. If it detects that it is in danger of dying, it immediately moves to the nearest refuge to heal. Similarly, if it detects that its water level has reached 0, it goes to the nearest refuge in order to replenish its tank.

The last check involves looping through any commands it has received from the fire station to extinguish buildings that have caught fire. If it is close enough to the building requested, it checks the local area to determine the most easily extinguishable fire in reach, which it then extinguishes. Otherwise it plans a path to the building and moves on that path. Once the station commands have been exhausted, it loops through an internally maintained list of buildings it has seen that were on fire in a similar way.

If the decision making process falls through all of these checks, it means that the fire brigade has nothing to do. If the simulation is still early, i.e. less than 1/3 of the way through, this may just mean that the fire brigade has not explored enough. In this case, it moves randomly. However, towards the end of the simulation, this most likely means that all fires have been extinguished. If so, the fire brigade responds to requests sent by ambulances to explore unknown buildings to help discover trapped civilians.

Initially, we used A* search to plan paths for our fire brigades. Although optimal, this method failed to account for blockades and other path obstructions, and agents used a substantial amount of their processing time computing and recomputing new paths. To get better results, we have chosen to use the D*-Lite algorithm [13]. Its main focus is goal-directed navigation in unknown terrain. D*-Lite performs better than A* because it computes optimal paths (the same paths that would be computed using A*) more quickly and more efficiently by modifying previous search results locally. With A*, sometimes search would not complete in the brief amount of time that an agent had to make decisions, resulting in agents not moving at all for several cycles at a time.

5 Experimental Results

Police agents contribute to the competition score only indirectly by clearing roads for ambulances and fire brigades. Therefore, their ability to clear blockages is the best measure of their performance.

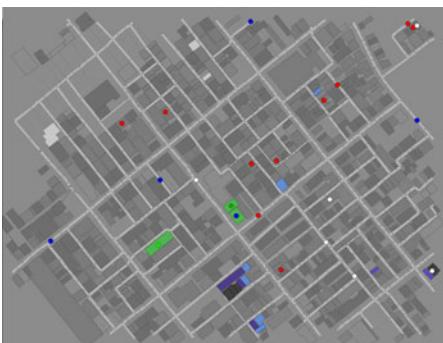


Fig. 3. MinERS at the end of simulation on Kobe. Very few buildings are burnt and no civilians are dead.

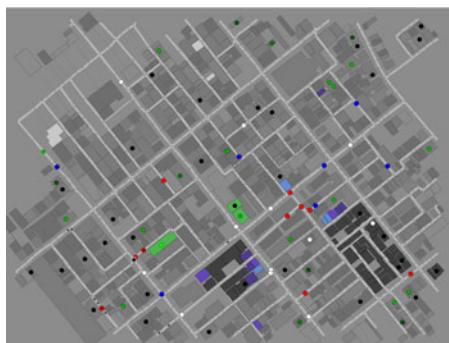


Fig. 4. Sample agents at the end of simulation on Kobe. More civilians are dead and buildings burnt.

The simulator provides baseline agents, which we refer to as “Sample”. These agents perform a random walk over the map, and handle all problems encountered while on the random walk. Figure 3 and Figure 4 show the status of the Kobe map towards the end of the simulation with MinERS and Sample agents respectively. It is clear from the figures that our fire brigade agents are able to control the spread of fire better.

Comparisons of our police agents with the Sample police agents are in Table 1. In these experiments we use only police agents and blockades. This was done in order to evaluate the performance of the police agents without interference from other agents. Typically, Sample police agents work clear blockades till the end of simulation, while the MinERS Police Agents clear the blockades much earlier.

Table 1. Number of blockages cleared by Sample and by MinERS Police Agents

Map	Sample	MinERS
Kobe	227	244
Random Small	536	600
Foligno	264	377
Virtual City	247	270

We compared the performance of MinERS vs. agents which have competed in the 2009 competition and vs. the Sample agents. We chose three other teams for comparison, two of which (MRL and Poseidon) have participated regularly in the competition and were finalists in 2009, and one (Lotus) was a new team like ours. The scores of the four teams that were in finals were very close. MRL ranked second in 2009, and Poseidon third. We could not get the code of the competition winner to run.

Table 2 shows data on the maps used, including starting scores, number of agents, and number of civilians to be rescued. The maps drive the simulation – they contain all the information about the location of agents and civilians, ignition points (the points where fire will start) and blockades, and have different challenges for the different maps.

Table 2. The different maps used in the experiments and their properties

Map	Number of						Initial Score
	Ambulances	Fire Brigades	Police	Buildings	Civilians	Fires	
Kobe	15	10	8	740	144	6	178
Random Small	7	10	11	1219	66	6	103
Foligno	8	10	10	1085	70	4	99
Virtual City	7	6	11	1271	80	5	105

We show the mean and standard deviation of the scores of all the field agents on four maps in Table 3, obtained from 30 independent trials of each agent in each map. In the table we include two version of MineERS, one without using auctions and one using auctions. We note that the performance of MinERS is markedly better than that of the Sample agents on all the maps. The ambulance agents also succeed in transporting a significantly higher proportion of the civilians to the refuge. The better performance is evident from the score difference between the maps.

Our agents perform better than the Sample agents and the Lotus team agents, however we need to make improvements to be competitive against the top agents. According

Table 3. Comparison of performance of MinERS with other agents on maps with blockades

Map	Sample	Lotus	MinERS no auctions	MinERS	Poseidon	MRL
Kobe	$\mu(\sigma)$ 90.44 (8.84)	$\mu(\sigma)$ 115.86 (3.96)	$\mu(\sigma)$ 120.08 (9.97)	$\mu(\sigma)$ 138.18 (3.54)	$\mu(\sigma)$ 154.70 (3.06)	$\mu(\sigma)$ 159.21 (4.50)
Random S.	55.58 (5.80)	65.70 (2.60)	71.55 (4.40)	80.96 (2.29)	89.70 (1.65)	77.67 (9.87)
Foligno	13.33 (1.61)	15.58 (0.82)	14.42 (0.92)	18.18 (0.68)	21.85 (0.83)	22.91 (1.37)
Virtual C.	32.72 (1.84)	41.06 (1.59)	37.31 (2.34)	48.90 (2.28)	60.14 (1.38)	63.49 (2.05)

to [9], some teams use specialized domain information, while we have kept our solutions generalizable.

We assume the scores for an agent on each map are normally distributed. We performed a one-tailed Welch's t-test (with $n = 30, \alpha = 0.001$) on our agents. We found that the difference in performance is significant at the 0.001 level (p was on the order of 10^{-11} to 10^{-27} for the four maps we studied) for MinERS using auctions compared to MinERS without using auctions.

Auctions reduce the time taken to address each task – previously multiple tasks were getting neglected while agents converged on a single task. The auction and clustering mechanism enables the centers to ensure that tasks get proportional attention and no task gets neglected. This has been especially helpful since now police agents enable fire brigades to reach fires faster when they are easier to contain, and ambulances are able to explore more of the city faster.

The second major contributor in the improved score is the searching done by the fire brigades and police agents after their tasks are completed. This speeds up the rate at which the city is explored, and resulted in a larger number of civilians being rescued. A few points to note are:

1. Thanks to communication and coordination, our fire brigade agents are able to reach all the sites where fire breaks out. On the other hand, lack of communication results in idling of Sample agents despite task availability.
2. Our police agents locate and clear all the blockades in the Kobe city map, and consistently clear more blockades than the Sample agents.
3. Our police agents respond to requests to clear blockades from other agents, allowing them to start performing their tasks early on in the simulation.
4. Both the police and fire brigade agents respond to ambulance center requests to check buildings for civilians when they have completed their tasks. This relieves the ambulance agents of extra work and accelerates discovering and locating trapped civilians.
5. The ambulance agents are able to explore all the buildings in the Kobe city map, and locate and rescue all the civilians in that map by the end of the simulation.

6 Conclusions and Future Work

We have presented solutions to task prioritization and allocation among distributed heterogeneous agents in large scale urban search and rescue. We used the RCR simulator

to develop and test our strategies for the rescue agents, and presented the results we obtained. Agents using our algorithms perform markedly better as compared to agents which work stand-alone. We have shown how the coordination algorithms we use at the center level have increased the rate at which our agents can successfully respond.

For the future, we plan on porting our algorithms to the new simulator that has been released this year and on adding further heuristics at the individual agent level. We will also be working on a closer integration of the different agent types, and to increase the interaction among the three centers.

References

1. Kitano, H., Tadokoro, S.: RoboCup rescue: A grand challenge for multiagent and intelligent systems. *AI Magazine* 22(1), 39–52 (2001)
2. Takeshi, M.: How to develop a RoboCupRescue agent (2000)
3. Paquet, S., Bernier, N., Chaib-draa, B.: Comparison of different coordination strategies for the RoboCup rescue simulation. In: Orchard, B., Yang, C., Ali, M. (eds.) *IEA/AIE 2004. LNCS (LNAI)*, vol. 3029, pp. 987–996. Springer, Heidelberg (2004)
4. Mohammadi, Y.B., Tazari, A., Mehrandezh, M.: A new hybrid task sharing method for co-operative multi agent systems. In: Canadian Conf. on Electrical and Computer Engineering, pp. 2045–2048 (May 2005)
5. Martínez, I.C., Ojeda, D., Zamora, E.A.: Ambulance decision support using evolutionary reinforcement learning in RoboCup rescue simulation league. In: Lakemeyer, G., Sklar, E., Sorrenti, D.G., Takahashi, T. (eds.) *RoboCup 2006: Robot Soccer World Cup X. LNCS (LNAI)*, vol. 4434, pp. 556–563. Springer, Heidelberg (2007)
6. Scerri, P., Farinelli, A., Okamoto, S., Tambe, M.: Allocating tasks in extreme teams. In: Proc. Int'l Conference on Autonomous Agents and Multi-Agent Systems, pp. 727–734. ACM, New York (2005)
7. Bredenfeld, A., et al.: RoboCup 2005. *LNCS (LNAI)*, vol. 4020, pp. 166–172. Springer, Heidelberg (2006)
8. Nair, R., Ito, T., Tambe, M., Marsella, S.: Task allocation in the RoboCup rescue simulation domain: A short note. In: Int'l Symposium on RoboCup (2001)
9. Ramchurn, S., Farinelli, A., Macarthur, K., Polukarov, M., Jennings, N.: Decentralised coordination in RoboCup rescue. *The Computer Journal* (January 2009)
10. Koenig, S., Tovey, C., Lagoudakis, M., Markakis, V., Kempe, D., Keskinocak, P., Kleywegt, A., Meyerson, A., Jain, S.: The power of sequential single-item auctions for agent coordination. In: Proc. Nat'l Conference on Artificial Intelligence, pp. 1625–1629 (2006)
11. Dias, M.B., Zlot, R.M., Kalra, N., Stentz, A.: Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE* 94(7), 1257–1270 (2006)
12. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: *Introduction to Algorithms*. Prentice Hall of India (2001)
13. Koenig, S., Likhachev, M.: D* lite. In: Proc. Nat'l Conference on Artificial Intelligence, pp. 476–483 (2002)

Swarm Dispersion via Potential Fields, Leader Election, and Counting Hops

Anuraag Pakanati and Maria Gini

Department of Computer Science and Engineering, University of Minnesota
pakanati@msu.edu, gini@cs.umn.edu

Abstract. We examine how robots using only laser range data and the ability to communicate within a certain radius may disperse in a complex unknown environment. We demonstrate an algorithm which attempts to maximize coverage while minimizing loss of connectivity during deployment. We demonstrate using Player/Stage that this algorithm can reliably achieve significant coverage over a variety of environments.

1 Introduction

Consider a domain such as a cave or a disaster area in which the layout is unknown ahead of time and where we need to create a wireless network in order to facilitate the deployment of men and material, or gather and transmit sensor information. Automated methods to accomplish this are highly desirable since not only they avoid placing humans in harm's way, but they allow human resources to be focused, either elsewhere or more precisely with information gathered by such a network.

Robot swarm dispersion refers to a broad class of problems in which a collection of robots starts out in a relatively compact space and must spread out in order to cover as much area as possible while preserving connectivity of the network. Different applications typically have different requirements. For example, robustness to node failure may be important in a tracking application such as radar, but less important in other applications. Metrics for the effectiveness of dispersion involve evaluating the amount of coverage achieved at a certain time, the rate of coverage increase over time, the time needed to deploy the network, robustness to node failure, and degree of overcoverage, among others.

Our robots lack GPS, maps, compass, or any form of absolute localization, but do have access to laser range-finders and wireless communication. We use a model-free approach where robots achieve coverage in a distributed fashion and do not attempt to build a map of the environment. We build upon artificial potential fields [1] and use connectivity information in order to preserve network connectivity while achieving dispersion. Our approach is empirical due to the extreme difficulty of providing theoretical guarantees on the performance of such systems. We present two major contributions:

1. a novel algorithm to enable dispersion of a group of minimally equipped robots in an unknown environment. The algorithm uses leader-election, hop

- counting, and potential fields to maximize coverage while maintaining the robots within communication range of each other.
2. extensive experimental results obtained in simulation in a variety of environments. The results show the effectiveness of the algorithm in achieving a significant coverage while maintaining connectivity.

In Section 2 we discuss relevant background literature. We present our algorithm in Section 3, followed in Section 4 by experimental results, where we measure the performance of our algorithm in four simulated environments. Finally, in Section 5 we wrap up with conclusions and discussion of future work.

2 Related Work

A common way of achieving robot dispersion is to use artificial potential fields [1] to repel robots away from obstacles and attract them to their goal. There are many variations in the way artificial potential fields are used, but all approaches assume that the relative locations (distance and bearing) of obstacles is available to each robot through its sensors or communication system.

Dispersion in an unknown environment was tackled in [2] using various behaviors [3] to achieve dispersion but without regard for network connectivity. In [4] robots equipped with 360° laser range-finders use potential fields and a viscous friction term in order to assure that the network converges. This is similar to our work, since we also use 360° lasers and potential fields, but we use communication and connectivity to prevent disconnection of the network. In [5] mobile nodes are treated as charged particles with two forces, one repulsive that tries to maximize coverage, the other attractive that attempts to maintain for each node not on the boundary at least one neighbor in every θ sector of communication range. This requires each robot to know the relative position of its neighbors. There are only convex obstacles to limit the ability of the nodes to move, so the approach does not work for deploying a network in a building.

To maintain connectivity in [6] robots are deployed sequentially using various heuristics to select the place for the next robot. Because of the sequential deployment the process is very slow. The LRV algorithm [7] uses a single robot carrying around network nodes in an unexplored area and deploying them. The deployment is again sequential, one sensor at the time. A dispersion algorithm that uses virtual pheromones similar to those in [8] enables small robots to explore [9]. The communication system provides range and bearing to any robot in the communication range, and the space is open compared to the robot size.

Our work most closely resembles the clique algorithm [10], which uses wireless intensity signals to prevent robots from running out of reach from each other. Robots share connectivity information with each other so that they can all agree on which robots should act as “sentries” and not move, allowing the other robots to move. Additionally, we explored some of the Neighbor-Every-Theta connectivity ideas in relation to potential fields [11, 12], although we assume different capabilities which alters how those ideas are applied. Lastly, we use leader-election and hop-counting to help guarantee connectivity.

3 Dispersion Algorithm

Our algorithm combines artificial potential fields and behaviors with selecting a leader of the swarm, counting hops from the leader, and sending alarms to prevent disconnection from the network.

3.1 Artificial Potential Fields

In our artificial potential fields we use three forces. The first, and largest, is a *radial force* \mathbf{F}_R whose magnitude is determined by distance to obstacles. We use a combination of linear and exponential magnitudes to determine the force, as shown in (1). Here, \mathbf{p} is a laser measurement corresponding to a point mass (obstacle) along a vector, $d(\mathbf{p})$ is the distance of that point mass and $m(\mathbf{p})$ is the magnitude of the resulting force exerted on the robot by that point, I is an indicator function which evaluates to 1 if its condition is true and 0 otherwise, and k is a constant (4 throughout our experiments).

$$\mathbf{F}_R = \sum_{\mathbf{p} \in P} -m(\mathbf{p}) \frac{\mathbf{p}}{\|\mathbf{p}\|}, \text{ where } m(\mathbf{p}) = k/d(\mathbf{p}) + I_{d(\mathbf{p}) \leq 1} e^{-d(\mathbf{p})} \quad (1)$$

The second force is an *open force* \mathbf{F}_O that pulls a robot towards empty regions. It is generated by a two pass algorithm which first identifies in the range-finder data consecutive readings above a given threshold and creates an attractive force towards the largest opening found. Let \mathbf{O} be the collection of all openings and \mathbf{O}_C the vector pointing outwards towards the center of the open area, then:

$$\mathbf{F}_O = \max_{O \in \mathbf{O}} |O| \times O_C \quad (2)$$

Finally, a *tangential force* is generated from each pair \mathbf{p} and \mathbf{p}' of successive point readings to make the robot follow the wall. There are two candidates with opposite directions. The ambiguity is broken by using the one with the smaller angle relative to \mathbf{F}_R . The magnitude of $\mathbf{f}_T(\mathbf{p}, \mathbf{p}')$ is determined by the closer of the two points, and is smaller than \mathbf{F}_R . The total tangent force \mathbf{F}_T is simply this force calculated over every pair of successive readings, \mathbf{p} and \mathbf{p}' :

$$\mathbf{F}_T = \sum_{\{\mathbf{p}, \mathbf{p}'\} \in P} \mathbf{f}_T(\mathbf{p}, \mathbf{p}') \quad (3)$$

where, as illustrated in Figure 11:

$$\mathbf{f}_T(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} - \mathbf{B}}{\|\mathbf{A} - \mathbf{B}\|} \cdot \text{sign}((\mathbf{A} - \mathbf{B}) \circ \mathbf{F}_R) \cdot \frac{\max(m(\mathbf{A}), m(\mathbf{B}))}{4} \quad (4)$$

Potential fields offer no innate support for preserving connectivity between robots. Potential fields also lend themselves to a few problems. The first issue we discovered in our experiments was that the runs were very sensitive to minor random fluctuations. Despite having the same initial conditions for each run, the final deployed networks were very different in both positioning for each

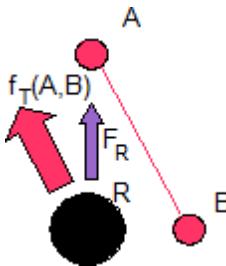


Fig. 1. Tangent force, \mathbf{f}_T , purple arrow, determined by two point masses **A** and **B**

robot, and frequently in the overall pattern of dispersion, due to unpredictable multithreading effects. The overall coverage and connectivity did not drastically change from run to run, but the networks in existence in the end were qualitatively different. Another issue is that robots can become stuck in suboptimal positions, causing the entire system to also become stuck in a suboptimal configuration. Less frequent, but still possible, is the occurrence of periodic oscillations in robot actions. Finally, a small fluctuation in one robot's position can cause other robots to adjust their own positions and this can ripple through the entire network. We found that incorporating a *dead-zone* in which a robot is less or non-responsive to changes limits these effects since minor changes in a neighbor's position will not necessarily force a reassessment, which in turn damps the effects of minor perturbations on the network as a whole.

3.2 Counting Hops with Alarms

There are two phases in our algorithm. In the first phase the robots select a common reference point, or leader, and in the second phase they use the leader to coordinate movement. We assume that each robot has a unique ID and that the robot with ID 1 has been elected as the leader. To select a leader a $O(r)$ algorithm is sufficient to find the robot with the lowest ID, where r is the number of robots. This is achieved by each robot broadcasting the minimum of its own ID and the lowest ID that it has ever received. After r iterations, the entire (connected) network has the same ID as the lowest, which becomes the leader.

Once the leader is established, robots count hops to the leader. A neighbor hop count of zero is assigned to the leader itself. Other robots calculate their own hop count by measuring broadcasts from neighbors and assigning themselves a hop count corresponding to the lowest neighbor hop count plus one, as in equation 5, where H_R is the hop count of robot R , and N_R are its neighbors:

$$H_R = \min_{N \in N_R} (H_N) + 1 \quad (5)$$

Figure 2 demonstrates how counting hops works. While moving, each robot should maintain at least one neighbor with a lower hop count. The leader does not move and provides an anchor for the entire network. This reduces disconnections in the network, but it is still possible for robots to drift apart, severing

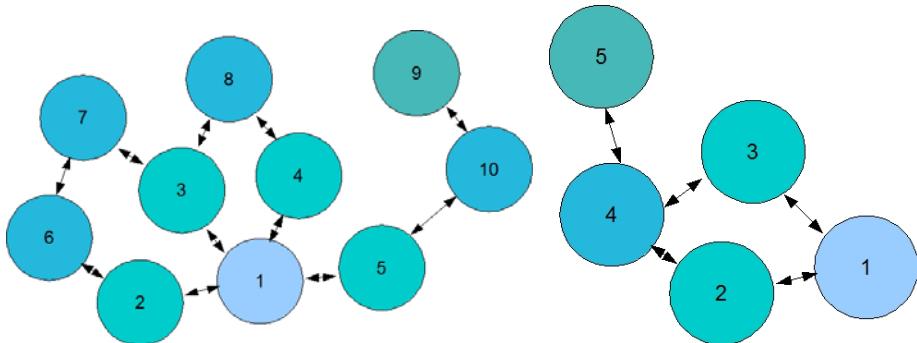


Fig. 2. Counting hops. The leader is Robot 1. **Fig. 3.** Robot 5 stays put since Robots 2, 3, 4, and 5 are one hop away. Robots 6, 7, 8, and 10 are two hops away. Robot 9 is three hops away.

Fig. 3. Robot 5 stays put since its only neighbor with a lower hop count is Robot 4. Robot 4 has two neighbors (2 and 3) with a smaller hop count so it may move and disconnect Robot 5.

the network, as shown in Figure 3. To avoid splitting the network, we add alarm messages. A robot sends an alarm if and only if there are no robots with a lower hop count within a given threshold of its communication range. We use .8 of the maximum communication range as the threshold. This keeps the robots together and in our experiments prevented the network from splitting. An alarm has a target – the target must immediately stop and freeze. This is the algorithm each robot uses:

Algorithm 1. Alarm Counting Hops – Robot State Decision Flow

```

1: procedure ROBOTDECISIONLOOP
2:   if robot has  $\geq 1$  neighbor broadcasting an alarm signal then
3:     State  $\leftarrow$  Freeze
4:   else if State == Freeze and robot has  $> 1$  neighbor with lower hops then
5:     State  $\leftarrow$  Scatter
6:   else if robot has  $\leq 1$  neighbor with a lower hop count then
7:     robot sends alarm to neighbor with lowest hop count & lowest ID.
8:     State  $\leftarrow$  Freeze
9:   else if robot is receiving an alarm then
10:    State  $\leftarrow$  Freeze
11:   else if robot is stuck then
12:     20% chance State  $\leftarrow$  Backup
13:     20% chance State  $\leftarrow$  RandomTurn
14:     20% chance State  $\leftarrow$  Forward
15:     20% chance State  $\leftarrow$  Random
16:     20% chance State  $\leftarrow$  Scatter
17:   end if
18: end procedure
```

The robot behaviors are:

- *Random*: robots pick a random turn velocity and forward velocity.
- *RandomTurn*: robots reorient themselves in place for 5 seconds picking a random direction to rotate.
- *Scatter*: robots follow the forces in the artificial potential field.
- *Forward*: identical to *Scatter*, except the resultant vector adds a strong positive vector in the current robot direction. This has the effect of influencing the robot to continue moving in the direction it is facing.
- *Backup*: also identical to *Scatter*, except the resultant vector adds a strong vector in the opposite direction of the current robot direction.
- *Freeze*: is triggered when either a robot is sending an alarm, or another robot is sending an alarm to it.

The initial behavior for all robots is *Scatter*. When a collision occurs or a robot gets stuck, the robot randomly picks another behavior, pursues it for a few seconds, and keeps on picking a new behavior until it is once again in *Scatter*. The exception is *Freeze* which instructs the robot to cease moving. The leader stays in *Freeze* and does not move, thereby providing an anchor for the network.

4 Experiments

We utilized the Player/Stage [11] robot simulation environment for our experiments using virtual Pioneer robots. Robotic controllers were implemented in python, using the SWIG python interface to the native C Player/Stage libraries. We assume that the intensity of the wireless signal is proportional to the inverse square of distance, with uniform degradation of signal. We also assume robots can identify the source of a signal by ID, and that a robot's signal cannot be perceived outside of its communication radius.

In order to test the generalizability of our approach, we used four different worlds: cave, autolab, hospital, and house. The first three are distributed with the Player/Stage project. The last is a custom drawing of a house. The environments vary in complexity with the cave and autolab environments being the easiest, the house environment being slightly more difficult, and the hospital being the most difficult due to the many rooms. The starting locations of the robots also have an effect on the complexity of the dispersion problem—some arrangements are easier to disperse from effectively than others. The robots were placed in a tightly packed area and in such a way that the network is connected. The connectedness is a requirement for this algorithm to work, as it does not inherently contain any method for repair, except that a robot can reconnect via random wandering.

4.1 Theoretical Maximum Coverage

Figure 4 shows the minimum overlap of two robots which are in direct communication range. The edges of each of their wireless ranges, depicted by the circles, must include the other robot, which corresponds to the center of the

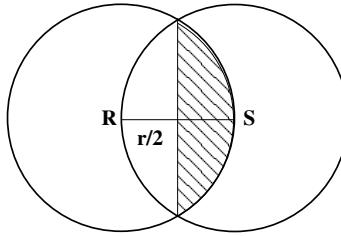


Fig. 4. Robots R and S in communication range

other circle. Now assume that each robot has a wireless range with a radius of r meters (in our experiments we use $r = 2$). The minimum region of overlap between two circles is then twice the shaded area $2(\frac{1}{3}r^2\pi - \frac{\sqrt{3}}{4}r^2) = (\frac{2}{3}r^2\pi - \frac{\sqrt{3}}{2}r^2)$. For n robots in a connected network, there must be at least $n - 1$ robots mutually in connection range (just as when a connected graph has V nodes, there must be at least $V-1$ edges). Thus, the minimum possible overlap is necessarily $(n-1)(\frac{2}{3}r^2\pi - \frac{\sqrt{3}}{2}r^2)$. It follows then the maximum possible coverage for n robots is $n\pi r^2 - (n-1)(\frac{2}{3}r^2\pi - \frac{\sqrt{3}}{2}r^2)$. Note that this network corresponds to a linear chain of robots. While this maximizes coverage, it is not necessarily feasible in any given environments, nor is it even desirable. One severe drawback is the brittleness of this network – the loss of any robot besides the two on the ends would fracture the network into two pieces. Nonetheless, it serves as an upper bound on the achievable coverage.

4.2 Computing Bounds with Incremental Greedy Algorithm

Since the theoretical upper bound is too lose and not achievable, we implemented an incremental greedy algorithm, similar to the one in [6], but, for simplicity of implementation, we place each robot at its desired location, instead of making the robot to navigate to reach its destination. Algorithm 2 places sequentially each robot at a legal location within wireless range. The location is chosen to maximize the total visual area seen by the robot swarm while keeping connectivity.

Note that the results depend on the sampling distance of points to produce C. We experimented with different values, from every 32 cm down to 8 cm. The spatial configurations of the networks obtained, shown in Figure 5, are noticeably different, despite the same initial conditions. Figure 6 shows the results in a more complex environment. Presumably, continuing to increase the sample rate would yield limiting behavior, but since the run time is quadratic in the sampling rate, this limit is expensive to find. Also since the algorithm is greedy, choosing an initially profitable location might ultimately lead to a suboptimal placement. With a sampling rate of 8 cm and with optimizations to reduce repeated sampling it took about 8 hours on a 3.6GHz computer to deploy 24 robots. Once again, our goal is to find an upper bound and provide a metric of reference.

Algorithm 2. Incremental greedy algorithm

```

1: procedure INCGREEDY( $\mathbf{R}, x_0, y_0$ )
2:    $\mathbf{R}(1) \leftarrow (x_0, y_0)$ ; ▷ position of first robot
3:    $\mathbf{C} \leftarrow$  Visual Coverage of  $\mathbf{R}(1)$ ;  $\mathbf{W} \leftarrow$  Wireless Coverage of  $\mathbf{R}(1)$ ;  $r \leftarrow 2$ 
4:   while  $r \leq |R|$  do ▷ for each robot
5:      $\mathbf{C}_{best} \leftarrow \mathbf{C}$ ;  $Area_{best} \leftarrow 0$ 
6:     for  $w = (w_x, w_y) \in \mathbf{W}$  do
7:        $\mathbf{C}_{cur} \leftarrow$  Visual Coverage of  $\mathbf{R}(1), \dots, \mathbf{R}(r)$ ;  $Area \leftarrow sum(C_{cur})$ 
8:       if  $Area > Area_{best}$  then
9:          $\mathbf{C}_{best} \leftarrow \mathbf{C}_{cur}$ ;  $\mathbf{R}_{best} \leftarrow (w_x, w_y)$ 
10:         $\mathbf{W} \leftarrow$  Wireless Coverage of  $\mathbf{R}(1), \dots, \mathbf{R}(r)$ 
11:      end if
12:    end for
13:     $\mathbf{R}(r) \leftarrow \mathbf{R}_{best}$ ;  $\mathbf{C} \leftarrow \mathbf{C}_{best}$ ;  $r \leftarrow r + 1$ 
14:  end while
15:  return  $\mathbf{R}, \mathbf{C}$  ▷ robot positions are in  $\mathbf{R}$  and coverage is in  $\mathbf{C}$ 
16: end procedure

```

Figure 7 shows the coverage obtained after the placement of each successive robot. Note that although the values are not directly comparable due to differences in the initial robot placement and the environment, the more constricted the environment, the lower is the final coverage obtained. As the complexity of the environment increases, the solution obtained by the incremental greedy algorithm becomes unrealistic. This is because the algorithm may skip over walls and other barriers when placing robots, which may make it impossible to replicate the results with real robots. A location may be physically accessible, but not realistically achievable if a long circuitous path is required. The overriding purpose here is to get a deterministic bound on the maximum achievable coverage.

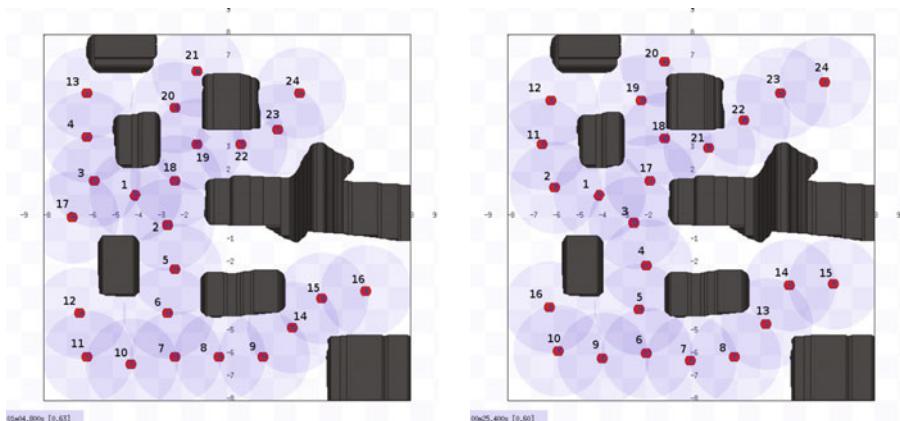


Fig. 5. Cave– Incremental greedy algorithm for different sample distances



Fig. 6. Hospital Section– Incremental greedy algorithm, .08 sample distance

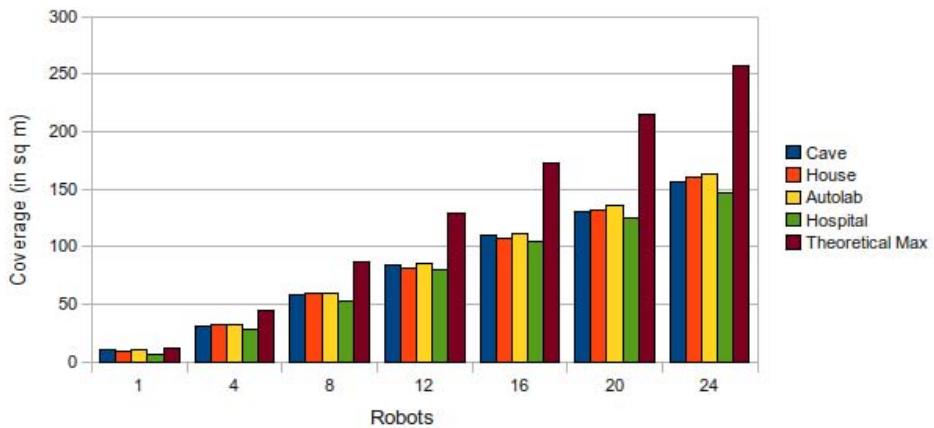


Fig. 7. Coverage of incremental greedy algorithm for increasing numbers of robots in different environments

4.3 Results on the Counting Hops with Alarms Algorithm

We performed three set of experiments of the Hop Counting with Alarms algorithm. The first was to measure sensitivity caused by minor changes in robot position, the second to measure performance on different environments, and the last to measure the effects of reducing the number of laser readings.

1. Sensitivity analysis. To measure sensitivity to initial positions we compared the results of 20 runs with fixed initial positions against 20 runs with randomized initial positions of 24 robots in the cave environment. The fixed positions coverage increases from a starting coverage of $35.372m^2$ to an average of $81.443m^2$. After 5 minutes, the robots covered $74.002m^2$, and 10 minutes, $78.182m^2$. Thus roughly 83.85% of the expansion is done by 5 minutes, and roughly 92.92 by 10

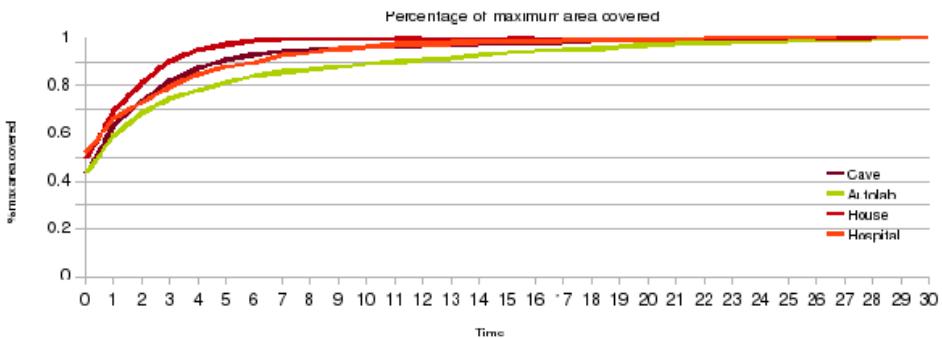


Fig. 8. Coverage over time in different environments as percentage of the coverage achieved in 30 minutes (average of 20 runs)

minutes. The final total fell well short of the maximum coverage predicted by the incremental greedy algorithm, obtaining $81.433m^2 / 156.5972m^2$ in the end, or 52.00% of the possible expansion. Again though, the incremental algorithm is not actually possible in a real deployment.

With random placement the coverage increases from a starting coverage of $35.372m^2$ to an average of $82.244m^2$. After 5 minutes, the average coverage was $74.775m^2$, and 10 minutes, $79.05m^2$. Thus 84.07% of the expansion is done by 5 minutes, and by 10 minutes roughly 93.23% of the final coverage is completed. There was little or no difference between the fixed and randomized cases, which made us to conclude that at least in terms of aggregate behavior, the algorithm was not sensitive to minor positional variations.

2. Performance in different environments. The second set of experiments included twenty runs of each of the autolab, house, and hospital environments mentioned earlier. The cave and autolab environments were the more forgiving, while the house and especially the hospital proved more difficult because robots had to leave or enter rooms to maximize coverage. To emphasize the progress with time, the results in Figure 8 show the percentage of the coverage achieved after 30 minutes, since at that point the coverage had plateaued.

In the house environment coverage increased from $37.792m^2$ at the start to $73.995m^2$ after 5 minutes. $75.867m^2$ after 10 minutes, and $76.128m^2$ after 30 minutes. Thus, by 5 minutes, 94.44% of the expansion was complete; by 10 minutes, 99.3% was finished. In this environment, the robots spread out almost entirely in the original room, and ignored the adjoining room. The final coverage of $76.128m^2$ (confidence interval at 95% is (71.01, 81.3)) was also short of the $160.27m^2$ obtained by the incremental greedy algorithm. In the autolab environment, the final dispersion achieved a coverage of roughly $99.343m^2$. After 5 minutes, the dispersion was $80.791m^2$, which represented 66.96% of the total dispersion; after 10 minutes, the total dispersion was roughly $88.564m^2$ (confidence interval at 95% is (77.65, 104.37)) which was 80.98% of the total dispersion. The hospital environment highlighted the inadequacies of the algorithm in handling narrow doorways. The robots in the hallway did manage to

spread out, but robots in the room for the most part never managed to escape the room. Frequently, the doorway would become blocked during the course of a run, and that would prevent robots from leaving. The final expansion was on average around $59.205m^2$, after 30 minutes, starting from an initial coverage of 30.833. After 5 minutes, the algorithm covered $51.95m^2$, representing 74.43% of the total expansion. After 10 minutes, the average coverage was $57.008m^2$, which represented an average of 92.26% of the total expansion achieved by the algorithm. The total expansion that the algorithm achieved, $59.205m^2$ (confidence interval at 95% is (48.33, 70.03)) was also well short of the maximum $146.759m^2$ that the incremental greedy algorithm obtained in this environment, in large part because of the inability to get many robots out of the starting room.

The algorithm had similar characteristics across all environments. Most of the coverage was obtained in the first 5 to 10 minutes, with very small gains in further coverage after that point. The algorithm did not do well in complex environments with doorways and enclosed rooms—the hospital environment in particular was quite difficult. As mentioned before, robots already in the hallway fared well, while robots in the room were unsuccessful in leaving the room. Overall the ratio of coverage of our algorithm to the incremental greedy algorithm is 0.53 for the cave, 0.61 for the autolab, 0.48 for the house, and 0.4 for the hospital.

3. Dispersion with fewer laser measurements. The last set of experiments was with reduced numbers of laser readings on each robot. We tried several different conditions, from the 360 measurements we used in earlier experiments down to 180, 90, 45, 20, and 6. In each case the measurements are equidistant. The algorithm is fairly robust to a reduction in resolution; only when the number of laser readings was reduced to 6 was there a significant decline in observed coverage. However, even in that case, the robots were able to disperse to cover 183.42% of their initial coverage indicating that even very limited robots may succeed at dispersing, although not as much as with denser sensor readings. Confidence intervals at 95% range from (66.97, 99.23) for 180 measurements to (50.37, 79.43) for 6 measurements.

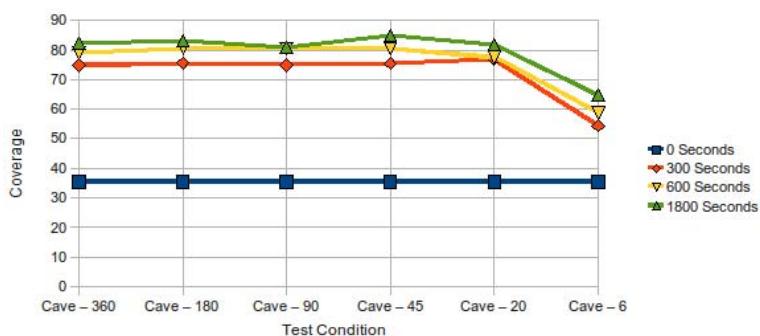


Fig. 9. Coverage with fewer laser range-finder readings in the cave environment

5 Conclusions and Future Work

We have proposed a new algorithm for robotic dispersion which uses leader-election, counting hops, alarms, and potential fields to disperse robots and create a mobile sensor network. We have demonstrated within Player/Stage that this approach achieves a significant coverage, while maintaining connectivity, requiring robots with only wireless communication and a laser range-finder. We have also shown that the approach is robust to changes in initial conditions, that it can be applied effectively to robots with range-finders with significantly reduced resolution, and that it works in different types of environments.

There are new and exciting directions for future research. A systematic approach to set values of parameters, such as that provided by a learning algorithm or an evolutionary approach, would likely improve the performance across environments. Another extension is to incorporate visual data to enable robots to move more intelligently. Currently robots do not distinguish robots in their visual field from other obstacles. This would help deal with some of the issues in narrow openings and help robots to spread out in a more coordinated fashion. Finally we would like to implement this approach with real robots.

References

1. Khatib, O.: Real-time obstacle avoidance for manipulators and mobile robots. *Int. J. Rob. Res.* 5(1), 90–98 (1986)
2. Morlok, R., Gini, M.: Dispersing robots in an unknown environment. In: Proc. Int'l Symp. on Distributed Autonomous Robotic Systems (2004)
3. Brooks, R.A.: A robust layered control system for a mobile robot. *Journal of Robotics and Automation RA* 2(1), 14–23 (1986)
4. Howard, A., Matarić, M.J., Sukhatme, G.S.: Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem. In: Proc. Int'l Symp. on Distributed Autonomous Robotic Systems, pp. 299–308 (2002)
5. Poduri, S., Pattem, S., Krishnamachari, B., Sukhatme, G.S.: Using local geometry for tunable topology control in sensor networks. *IEEE Trans. on Mobile Computing* 8(2), 218–230 (2009)
6. Howard, A., Matarić, M.J., Sukhatme, G.S.: An incremental self-deployment algorithm for mobile sensor networks. *Autonomous Robots* 13(2), 113–126 (2002)
7. Batalin, M., Sukhatme, G.S.: The design and analysis of an efficient local algorithm for coverage and exploration based on sensor network deployment. *IEEE Trans. on Robotics* 23(4), 661–675 (2007)
8. Payton, D., Daily, M., Estowski, R., Howard, M., Lee, C.: Pheromone robotics. *Autonomous Robots* 11(3), 319–324 (2001)
9. McLurkin, J., Smith, J.: Distributed algorithms for dispersion in indoor environments using a swarm of autonomous mobile robots. In: Proc. Int'l Symp. on Distributed Autonomous Robotic Systems (2004)
10. Ludwig, L., Gini, M.: Robotic swarm dispersion using wireless intensity signals. In: Proc. Int'l Symp. on Distributed Autonomous Robotic Systems, pp. 135–144 (2006)
11. Gerkey, B., Vaughan, R., Howard, A.: The Player/Stage project: Tools for multi-robot and distributed sensor systems. In: 11th Int'l Conf. on Advanced Robotics (2003)

Compliant Robot Actuation by Feedforward Controlled Emulated Spring Stiffness

Katayon Radkhah, Stefan Kurowski, Thomas Lens, and Oskar von Stryk

Technische Universität Darmstadt, Department of Computer Science

Simulation, Systems Optimization and Robotics Group,

Hochschulstrasse 10, 64289 Darmstadt, Germany

{radkhah, kurowski, lens, stryk}@sim.tu-darmstadt.de

<http://www.sim.tu-darmstadt.de>

Abstract. Existing legged robots lack energy-efficiency, performance and adaptivity when confronted with situations that animals cope with on a routine basis. Bridging the gap between artificial and natural systems requires not only better sensorimotor and learning capabilities but also a corresponding motion apparatus and intelligent actuators. Current actuators with online adaptable compliance pose high requirements on software control algorithms and sensor systems. We present a novel feedforward trajectory shaping technique that allows for a virtual stiffness change of a deployed series elastic actuator with low energy requirements. The performance limits of the approach are assessed by comparing to an active and a passive compliant methodology in simulation. For this purpose we use a 2-degrees-of-freedom arm with and without periodic load representing a 2-segmented leg with and without ground contact. The simulation results indicate that the approach is well suited for the use in legged robots.

Keywords: locomotion, gait transition, feedforward control, spring stiffness, compliance, series elastic actuation.

1 Background

A key prerequisite of versatile and energy-efficient legged robots that move in a-priori unknown environments are proper actuation modules. Recent research has focused more and more on actuators with adaptable compliance that can change joint stiffness in order to adjust the overall leg properties with respect to robustness, energy efficiency and speed of motion.

One way to vary the compliance of an actuator is by software, i.e. impedance control of a stiff actuator. Based on the measurement of the external force or torque, the controller of the stiff actuator can mimic the behavior of a spring-damper system. This type of compliant actuators requires actuators, sensors, and controllers that are all fast enough for the target application in order to permit virtual compliance adjustment during operation. The programming of the characteristic of an imitated spring and its online adjustment is also known as active compliance [8]. The disadvantage of such actuators is the continuous energy dissipation since no energy can be stored in the actuation system. Furthermore, fast shocks can not be absorbed because of limited bandwidth of the controller.

The use of springs in legged locomotion is generally accepted as important and has been promoted very early [4]. Elasticity of legs, partially storing and releasing energy during contact with the ground, allows to achieve stable, rapid and energy-efficient locomotion. In fact, mechanical elasticity is a prerequisite for ballistic human- and animal-like movements. Passive compliance actuators contain at the minimum an elastic element. Their designs are divided into four groups: (1) equilibrium-controlled stiffness, (2) antagonistic-controlled stiffness, (3) structure-controlled stiffness, and (4) mechanically controlled stiffness [1]. A famous example of the first group is the original series elastic actuator (SEA) [5], a (stiff) rotary joint actuator in series with a spring. The compliance of the actuator is limited by the spring constant and is therefore not adjustable during operation. Easy force control is enabled by measuring the spring elongation and returning in a feedback loop.

This short foray into current actuation mechanisms and techniques reveals prevalent difficulties. Actuators require complex software control algorithms and sophisticated sensor systems in order to behave adaptable and compliant in contact with unknown environments. An actuation unit that can reach the performance of the biological muscle and its neuro-mechanical control system is missing. On the other hand, actuation mechanics and principles strongly depend on the application.

In this paper we present a new, basic but effective technique enabling variable compliance. We combine energy storage and adaptable compliance by using elastic elements to store energy and applying a method to change the compliance during operation. The actuation mechanism and technique are explained in Section 2. In Section 3 the proposed methodology is compared to purely active and passive compliant actuation using a 2-degrees-of-freedom (DOF) arm in a simulation study. The paper concludes with a discussion on the advantages and disadvantages of the presented actuation techniques with respect to legged locomotion.

2 Emulated Spring Stiffness

2.1 Actuation Mechanism

Joints are actuated by bionic drives consisting of a DC motor that is elastically coupled to the joint with antagonistic, elastic pulleys with progressive angle-torque characteristics, as illustrated in Fig. 1. This actuation module has been tested extensively in a real manipulator, the *BioRob* arm [3], and simulated four-legged robot [6] and falls into the category of SEAs. Compared to the original SEA [5], however, it allows the actuation of distal joints by means of the antagonistic pulleys resulting in low mass and inertia of the joint [9]. It also enables pretension of a joint. For the remaining of the paper and our experiments, however, we make use of rotary SEAs without encoders at the joints.

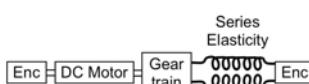


Fig. 1. Schematic of the actuation module, used in the *BioRob* arm [3] and a four-legged robot [6]

2.2 Technique

The deployed spring in each actuator has a predefined mechanical constant stiffness, therefore the physical compliance can not be adjusted during operation. But a dynamic adjustment of the equilibrium position of the spring, i.e. a different motor triggering, leads to a dynamic change in joint trajectory resulting in a different “virtual stiffness” of the actuator. We assume that the system is fed with a sine signal as reference input, similar to a pendulum movement between two angles, as shown in Fig. 2. Furthermore we assume a linear characteristic curve for the spring deployed in the actuator. We then use the following equations to correct the input signal such that the response of the system is adapted to the desired stiffness:

$$\rho_1 = \frac{\phi_1}{2} + \frac{\phi_2}{2} + \frac{\mu \cdot |\phi_1 - \phi_2|}{2 \cdot k} + o, \quad (1)$$

$$\rho_2 = \frac{\phi_1}{2} + \frac{\phi_2}{2} - \frac{\mu \cdot |\phi_1 - \phi_2|}{2 \cdot k} + o, \quad (2)$$

where k represents the mechanical stiffness of the deployed spring and μ the desired virtual spring stiffness. The variables ϕ_1 and ϕ_2 denote the two input angles between which the movements are oscillating whereas ρ_1 and ρ_2 stand for the novel, corrected angles due to the adjusted spring stiffness.

In reality, though, the linear spring does not represent the actual spring characteristic. Note that the system is not feedback-controlled, which also means, that dynamic movements of other joints affecting the positions of coupled joints are not taken into consideration by the controller. Consequently, a system that is feedforward controlled with these equations will produce deviations. Therefore, a manually tuned correction offset o is introduced in the Eqs. 1 and 2 to reduce the deviations.

The corrected sine trajectory with the upper and lower limits ρ_1 and ρ_2 based on the desired virtual spring stiffness parameter μ equals the motor position trajectory which generates spring torques τ_k and damping torques τ_d on the spring of the actuation module. The actuated revolute joint responds solely by mechanical feedback. For better clarification, this above described feedforward controlled emulated spring stiffness technique is also illustrated in Fig. 3.

3 Comparison of Different Techniques for Adaptable Compliance

Both concepts of active compliance and passive compliance are integrated in the proposed technique. Consequently, in order to assess the performance limits of our approach, we compared it to active compliance as realized in the DLR lightweight arm Justin [8] and passive compliance as realized with the mechanically controlled stiffness actuator Maccepa [2]. Since active compliance requires exact knowledge of the model, we modeled a basic 2-DOF arm that is used in the following for all comparisons. In order to apply active compliance, we also developed the inverse dynamics model of the 2-DOF arm. For replicability of the simulation results we listed the exact model parameters in Table 1. The 2-DOF arm, as depicted in Fig. 4, can also be considered as a 2-segmented leg. Ground contact is simulated by an additional periodical load at the end of the arm.

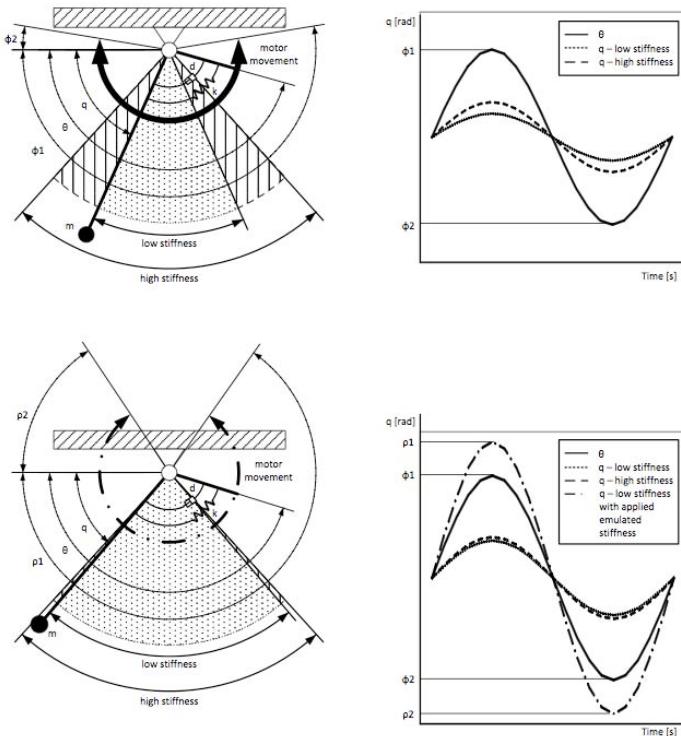


Fig. 2. Upper picture illustrates the resulting trajectories of a 1-DOF pendulum with a rotary SEA with low and high mechanical constant stiffness. A pendulum oscillates between the angles ϕ_1 and ϕ_2 . Stiffening the spring in the actuator leads to an extension of the movements, a higher amplitude. In the below picture the emulated spring stiffness technique is applied. As shown, even the system with the low mechanical stiffness can extend its movements by means of the emulated spring stiffness technique to the interval $[\rho_1, \rho_2]$.

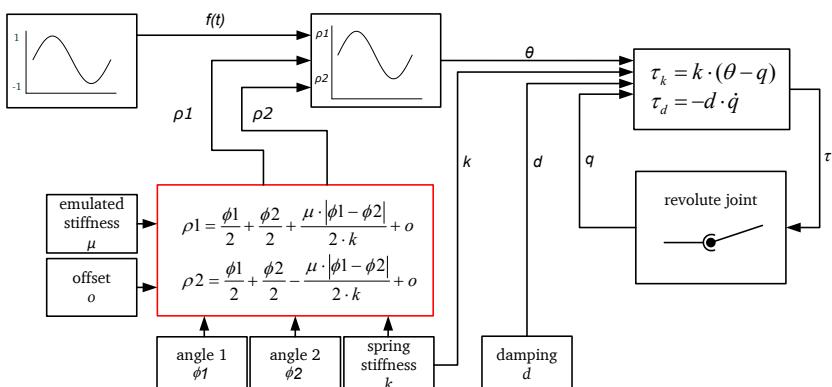


Fig. 3. Schematic of the feedforward controlled emulated spring stiffness technique

Table 1. Model Parameters of the 2-DOF arm

link length	L_1	0.3 m
	L_2	0.3 m
link mass	m_1	0.2 kg
	m_2	0.2 kg
link radius	r_1	0.02 m
	r_2	0.02 m
link inertia	I_L	$E \cdot (4 \cdot 10^{-5}, 1.5 \cdot 10^{-3}, 1.5 \cdot 10^{-3})^\top$
motor inertia	I_M	$0.33 \frac{\text{kg}}{\text{m}^2}$
gear inertia	I_G	$0.07 \frac{\text{kg}}{\text{m}^2}$

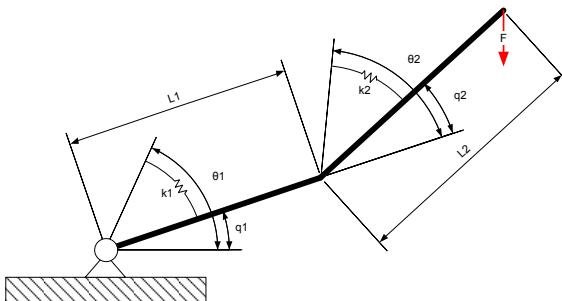


Fig. 4. Both joints of the 2-DOF arm are actuated by rotary SEAs. The variables θ_1 and θ_2 represent the motor positions while q_1 and q_2 stand for the joint positions. k_1 and k_2 denote the predefined mechanical spring stiffness of the SEAs.

The experiments are set up in the numerical computing environment Matlab. We performed different simulation runs, varying the “step frequency” of the 2-segmented leg and simulating an additional periodical ground. Due to the limited number of pages we will only focus on important aspects found in the simulation runs.

3.1 Experiment 1 with Step Frequency $f = 0.5 \text{ Hz}$

The data input for this experiment is illustrated in the two uppermost plots in Fig. 5. Both joints of the 2-DOF arm are fed with the same sine wave with the frequency $f = 0.5 \text{ Hz}$. The mechanical and emulated stiffness line overlap, i.e. emulated stiffness changes as the mechanical stiffness does. The stiffness is linearly changed from $5 \frac{\text{Nm}}{\text{rad}}$ to $14 \frac{\text{Nm}}{\text{rad}}$ during the simulation time of 9 s. Furthermore, the correction offset and the occurrence of ground contact, signalized by load, are displayed. This first run was conducted without ground contact.

Note that the deviations, i.e. squared errors, shown in the figures are all based on the so-called “ideal model”. The ideal model represents a system with “online adaptable spring stiffness” without any prerequisites such as special hardware or computing power. We assume that in simulation we can change the mechanical spring stiffness at

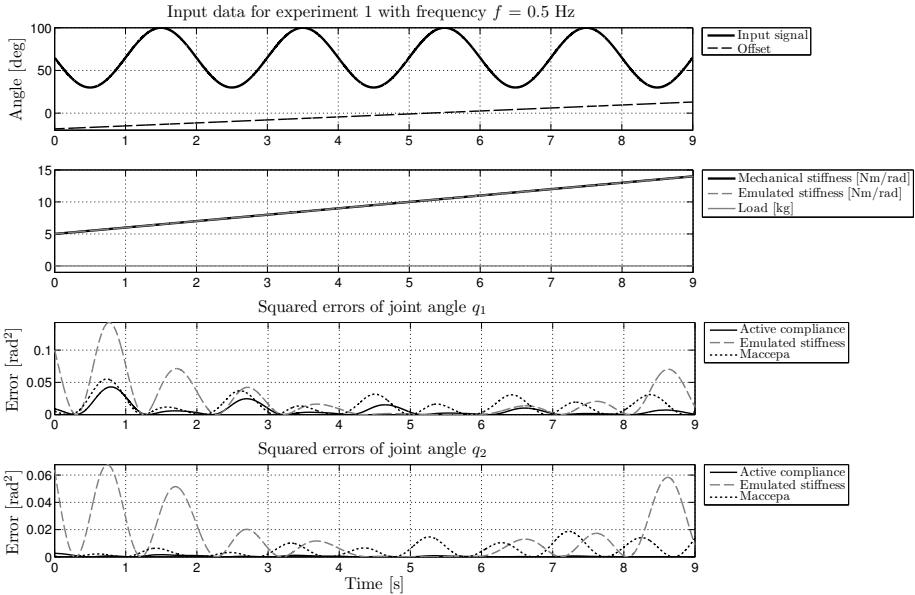


Fig. 5. Experiment 1: Sine wave as reference input for both joints with given emulated and mechanical spring stiffness with a step frequency of 0.5Hz. The experiment is performed without ground contact.

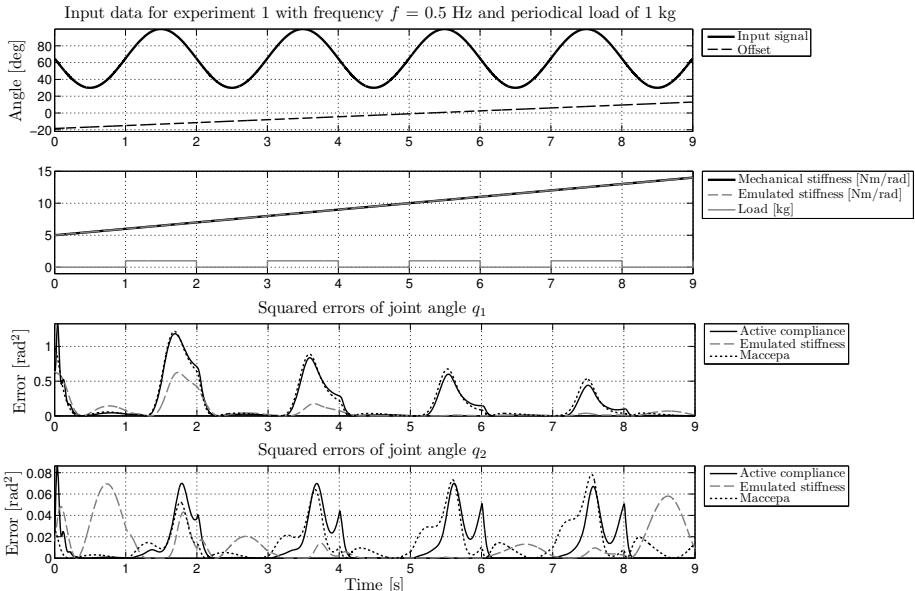


Fig. 6. Experiment 1, i.e. sine wave as reference input for both joints with given emulated and mechanical spring stiffness with a step frequency of 0.5Hz, is repeated additionally with ground contact which is achieved by a periodical load of 1kg

any specific time without consuming energy. A real mechanical spring stiffness cannot be changed sufficiently quickly. Thus the ideal system represents only an imaginary system.

The squared errors of the techniques “active compliance”, “emulated stiffness”, and “Maccepa” are separately illustrated for each joint in the third and fourth plot of Fig. 5. It can be recognized that the deviations of all techniques amount to the approximately same value after transition into steady state, less than 10° . In order to motivate legged locomotion with this kind of actuation, the same run was also performed with simulated ground contact, shown in Fig. 6. Compared to the first experiment, the squared errors increase for both joints during simulated ground contact with all techniques. Considering that positioning accuracy in dynamic locomotion does not or should not play an important role, however, we can adhere that the emulated spring stiffness technique copes well with the additional periodical loads.

3.2 Experiment 2 with Step Frequency $f = 3\text{ Hz}$

In this experiment we increased the step frequency by $f = 2.5\text{ Hz}$. We could observe that the squared errors of the active compliant and Maccepa actuated system increase while those of the emulated spring stiffness technique stay stable. Both joints of the arm approximate the desired trajectories with approximately the same precision as in Experiment 1.

3.3 Experiment 3 with Step Frequency $f = 5\text{ Hz}$

In our last experiment we changed the stiffness linearly from $4 \frac{\text{Nm}}{\text{rad}}$ to $34 \frac{\text{Nm}}{\text{rad}}$ during the simulation time of 30s. Furthermore the step frequency was increased to 5Hz. The runs shown here are performed without ground contact. The results obtained with the emulated spring stiffness technique indicate slight improvement despite higher frequency (see Fig. 7). Active compliance and Maccepa actuators, however, yield deviations that seem to grow synchronously with the frequency.

In order to probe the causes of the large deviations with active compliance and Maccepa actuators we examined the actual trajectories more in details. In Fig. 8 we compare the sensed responses, i.e. measured trajectories, q_1 and q_2 obtained by an ideal system with online adaptable spring stiffness with the responses obtained by Maccepa actuators and the active compliant system. As the uppermost plot in Fig. 8 indicates, the deviations of the actual trajectory q_1 with Maccepa actuation are large whereas the deviations of the actual trajectory q_2 about approximately 7° are still bearable for applications in legged locomotion. Taking a look at the actual trajectories obtained with the active compliant system, we note surprisingly that the deviations are in fact smaller than assumed solely by the squared errors. The measured trajectories approximate the ideal given trajectories both in the shape and amplitude quite well. Consequently, the large squared errors seen in Fig. 7 are not due to the small deviations of the amplitudes but rather due to the phase shifts.

In order to assess the performance of the emulated spring stiffness technique, we also compared both the desired reference inputs and measured responses obtained with the emulated spring stiffness technique with the inputs and responses obtained by the

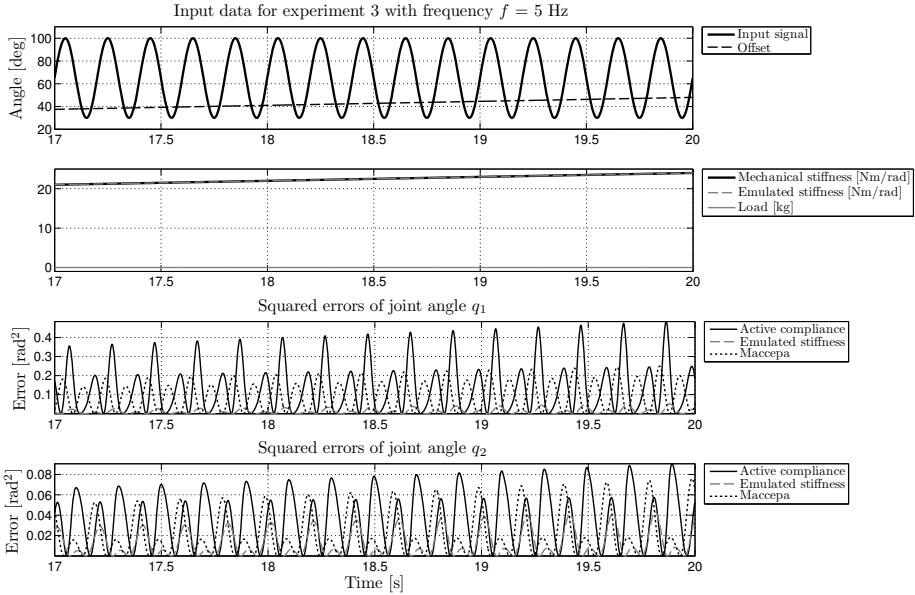


Fig. 7. Experiment 3: Similar setup as in Experiment 1, sine wave as reference input for both joints with given emulated and mechanical spring stiffness with an increased frequency of 5Hz without ground contact

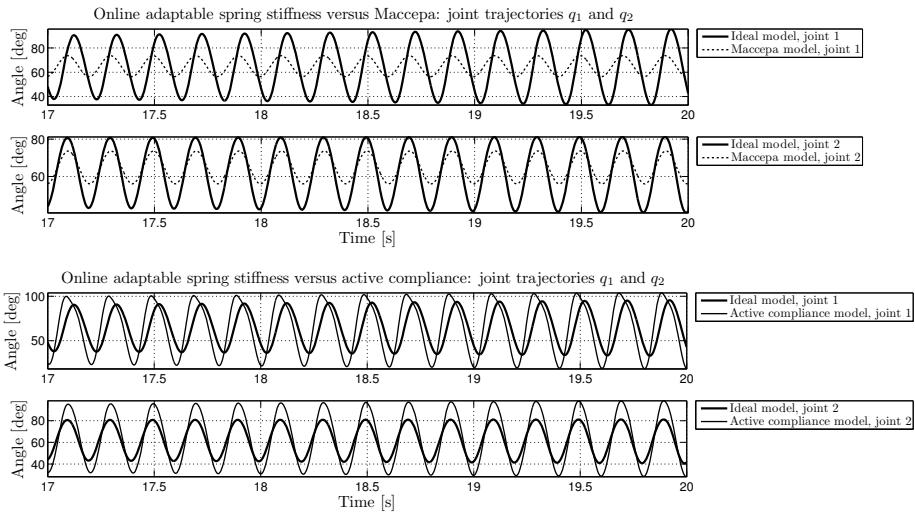


Fig. 8. Actual trajectories respectively measured responses obtained by the 2-DOF arm with online adaptable spring stiffness are compared to the responses obtained by Maccepa actuation for joints 1 and 2 in the uppermost plot. In the lower plot the responses of the ideal system with online adaptable spring stiffness are compared to the responses of the active compliant system.

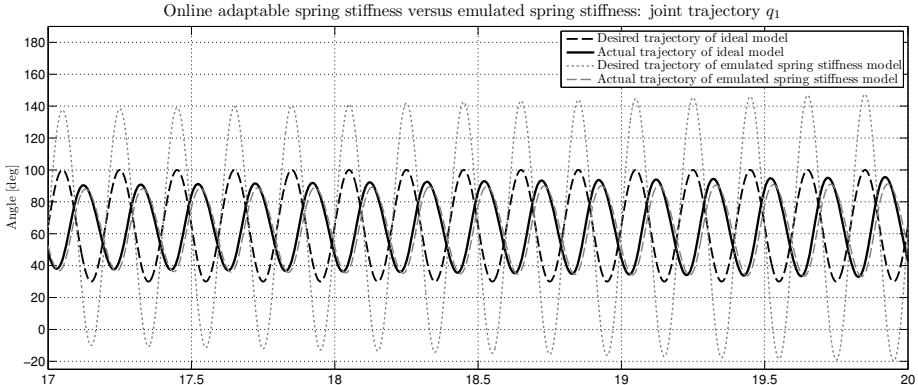


Fig. 9. Desired and actual trajectories obtained by the ideal system with online adaptable spring stiffness for joint q_1 versus the input and response trajectories obtained by the emulated spring stiffness technique.

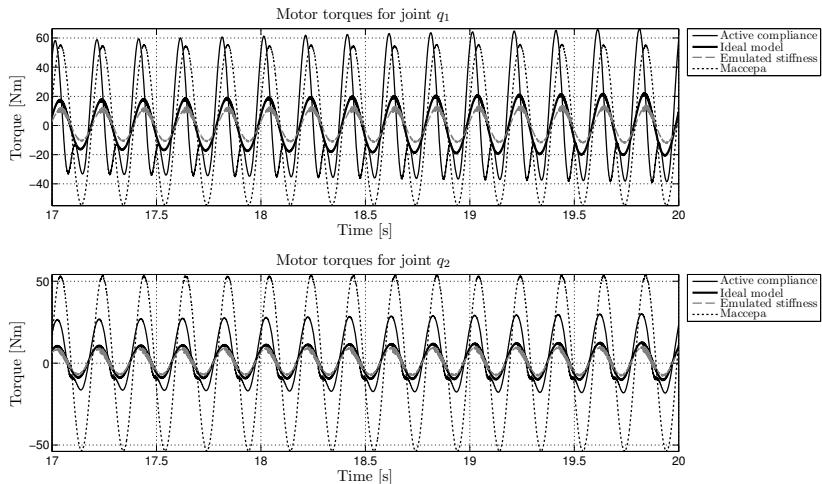


Fig. 10. Motor torques required for the desired trajectories q_1 and q_2 generated by the ideal system with online adaptable spring stiffness, active compliant system, Maccepa actuators and emulated stiffness technique.

ideal system with online adaptable spring stiffness. Fig. 9 displays the input and response trajectories q_1 . Neglecting the insignificantly small time delay between the actual trajectory of the ideal model and that of the emulated spring stiffness technique, we can note that the sole amplitudes of both actuation systems are almost identical. The model controlled by the emulated spring stiffness technique behaves almost exactly like a model with a mechanical spring stiffness that can change continuously online during operation.

When discussing different possibilities of actuation, not only the deviations play an enormous role but also the required motor torques and velocities. Therefore, we also examined the torques and velocities required for the desired trajectories of this

experiment. The velocities are not illustrated here, but they have also been examined. The velocities that are needed for the displayed motor torques can be generated by the same motor-gear combinations that generate the torques. As expected, the motor torques required for an active compliant system or Maccepa actuation are at least twice as high as for the ideal model with online-tunable spring stiffness (cf. Fig. 10). Interestingly the emulated spring stiffness technique requires even less torques than the ideal system with online adaptable spring stiffness.

4 Discussion

The purpose of the previous section was the evaluation of the performance and limits of the proposed technique of emulated spring stiffness in terms of deviations and motor specifications when comparing to two other techniques. Both techniques, the torque-controlled approach and the mechanically stiffness control with Maccepa actuators, are well known and often used methodologies. The large deviations observed in the figures are due to particular application dependent details, modeling inaccuracies and the used error measure.

Provided that the exact dynamic equations of the model are known and the inverse dynamics model is correctly determined, active compliance allows the accurate realization of a desired trajectory with any spring constant and damping value. Interestingly, this method requires neither elastic drives and elements nor any further additional hardware. Thus, it does not result in further weight and additional masses and inertias of the actuation module. The downside, however, is the computational complexity that is even more of a disadvantage when fast periodical motions of a more complex system than a 2-DOF arm are to be controlled. Furthermore, virtual compliance subsists on feedback, and therefore sufficiently fast sensors are essential. Due to the critical sensory flow, known in robotic applications, this technique may decrease in performance for fast motions. The noticed deviations of the active compliant system are owed to the error measure which is based on both the phase shifts and the deviations of the amplitudes. A different measure based only on the deviations of the amplitudes would yield smaller errors than depicted in the above figures.

As for Maccepa actuator, it remains to say that on-the-fly adjustment of the effective physical stiffness of a system by a small light motor is crucial and advantageous. It also has been used in the bipedal robot Veronica. However, the additional mass and inertia of the special joint construction play an important role in the energy increase [2]. Furthermore, due to the complex actuation mechanism the modeling of the actuator is quite challenging. For instance, the lever arm changes depending on the current configuration. These and other details that we were not able to consider in the simulation model without undertaking tremendous efforts lead to the noticed deviations.

The comparisons were designed to better assess the performance limits of the proposed technique. Results at this stage of investigations indicate that the emulated spring stiffness technique has the potential to be used in legged robots. Compared to other compliant actuators the mechanics of the proposed actuation can be considered as rather simple. It basically represents an extended SEA. Additionally, the implemented control strategy given by Eqs. (1) and (2) managed to approximate the given trajectories in the

experiments described in Section 3 very well, requiring beforehand a manually tuned correction offset to reduce occurring deviations.

As pure feedforward control strategy it does not need any sensory system. In general, no special hardware, most importantly no additional motor that would in turn result in additional weight, is required. The method does not require any prior knowledge about the model and can be applied to any joint. The technique turns out to represent a good combination of active and passive compliant actuation. Further investigations need to be carried out to ensure its advantages for different, also non-periodical motions. At the current development stage, if a position is held for a certain time, deviations to the desired position will occur. As Eqs. (1) and (2) do not approximate the real characteristic curve of the deployed spring, some parameters such as the offset o must be adapted to the specific operation. Therefore, this type of actuator and controller is even more advantageous if a prior analysis is performed to choose and determine the necessary force-elongation characteristic of the spring as well as the resulting compliance characteristic of the overall system. We are currently investigating a method to integrate a such prior analysis into the design of the proposed actuation module beforehand. Furthermore, the actuation principle can be enhanced by an intelligent coupling of feedforward and feedback control.

Finally, this kind of compliant actuation is essential in order to realize multiple modes of locomotion and locomotion on different terrains with varying ground stiffness in changing, unstructured environments. As this technique allows to change the joint stiffness online, it is possible to change the overall leg stiffness and therefore to change online the gait of a legged robot. An adjustment of leg stiffness leads to a different ground contact duration per step which in turn results in a different gait. This can be achieved solely by changing the emulated spring stiffness parameter μ (cf. Eqs. (1) and (2)). We have integrated this type of actuator and controller in a four-legged robot in simulation and experienced performance gains in the sense of multimodal locomotion: a real gait transition from walking into trotting was initiated by the emulated spring stiffness technique. For further details regarding the four-legged robot and online gait transitions we refer to [7].

5 Conclusion

To the end of developing biologically inspired legged robots that are capable of versatile, robust and efficient locomotion compliant actuation modules are essential. In this paper a basic but effective technique for the realization of adaptable compliance has been proposed and investigated. The underlying mechanical structure of the proposed actuation principle is a SEA containing two elastic elements. The implemented feedforward controller emulates the effect of a real change of the mechanical spring stiffness, requiring however a manually tuned correction offset for the small occurring deviations. The proposed technique was compared to an active compliant torque-controlled approach and a mechanically controlled stiffness technique using a 2-DOF arm. The results demonstrated that the proposed methodology can be applied to fast periodical motions with and without ground contact for legged locomotion. A system that is feed-forward controlled with the presented actuation module behaves almost like a virtual

ideal system with online adaptable mechanical spring stiffness at any time, however coming at the price of small deviations. The developed technique requires neither prior knowledge of the model nor any hardware and computing power. Adaptable spring stiffness is, of course, necessary if different gaits shall be demonstrated. A legged robot using the proposed actuation module, as shown in simulation so far, is capable of different gaits, by changing only the emulated spring stiffness parameter. Investigations on a real robot model are necessary to assess the performance limits of the presented technique for real applications in the future.

Acknowledgements. Parts of this research have been supported by the German Research Foundation (DFG) under grant no. STR 533/7-1 and within the Research Training Group 1362.

References

1. Ham, R.V., Sugar, T.G., Vanderborght, B., Hollander, K.W., Lefeber, D.: Compliant actuator designs. *IEEE Robotics and Automation Magazine* 16(3), 81–94 (2009)
2. Ham, R.V., Vanderborght, B., Damme, M.V., Verrelst, B., Lefeber, D.: Maccepa: the mechanically adjustable compliance and controllable equilibrium position actuator for controlled passive walking. In: *IEEE International Conference on Robotics and Automation*, pp. 2195–2200 (2006)
3. Klug, S., Lens, T., von Stryk, O., Möhl, B., Karguth, A.: Biologically inspired robot manipulator for new applications in automation engineering. In: Proc. of Robotik 2008, VDI-Berichte, VDI Wissensforum GmbH, vol. 2012 (June 2008), <http://www.biorob.de>
4. McN.Alexander, R.: Three uses for springs in legged locomotion. *The International Journal of Robotics Research* 9(2), 53–61 (1990)
5. Pratt, G.A., Williamson, M.M.: Series elastic actuators. In: Proc. of the IEEE International Workshop on Intelligent Robots and Systems, pp. 399–406 (1995)
6. Radkhah, K., Kurowski, S., von Stryk, O.: Design considerations for a biologically inspired compliant four-legged robot. In: Proc. of the IEEE International Conference on Robotics and Biomimetics, pp. 598–603 (December 19-23, 2009); finalist for Best Paper Award in Biomimetics
7. Radkhah, K., Kurowski, S., von Stryk, O.: Feedforward-controlled emulated spring stiffness for a biologically inspired four-legged robot. Tech. rep. TUD-CS-2010-0213, Department of Computer Science, Technische Universität Darmstadt, Darmstadt, Germany (2010), www.sim.tu-darmstadt.de/~radkhah
8. Schäffer, A.A., Eiberger, O., Grebenstein, M., Haddadin, S., Ott, C., Wimböck, T., Wolf, S., Hirzinger, G.: Soft robotics, from torque feedback-controlled lightweight robots to intrinsically compliant systems. *IEEE Robotics and Automation Magazine* 15(3), 20–30 (2008)
9. Veneman, J.F., Ekkelenkamp, R., Kruidhof, R., van der Helm, F.C.T., van der Kooij, H.: A series elastic- and bowden-cable-based actuation system for use as torque actuator in exoskeleton-type robots. *The International Journal of Robotics Research* 25(3), 261–281 (2006)

Different Approaches in Feeding of a Flexible Manufacturing Cell

Claudia Raluca Tudorie

University Politehnica of Bucharest, Spl. Independenței 313, Romania

Tel.: +40 214029314

raluca.tudorie@cimr.pub.ro

<http://www.cimr.pub.ro>

Abstract. The paper aims to present a comparison between two possibilities of feeding a manufacturing cell. There are presented the theoretical aspects in using a 3D robot vision system that will be implemented in the laboratory and some experiments that were made using a part feeding workstation. A general overview of the robot vision implementation in the field of assembly is presented, considering robot vision with its advantages and disadvantages and the structure of a typical robot vision system with its basic components. Parts grasping and manipulation, identification, location and spatial orientation problems were treated during the experiments.

Keywords: robot vision, flexible feeding, robotic manipulators, bin picking, stereo matching.

1 Introduction

In flexible manufacturing cells, there is a strong need for advanced object detection and recognition, for object grasping and for the capability to perform operations with objects that are randomly positioned in robots workspaces. Today, vision based robotic assembly systems can be effectively applied to advanced robot-based assembly tasks.

The first industrial robots proved to be useful in repetitive tasks, as they were considered blind slaves. In such tasks the form and posture of the work-pieces are predictable. The robots were limited in total inability to cope with new and unexpected situations. While the positional accuracy of the robot may be very high, it may be expected to operate in an environment where the parts delivered to it are in an unknown position or posture, or may have a variable size or form. In such situations, there have to be implemented vision systems. [1]

In recent time, there are often used modern robot vision configurations with advanced recognition systems. Their role is to adjust the coordinates from where the robot expected to find the object to where it actually is located.

The advances in 3D vision have made robots adept at recognizing a changing environment and adapting to it. This flexibility has allowed robots to work on projects that lack precise consistency, something that was very difficult for a robot to do in the

past. Currently, robotic vision research is expanding into many new areas. Robots can now pick variously shaped objects from a conveyor or eliminate the need for part designated in-feed systems and machine. [3]

Implementation of robot vision in assembly processes is a multilayer problem and demands knowledge, experiences, innovations and most often a specific solution of the problem. The development and planning procedure of an assembly, inspection and measurement process is split into tasks and goals such as: detection, recognition, grasping, handling, measurement, fault detection and into machine vision component selection and working conditions determination such as: camera, computer, lenses and optics, illumination, position determination, etc. [6]

A typical assembly cell will comprise of computer-controlled devices (such as robots, grippers, etc.) components and fixtures that can functionally accomplish or support the accomplishment of one or all of the following tasks:

- grasping of a target part;
- manipulation and placement/assembly of parts;
- part and object recognition before, during and after assembly;
- planning and control of the actuators and grippers to accomplish the physical assembly. [2]

In recent time, industrial robots can perform handling and assembly tasks with very high speed and precision. But, compared to human operators, robots are hampered by their lack of sensory perception and their need for advanced sensorial capabilities in order to achieve more sophisticated tasks in a non-structured environment.

2 Robot Vision

Robot vision is the application of computer vision to industry and manufacturing, especially in robotics. Robot vision requires digital input/output devices and computer networks to control manufacturing equipment such as robotic arms. Specific advantages of robot vision systems include precision, flexibility, consistency and cost effectiveness.

A machine vision system for assembly tasks (see fig.1) consists of several of the following components, depending on different factors (such as: environment, application and budget):

- one or more digital or analogue cameras or scanner (black-and-white or colour) with suitable optics for acquiring images;
- a video-computer interface module (frame grabber) – depends on the application;
- a processor (often a PC or embedded processor, such as a DSP) – when processors and frame grabbers are integrated into the camera itself, such cameras are called “smart cameras”; for high-speed applications, dedicated electronic image processing hardware may replace the computer and software;
- input/output hardware or communication links;
- optics - lenses to focus the desired field of view onto the image sensor;
- light source (LED, fluorescent or halogen lamps, etc.);
- software, implementing algorithmic and heuristic procedures devised specially for each application;

- a synchronizing sensor for part detection to trigger image acquisition and processing;
- actuator (for AVI, this is likely to be a simple *accept/reject* mechanism);
- integrating these so that they operate together in an harmonious manner.

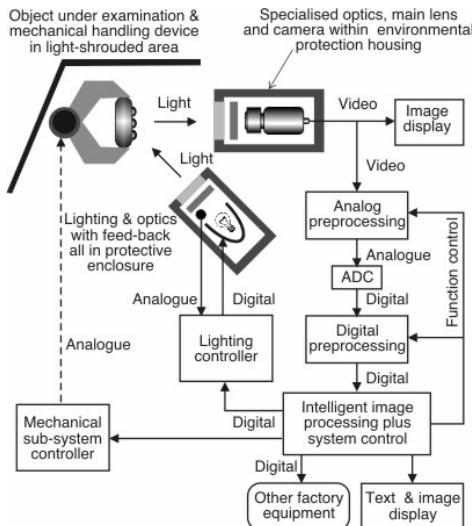


Fig. 1. Basic components of a computer vision system [1]

The most commonly used vision sensors in robot-based assembly are black-and-white or colour CCD (charge-coupled device) cameras. 2D-vision systems consist of standard industrial CCD cameras used to take images that are processed by the robot to make decisions on how should be handled the objects. Such systems are appropriate for objects that are lying on a belt or in a bin.

For parts that can stack upon each other or that may shift from side to side as the parts stack up or when parts are oriented randomly in a bin, depth estimation for a vision system is necessary. For depth estimation most commonly stereo vision systems are used. For such applications 3-D vision systems have to be applied to get a range image and the orientation of a part in 3D-space. Different laser sensors in conjunction with 2-D cameras, sensors with structured light and stereo cameras, together with different algorithms, can provide 3-D information.

Nowadays, many industrial assembly applications are successfully being handled using computer vision and robots, especially in robot-based assembly cells. However, for these applications objects have a simple (2D) shape and/or are organized in a structured manner. On the other hand, a general so-called bin-picking, where the objects have a 3-D shape and are randomly organized in a box, still remains a problem. Despite of many research works, that offer special solutions and improvements in the overcoming the bin-picking problem, the oldest challenge in robotics remains still unsolved. [6]

This paper presents the theoretical aspects in using a 3D robot vision system (method 1: pin picking) and experimental results that were obtained in our laboratory using a part feeding workstation (method 2: vibratory feeders).

3 Method 1: Bin Picking

The automatic grasping of parts in a bin with the use of a robotic arm is commonly known as robotic bin picking. One of the important tasks found in a modern flexible manufacturing cell is the need to present work pieces to automated machinery from a supply bin. The bin picking problem was solved using mechanical vibratory feeders, where vision feedback was unavailable. This solution has some problems with parts occlusion and overlapping objects. Due to these advantages, modern bin picking systems perform grasping and manipulation operations using vision feedback.

In order to carry out grasping and manipulation operations safely and efficiently, there is a need to know the identity, location and spatial orientation of the objects that lie in a bin. [5]

However, loading and unloading of work pieces to and from machine tools were still done by human operators even in the 1990s. For the first time, the intelligent robots appeared in 2001. The term intelligent doesn't mean a humanoid robot that can talk and walk like a human being, but rather one that performs highly complicated tasks like a skilled worker on the production site by using vision and force sensors. It has first enabled the automatic precision loading of parts to the fixture of the machining center and eliminated the need for dedicated parts supply equipment, as the robot picks up parts one by one using its vision sensor once the parts are delivered to a basket near the robot. This also eliminated a burdensome process imposed upon human operators: that of arraying workpieces for the dedicated equipment. [8]

Two types of vision sensors are often used in factory floor: two-dimensional (2D) and three-dimensional (3D) vision sensors. The 2D vision sensors acquire two-dimensional images of an object by irradiating natural light or artificial light on the object and taking the image of the reflected light with a CCD or other type of camera. This enables obtaining a two-dimensional position and rotation angle of the object. Recently, 2D vision sensors have been made useful under the severe production environment at the factory floor due to enhanced tolerance to change in brightness and image degradation based on improved processing algorithms and increased speed.

There are two major methods for 3D vision sensors, the *structured light method* and the *stereo method*. The structured light method irradiates the structured light such as slit light or pattern light on an object, and takes images of the reflected light by a CCD or other type of camera to obtain images of the object. The 3D position and posture of the object are calculated with accuracy from these images.

Figure 2 shows the bin-picking function realized using the 3D vision sensor with the structured light method. The robot can pick up workpieces one by one from those randomly piled in a basket using the 3D vision sensor [8].

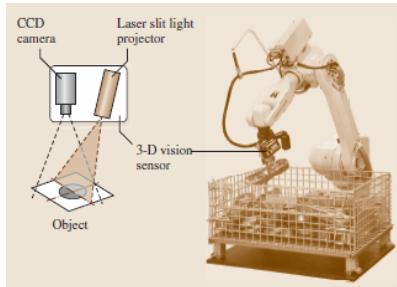


Fig. 2. Bin-picking function using 3D vision sensor with structured light method [8]

The other 3D vision method, the stereovision method, uses images taken by two cameras, by matching the correspondence points on the left and right images to calculate the objects' 3D position and posture. Although it takes some time to match corresponding points, this method may be advantageous in such a case where a mobile robot has to recognize its surroundings, as there is no need to irradiate auxiliary light as the structured light method.

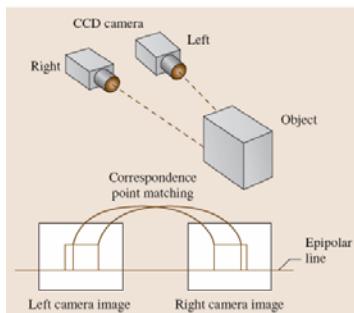


Fig. 3. Stereo vision method [8]

A bin picking application has to address three difficult problems: scene interpretation, object recognition and pose estimation. Initial approaches to these tasks were based on modeling parts using 2D surface representations. Typical 2D representations include invariant shape descriptors, algebraic curves and appearance based models. These systems are not able to deal with severe viewpoint distortions or objects with complex shapes/textures and the spatial orientation cannot be robustly estimated for objects with free-form contours.

To solve these problems, most bin picking systems attempt to recognize the scene and to estimate the spatial orientation of the objects using 3D information. The most difficult problem for 3D bin picking systems is the procedure required to perform the scene to model feature matching [4], [5].

In many cases in area of assembly processes, a robot must perceive its 3D environment to be effective. Yet recovering 3D information and describing it still

remains the subject of fundamental research. Appearance of a 3D scene depends mostly on illumination, position and the direction of the vision sensor.

4 Method 2: Vibratory Feeders

This section is dedicated to the presentation of a part feeding workstation that exists in our laboratory. The system comprises a horizontally articulated SCARA robot with a pneumatically actuated gripper, two part feeders - each of it having a camera above the illuminated surfaces and a remote PC. The Adept SmartController CX runs Adept Sight which executes commands from the remote PC via a TCP/IP Ethernet connection. The images are stored in 8-bit grayscale format and are transferred to the PC host through IEEE-1394 fire wire ports.

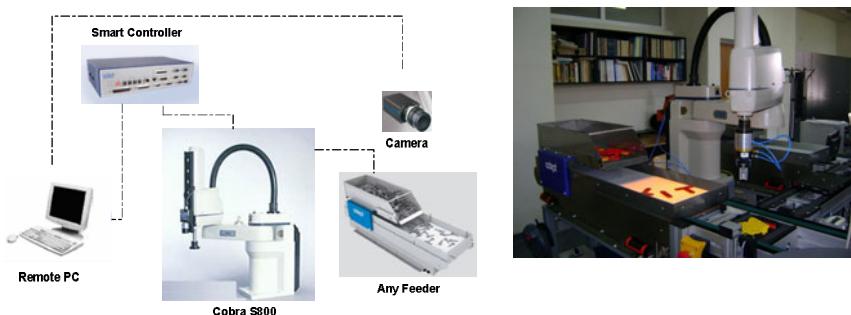


Fig. 4.Vision guided robot system components

The feeder is composed of two vibrating surfaces: bulk container and feed surface. Its role is to choose the work pieces from the main parts storage (bulk container) and spread them out on the illuminated area of the feed surface in order to be inspected by the camera. The camera can be a fixed down-looking or a mobile arm-mounted camera, in our case a fixed down-looking camera was used. After the inspection of the illuminated area, there are three possibilities:

- the robot can pick up some workpieces;
- a decision is taken in order to bring more parts from the bulk container;
- a decision is taken to change the way the parts are placed on the feed surface.

To realize this, the feeder can execute several vibrating movements:

- Dispense - vibrates bulk container and feed surface together in order to send parts falling on to the feed surface;
- Feed Forward - moves parts forward on the feed surface;
- Feed Backward - moves parts backward on the feed surface;
- Flip - flips parts on the other side;
- Flip Forward / Flip Backwards – associates the flip motion with the move forward / backwards motion;

- Purge - purges parts from feed surface and bulk container (parts from feed surface fall outside the back of the feeder, parts from bulk container fall only if the retainer gate is manually opened).

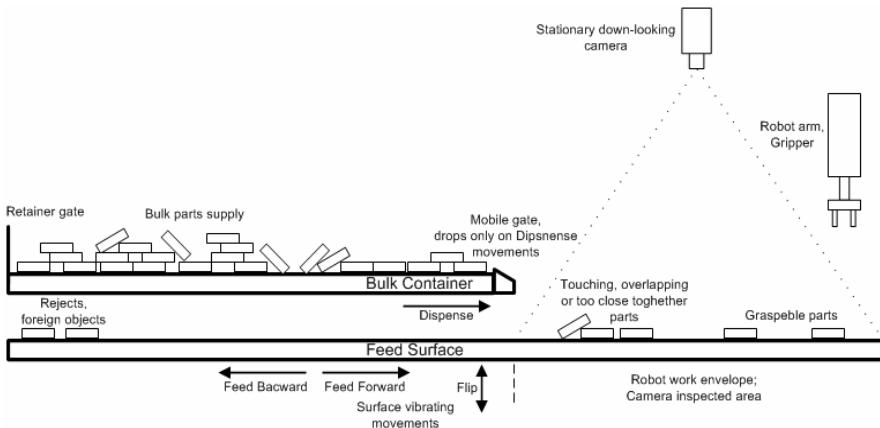


Fig. 5. Vibrating parts feeder's surfaces and working principle

In order to send commands to the feeder and receive its status, two communication methods are available: the RS232 serial communication and digital I/O. Since digital I/O lines of industrial standard are rather expensive and the robot controller has 4 available serial interfaces, the serial communication method was chosen and implemented.

The serial communication with the feeder is based on sending/receiving strings. The strings that represent the feeder's commands are presented in Table 1.

Table 1. AnyFeeder serial commands

Command	Explanation
<code>>x=code<cr></code>	Executes a movement (<i>code</i> is the movement command's code) Ex: for <i>code</i> =1, executes a feed forward.
<code>>ab[code]=[turns]<cr></code>	Sets number of repetitions for a movement command (<i>code</i> is the command code, <i>turns</i> is the number of the repetitions).
<code>>ab[code]=[turns]_x=code<cr></code>	Sets number of repetitions and execute motion.
<code>>ab[code]=[speed]_x=[code]<cr></code>	Sets speed.

In Table 1, the symbol `>` represents the prompt of a text terminal window, `<cr>` represents carriage return, ASCII code 13 (decimal), `[speed]` integer in the range 1..10, `[turns]` integer in the range 1..10 and `_` explicit space (not an underscore).

In response, the feeder will send the following messages:

- m11 – motor 1 received command and it is busy;
- m10 – motor 1 completed action successfully;
- m21 – motor 2 received command and is busy;
- m20 – motor 2 completed action successfully;
- m1x or m2x – motor 1 or 2 is reporting x error code.

The steps below describe a simple program flow using the AnyFeeder.

1. Initialize the AnyFeeder.
2. Send “dispense” command to AnyFeeder to feed parts.
3. Acquire vision image and locate “usable” parts; store part count.
4. Command robot to pick-place the usable parts; decrement part count for each pick.
5. When part count reaches 0, send combinations of “feed forward”, “feed backward”, “flip”, “feed/flip forward” and “feed/flip backward” commands, along with acquiring vision images, to locate more usable parts.
 - a. If usable parts are found, store part count and go back to step 4.
 - b. If no usable parts are found, and feed area is not empty, repeat step 5.
 - c. If no usable parts are found, and feed area is empty, go to step 2.

4.1 Vision Controlled Part Recognition and Grasping

There are two major problems that have to be treated while working with visual guidance software: part occlusion and overlapping objects.

The first problem (part occlusion) is generated by the fact that the robot uses a gripper with parallel fingers; in order to approach the part, the gripper needs to have two free fingerprints next to the object in the grasping position.

If a part recognized by the vision's software locator tool (see fig. 6) is presented on the feed surface in a way that an obstacle occupies one of the gripper's fingerprints, then the gripper will come into collision with this obstacle. To avoid this situation, two histogram tools are attached to the model of the object, each having the gripper's fingerprint dimensions. The histogram tools are placed in the grasping positions.

For each of these tools the "minimum grey level value" is inspected. If the value is 255, then all the pixels inside the tool are white and the fingerprint is declared free of obstacles; if the value is smaller than 255, it is considered that an obstacle is present and the fingerprint is not free of obstacles (see fig. 8). The object is considered to be graspable when the two fingerprints are declared free. Now, the robot is commanded to pick up the part.

The second problem (overlapping objects) refers to the fact that the locator tool doesn't recognize overlapping objects (see Fig.8). This is a good thing, because this robot's gripper cannot grasp overlapping objects and it is difficult to detect which object is on top.

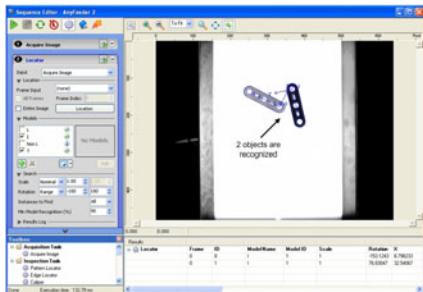


Fig. 6. Locator recognizes two objects

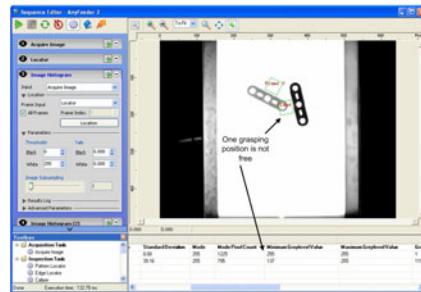


Fig. 7. Grasping position of an object is blocked

The system needs to know if there are unrecognized objects on the feed surface, in order to present them in a different manner to the camera. Otherwise, if the system only picks recognized objects and commands the feeder to bring new objects on the feed surface, the feed surface will soon become blocked with objects that, even if they are recognized, are not graspable.

Blobs are areas of adjacent pixels whose gray level satisfy a condition. Since the grey level of free pixels is 255, it means that the pixels occupied by parts have a grey level smaller than 255, so the Blob Analyzer is set up to detect blob's that have a smaller than 255 grey level (see Fig.8). Comparing the number of recognized objects with the number of detected blobs, we can determine if the feed surface contains unrecognized objects (which is true in this case).

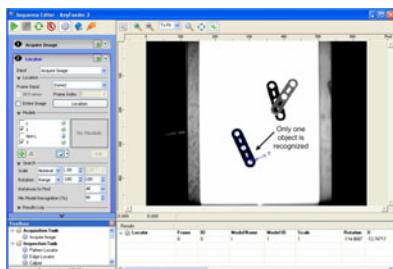


Fig. 8. Locator recognizes one object

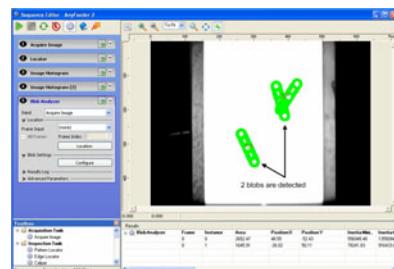


Fig. 9. Blob analyzer detects two blobs

4.2 Parts Feeding Strategies

In order to successfully pick parts from the feed surface, a strategy has to be implemented. This strategy has to check the following problems:

- robot picks only recognized graspable parts;
- program exits when desired number of parts is picked;
- if feed surface contains no parts, then parts are brought from the bulk container (dispense);
- if feed surface contains unrecognized parts, then parts are flipped;

- if feed surface contains blocked parts and parts were picked, then it is inspected again to detect possibly free parts;
- if flip is repeated too many times without detected parts becoming recognized, present parts are considered rejects and purged;
- if dispense is repeated too many times without resulting in parts on the feed surface, the "No parts in Bulk Container" error is issued, program exits;
- if parts are still on the feed surface after a purge command, the "Rejects blocked on Feed Surface" error is issued, program exits;
- if too many purge commands are issued, then "Too many rejects" error is issued and program exits.

For solving all these tasks, the following algorithm was developed:

```
Init: initialize variables;

Step 1: Inspect Feed Surface, if it contains parts
then go to Step 3, otherwise go to Step 2;

Step 2: Dispense parts, if maximum number of
successive Dispense commands is reached then go to
Errors, else go to Step 1;

Step 3: Pick recognized graspable parts, if desired
number of picked parts is reached then go to End, if
Feed Surface contains blocked parts and parts were
picked then go to Step 1, if Feed Surface contains
only unrecognized/ungraspable parts then go to Step
4;

Step 4: Flip parts, if maximum number of successive
Flip repetitions is reached then go to Step 5, else
go to Step 1;

Step 5: Purge rejects, if Feed Surface still contains
rejects then go to Errors, else go to Step 2;

Errors: report encountered error, go to End;

End: end program.
```

After developing and implementing the program, tests were made. The robot was commanded to pick up 20 parts, first using only one feeder, after that using the both feeders that exist in the laboratory. Both situations were repeated 10 times.

Using the second feeder, we have succeeded in increasing the overall robot working time by about 20%. Also, the total working time has dropped by about 30%. Another important aspect is that the working time with two feeders spans a narrower range of values (20 versus 35 seconds), thus making the workstation's performances more consistent, easier to predict and easier to integrate in the production schedule.

On the price versus performance aspect, adding a second feeder to an existing workstation increases its cost with only 15%, while the performance increases with 30%. So, if only one feeder is present and performances are not satisfactory, the right thing to do is adding a second feeder and not adding a second workstation.

5 Conclusions

The industrial intelligent robots still have tasks in which they can't complete with skilled workers, though they have a high level of skills. The application of robot vision in robot assembly and control processes provides immense potential and challenges, at the same time, for both research and industrial applications.

A general overview and the state-of-the art of the robot vision implementation in the field of assembly is given, considering robot vision with its advantages and disadvantages and the structure of a typical robot vision system with its basic components description. Some methods, most commonly used in the robot vision, are described. Also the bin-picking problem with randomly organized 3-D parts in a bin is presented by its theoretical aspects. The bin picking problem was solved using mechanical vibratory feeders, (called any-feeders) where vision feedback was unavailable. This solution has some problems with parts occlusion and overlapping objects. Due to these advantages, modern bin picking systems perform grasping and manipulation operations using vision feedback.

In order to carry out grasping and manipulation operations safety and efficiently, there is a need to know the identity, location and spatial orientation of the objects that lie in a bin.

Acknowledgements

The work has been funded by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of Labour, Family and Social Protection through the Financial Agreement POSDRU/6/1.5/S/16. The author is grateful to Mihai Parlea for his contribution in software development and experimental evaluation.

References

1. Batchelor, B.G.: Natural and Artificial Vision. Cardiff University (2006)
2. Cecil, J., Powell, D., Vasquez, D.: Assembly and Manipulation of Micro Devices – A State of the Art Survey. *Robotics and Computer-Integrated Manufacturing* 23 (2007)
3. Christe, B.: Robotic Application Research: Past, Present & Future (July 2009), <http://www.robotics.org>
4. Ghita, O., Whelan, P.F., Mallon, J.: Computational Approach for Depth from Defocus. *Journal of Electronic Imaging* 14 (2005)
5. Ghita, O., Whelan, P.F.: A Systems Engineering Approach to Robotic Bin Picking. In: *Stereo Vision*, pp. 59–72. In-teh (2008)
6. Herakovic, N.: Robot Vision in Industrial Assembly and Quality Control Processes. In: *Robot Vision*, pp. 501–534. In-teh (2010)
7. Hussmann, S., Ringbeck, T., Hagebeuker, B.: A Performance Review of 3D TOF Vision Systems in Comparison to Stereo Vision Systems. In: *Stereo Vision*, pp. 103–120. In-Teh (2008)

8. Inaba, Y., Sakakibara, S.: Industrial Intelligent Robots. Springer Handbook of Automation, pp. 349–363 (2009)
9. Scharstein, D., Szeliski, R.: A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. International Journal of Computer Vision 47, 7–42 (2002)
10. Schraft, L.G., Ledermann, T.: Intelligent picking of chaotically stored objects. Assembly Automation 23(1), 38–42 (2003)

On the Stability of Bipedal Walking

Pieter van Zutven, Dragan Kostić, and Henk Nijmeijer

Eindhoven University of Technology, Faculty of Mechanical Engineering,
Dynamics and Control group, 5600 MB Eindhoven, The Netherlands
`p.w.m.v.zutven@tue.nl`

Abstract. Stability of bipedal locomotion is analyzed using a model of a planar biped written in the framework of systems with unilateral constraints. Based on this model, two different stable walking gaits are derived: one which fulfills the widely used criterion of the Zero Moment Point (ZMP) and another one violating this criterion. Both gaits are determined using systematic model-based designs. The model and the two gaits are used in simulations to illustrate conservatisms of two commonly used methods for stability analysis of bipedal walking: the ZMP criterion and Poincaré return map method. We show that none of these two methods can give us a general qualification of bipedal walking stability.

Keywords: humanoid robotics, bipedal locomotion, stability analysis, zero moment point, Poincaré return map.

1 Introduction

Stability analysis of bipedal walking is difficult, since dynamics of the bipedal robots are highly non-linear, under actuated, subject to impacts, variable external forces, and discrete changes between different modes. The common strategies, such as analysis of the eigenvalues, gain and phase margins or Lyapunov stability theory, can be applied to particular modes, such as a single or double stance, but are usually incapable to characterize stability of all modes in total. So far, stability of bipedal walking is analyzed by specific techniques, such as Zero Moment Point (ZMP) [12], Poincaré return maps [2], [3], [13], Foot Rotation Indicator [1], the theory of capture points [7], and the foot placement estimator [4].

In this paper we analyze stability of bipedal locomotion using a model of a planar biped. The Lagrange-Euler equations of motion of the bipedal robot are represented in the framework of systems with unilateral constraints [4], by implementing set-valued force laws for normal contact and tangential Coulomb friction. Taking these effects into account is crucial for accurate dynamical modeling of the biped. The derived model is used in systematic design of two different stable walking gaits: one which fulfills the widely used ZMP-criterion and another one violating this criterion. The same model is later on used in simulations to illustrate conservatisms of two commonly used methods for stability analysis of bipedal walking: the ZMP criterion and Poincaré return map method.

There are two major contributions of this paper. First, we develop a model of a bipedal robot using the framework of systems with unilateral constraints.

The second contribution is a systematic model-based design of a limit-cycle walking (LCW) gait [3]. Finally, we illustrate, by means of simulations, serious conservatism of two of the most common stability criteria for bipedal walking, namely the methods of ZMP and the Poincaré mapping. Numerical illustrations of their conservatism are seldom in the literature. The numerical results given in this paper confirm that neither criterion can be used to establish stability of an arbitrary walking gait, which motivates further research towards more general qualification of stability of bipedal walking.

The paper is organized as follows. In Sect. 2 a model of a planar biped is described. In Sect. 3 a ZMP based and a LCW gait are derived. Stability of these gaits is analyzed in Sect. 4. Conclusions are given in Sect. 5.

2 Model of a Planar Biped

A model of a planar biped is introduced in this section. This model is used for design of walking gaits and stability analysis in Sects. 3 and 4, respectively.

2.1 Unconstrained Dynamics

We model dynamics of a planar bipedal robot with hips, knees and feet. A kinematics representation of this robot is shown in Fig. 1 with the coordinate frames assigned according to the Denavit-Hartenberg convention [11]. The robot consists of two upper legs, two lower legs and two feet. Consequently, six actuated degrees of freedom (dofs) are present. To facilitate derivation of the Lagrange-Euler equations of motion, a virtual robotic arm is attached to the hip. This arm adds two extra dofs that describe Cartesian motions of the robot relative to the world frame $[x_0, z_0]^T$. The equations of the unconstrained robot dynamics, i.e. the robot not being in contact with the ground, are derived using the Lagrange-Euler method [11] and presented here in standard form [4], [5]:

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{u}} - \mathbf{h}(\mathbf{q}, \mathbf{u}) = \boldsymbol{\tau}, \quad (1)$$

where $\mathbf{q} \in \mathbb{R}^n$ is a minimal set of n generalized coordinates, $\mathbf{u} = \dot{\mathbf{q}}$, $\mathbf{M} \in \mathbb{R}^{n \times n}$ is the symmetric and positive definite inertia matrix, $\mathbf{h} \in \mathbb{R}^n$ is the vector containing all differentiable forces, such as gravitational forces and gyroscopic terms and $\boldsymbol{\tau} \in \mathbb{R}^n$ are the controller torques.

2.2 Contact Dynamics

Both feet of the biped can make contact with the ground with two contact points, located at the toe and heel. The contact points need to satisfy Signorini's set-valued force law, which states that the contact points cannot penetrate the ground. This results in a complementarity condition between the contact distance g_{Ni} and normal contact force λ_{Ni} at contact point $i \in \mathcal{I}_N$, where \mathcal{I}_N is the set of n_N active contact points:

$$g_{Ni} \geq 0, \quad \lambda_{Ni} \geq 0, \quad g_{Ni}\lambda_{Ni} = 0. \quad (2)$$

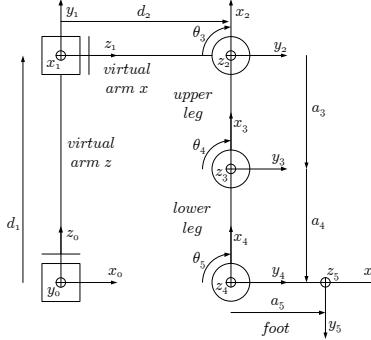


Fig. 1. Kinematic scheme of a planar bipedal robot with 6 actuated dofs

When the robot makes contact with the ground, an impact occurs. Newton's impact law is used to model these impacts. This law relates the pre- and post-impact velocity of contact point $i \in \mathcal{I}_N$ with a restitution coefficient e_{Ni} :

$$\gamma_{Ni}^+ = e_{Ni} \gamma_{Ni}^- , \quad (3)$$

where $\gamma_{Ni} = \frac{d}{dt} g_{Ni}$. When a robot foot is in contact with the ground, it is subject to the ground friction. This is modeled using the Coulomb's friction law: at contact point $i \in \mathcal{I}_N$, the friction coefficient μ_i relates the friction force λ_{Ti} with the normal contact force λ_{Ni} and the tangential contact velocity γ_{Ti} :

$$\lambda_{Ti} \in -\mu_i \text{Sign}(\gamma_{Ti}) \lambda_{Ni}, \quad \text{Sign}(x) \in \begin{cases} \{-1\} & x < 0 \\ [-1, 1] & x = 0 \\ \{1\} & x > 0 \end{cases} . \quad (4)$$

By combining the unconstrained equations of motion (1), Signorini's set valued contact law (2), Newton's impact law (3), and Coulomb's set valued friction law (4), we retrieve a new model of the dynamics of the planar bipedal robot with ground contact and tangential friction:

$$\begin{aligned} \text{Constrained dynamics: } & \boldsymbol{M}\dot{\boldsymbol{u}} - \boldsymbol{h} = \boldsymbol{\tau} + \boldsymbol{W}_N \boldsymbol{\lambda}_N + \boldsymbol{W}_T \boldsymbol{\lambda}_T , \\ \text{Signorini's set valued contact law: } & \boldsymbol{\gamma}_N \geq \mathbf{0}, \quad \boldsymbol{\lambda}_N \geq \mathbf{0}, \quad \boldsymbol{\gamma}_N^T \boldsymbol{\lambda}_N = 0 , \\ \text{Coulomb's set valued friction law: } & \boldsymbol{\lambda}_T \in -\boldsymbol{\mu} \text{diag}\{\text{Sign}(\boldsymbol{\gamma}_T)\} \boldsymbol{\lambda}_N , \\ \text{Impact law: } & \boldsymbol{\gamma}_N^+ = e_N \boldsymbol{\gamma}_N^-, \quad \boldsymbol{\gamma}_T^+ = e_T \boldsymbol{\gamma}_T^- , \\ \text{Generalized force directions: } & \boldsymbol{W}_N = \left(\frac{\partial \boldsymbol{\gamma}_N}{\partial \boldsymbol{u}} \right)^T, \quad \boldsymbol{W}_T = \left(\frac{\partial \boldsymbol{\gamma}_T}{\partial \boldsymbol{u}} \right)^T , \end{aligned} \quad (5)$$

where $\boldsymbol{\lambda}_N = \text{col}\{\lambda_{Ni}\} \in \mathbb{R}^{n_N}$, $\boldsymbol{\lambda}_T = \text{col}\{\lambda_{Ti}\} \in \mathbb{R}^{n_N}$, $\boldsymbol{\gamma}_N = \text{col}\{\gamma_{Ni}\} \in \mathbb{R}^{n_N}$, $\boldsymbol{\gamma}_T = \text{col}\{\gamma_{Ti}\} \in \mathbb{R}^{n_N}$, $\boldsymbol{\mu} = \text{diag}\{\mu_i\} \in \mathbb{R}^{n_N \times n_N}$, $\boldsymbol{e}_N = \text{diag}\{e_{Ni}\} \in \mathbb{R}^{n_N \times n_N}$ and $\boldsymbol{e}_T = \text{diag}\{e_{Ti}\} \in \mathbb{R}^{n_N \times n_N}$. Here, the contact equation is derived on the velocity level, in order to facilitate numerical integration. Notice that the contact equations are only valid for active contacts, i.e. $i \in \mathcal{I}_N$.

2.3 Numerical Integration

For numerical integration of the dynamical model (5), the time-stepping method of Moreau [6], [4], [5] can be used. This method use a time-discretization of generalized positions \mathbf{q} and velocities \mathbf{u} . Forces acting on a system are taken into account in an integral way over every time step. This means that the time-stepping method does not make a distinction between impulsive and finite forces. The equations of motion are rewritten into measure differential equations [4], [5], which are not on the level of forces, but on the level of momenta. In this way only increments in position and velocity are computed. The acceleration is not used, as it might become infinite due to the impacts. The time-stepping method of Moreau is basically a midpoint differential algebraic equation-integrator. If events occur due to impact or release during a time step, it is not necessary to switch to another mode in the system model, because all forces are taken into account through their momenta in an integral. For the sake of numerical integration, the full set of equations of motion (5) needs to be written in equations rather than complementarity conditions. Using convex analysis and taking into account the notion of proximal point [4], [5], we can write the contact conditions as (implicit) equations. Furthermore, the equations of motion should be written as measure differential equations as stated before:

$$\begin{aligned} M\mathbf{d}\mathbf{u} - \mathbf{h}dt &= \mathbf{W}_N d\boldsymbol{\Lambda}_N + \mathbf{W}_T d\boldsymbol{\Lambda}_T , \\ d\boldsymbol{\Lambda}_N &= \text{prox}_{C_N}(d\boldsymbol{\Lambda}_N - r\boldsymbol{\xi}_N) , \\ d\boldsymbol{\Lambda}_T &= \text{prox}_{C_T}(d\boldsymbol{\Lambda}_T - r\boldsymbol{\xi}_T) , \end{aligned} \quad (6)$$

with

$$\begin{aligned} C_N &= \mathbb{R}^+, \quad C_T = \{d\boldsymbol{\Lambda}_T \mid -\boldsymbol{\mu}_T d\boldsymbol{\Lambda}_N \leq d\boldsymbol{\Lambda}_T \leq \boldsymbol{\mu}_T d\boldsymbol{\Lambda}_N\} , \\ \boldsymbol{\xi}_N &= \boldsymbol{\gamma}_N^+ + \mathbf{e}_N \boldsymbol{\gamma}_N^-, \quad \boldsymbol{\xi}_T = \boldsymbol{\gamma}_T^+ + \mathbf{e}_T \boldsymbol{\gamma}_T^- , \\ \text{prox}_C(\mathbf{z}) &= \underset{\mathbf{x} \in C}{\text{argmin}} \|\mathbf{z} - \mathbf{x}\| . \end{aligned}$$

Here, $r > 0$ is a tuning variable for the numerical iteration, while $d\mathbf{u}$, dt , $d\boldsymbol{\Lambda}_N = \boldsymbol{\lambda}_N dt + \mathbf{P}_N d\eta$ and $d\boldsymbol{\Lambda}_T = \boldsymbol{\lambda}_T dt + \mathbf{P}_T d\eta$ are the differential measures with $\mathbf{P}_N d\eta$ and $\mathbf{P}_T d\eta$ representing the so-called atomic parts, i.e. the impulsive part of the impact and friction forces with $d\eta = \delta(t_i) dt$, where $\delta(t_i)$ is the Dirac delta function at t_i . Notice that (6) is now written as (measure) differential equations and (implicit) contact equations.

3 Gait Design

There are basically two ways to design a walking gait: (i) by deriving reference trajectories for each robot joint; (ii) by designing control laws for the robot dynamics. In this section, we present two methods for the gait design that lead to different stable walking patterns.

3.1 Gait Design Preliminaries: Domains

A walking gait can be divided into several domains, as shown in Fig. 2. Each domain has own properties, needs own reference trajectories and control strategy.

1. *Push off*: At $t = t_1$, the robot is in double support. The goal is to lift the swing leg, by pushing off or by lifting the swing foot from the ground.
2. *Single support*: This domain starts at $t = t_2$ when all contact points on the swing foot become inactive. Often (not always), the goal is to reach knee lock of the swing leg by swinging the leg forward.
3. *Strike*: At $t = t_3$, the swing knee gets locked. The goal is to drive the swing foot to the ground.
4. *Double support*: At $t = t_4$, one of the contact points on the swing foot becomes active. At the same time, the swing leg and stance leg swap their functions. The goal is to interchange the weight support from one leg to another. This domain ends when the robot is ready to push off at $t = t_5$. After this, the gait moves to domain 1 again.

3.2 ZMP Based Gait Design

The method of ZMP is well-known and often used to design stable bipedal walking gaits. The ZMP is the location on the ground where the net moment generated from the ground reaction forces is strictly perpendicular to the ground [12]. As long as the ZMP lies inside the support polygon of the stand foot/feet, the biped does not tip over. When the ZMP lies on the edge of the support polygon, the robot might start tipping. Here, we present per domain shown in Fig. 2(a) how to design trajectories in each robot joint that satisfy the ZMP criterion for stable bipedal walking.

1. In this domain, the stance leg keeps the same configuration, $\mathbf{q}_{a1}(t) = \mathbf{q}_{a4}(t_1)$. This configuration is chosen such that ZMP remains above the stance foot during the complete swing phase. The swing foot needs to be lifted a distance h_{i1} from the ground. The initial and end positions of the swing ankle with respect to the hip are given by $\mathbf{p}_{i1}(t_1) := [x_{i1}, z_{i1}]$ and $\mathbf{p}_{i1}(t_2) := [x_{i1}, z_{i1} + h_{i1}]$, respectively. A Cartesian trajectory describing location of the swing ankle is then determined by fitting a sufficiently smooth function between $\mathbf{p}_{i1}(t_1)$ and $\mathbf{p}_{i1}(t_2)$. Trajectories with cosine velocity profiles and quintic polynomials [11] are standard options for such a fit. The swing leg joint trajectories $\mathbf{q}_{i1}(t)$ can now be calculated using the inverse kinematics.
2. For the same configuration in the stance leg as in the previous domain, which keeps ZMP above the stance foot, we have: $\mathbf{q}_{a2}(t) = \mathbf{q}_{a1}(t_2)$. The swing foot moves forward for a distance l_{i2} until the swing knee is locked. The initial and end positions of the swing ankle with respect to the hip are $\mathbf{p}_{i2}(t_2) := [x_{i2}, z_{i2}]$ and $\mathbf{p}_{i2}(t_3) := [x_{i2} + l_{i2}, z_{i2}]$, respectively. By fitting a sufficiently smooth Cartesian trajectory between $\mathbf{p}_{i2}(t_2)$ and $\mathbf{p}_{i2}(t_3)$, the swing leg joint trajectories $\mathbf{q}_{i2}(t)$ can be calculated using the inverse kinematics.

3. The stance leg keeps the same configuration in this domain, $\mathbf{q}_{a3}(t) = \mathbf{q}_{a2}(t_3)$, so ZMP remains above the stand foot. The swing moves downwards for a distance $h_{i3} = -h_{i1}$. The initial position and end positions of the swing ankle with respect to the hip are $\mathbf{p}_{i3}(t_3) := [x_{i3}, z_{i3}]$ and $\mathbf{p}_{i3}(t_4) := [x_{i3}, z_{i3} - h_{i1}]$, respectively. Again, a Cartesian trajectory fit from $\mathbf{p}_{i3}(t_3)$ to $\mathbf{p}_{i3}(t_4)$ and inverse kinematics calculations lead to the swing leg joint trajectories $\mathbf{q}_{i3}(t)$.
4. Both feet of the robot are on the ground, which means a reduction of one dof. Here, we cannot treat the motions of two legs separately, since their motions are coupled. To resolve this situation, we divide the domain into two parts. In the first part we declare the stance leg as the master and the swing leg as the slave. This means that the stance leg determines the motion and that the swing leg has to follow. These motions are further constrained such as that ZMP remains above the support polygon, which is in this domain determined by both stand feet. We define the initial position of the hip with respect to the stance ankle: $\mathbf{p}_{a4}(t_4) := [x_{a4}, z_{a4}]$. The end position is defined as the position in which the robot stands straight up: $\mathbf{p}_{a4}\left(\frac{1}{2}(t_4 + t_5)\right) := [x_{a4} + l_{a4}, z_{a4} + h_{a4}]$. Since the legs are interconnected at the hip, the swing leg should perform a motion such that the position of the hip with respect to the swing ankle \mathbf{p}_{i4} is the same as \mathbf{p}_{a4} in Cartesian coordinates. Using simple goniometric relations, \mathbf{p}_{i4} can be calculated from \mathbf{p}_{a4} . In the second part of this domain, the roles swap: the swing leg becomes the master and the stance leg the slave. We define the position of the hip with respect to the swing ankle: $\mathbf{p}_{i4}\left(\frac{1}{2}(t_4 + t_5)\right) := [x_{i4}, z_{i4}]$. The end position of this domain should be the initial one for domain 1: $\mathbf{p}_{i4}(t_5) := [x_{i4} + l_{i4}, z_{i4} + h_{i4}]$. Hence, the stance leg has to perform a motion which guarantees that \mathbf{p}_{a4} is the same as \mathbf{p}_{i4} in Cartesian coordinates. Again using simple goniometric relations, \mathbf{p}_{a4} can be calculated from \mathbf{p}_{i4} . Using a cosine velocity profile to smoothly connect $\mathbf{p}_{i4}(t_4)$ and $\mathbf{p}_{i4}(t_5)$, as well as $\mathbf{p}_{a4}(t_4)$ and $\mathbf{p}_{a4}(t_5)$, we find a Cartesian trajectory for the position of the hip with respect to the swing and stance ankles, respectively. The swing and stance leg joint trajectories $\mathbf{q}_{i4}(t)$ and $\mathbf{q}_{a4}(t)$ can now be calculated using the inverse kinematics of the swing and stance legs, respectively.

3.3 LCW Gait Design

In the recent years limit-cycle walking (LCW) gaits [3] attract increasing attention. The underlying rationale is the use of the natural dynamics of the biped during walking, which yields certain benefits. First of all, the walking speed can significantly be increased in comparison with the ZMP based gaits. Secondly, the energy efficiency can be increased if the biped correctly makes use of gravity. Thirdly, the LCW gaits usually look much more human-like than ZMP-based. A LCW biped is not locally stable at every time instant, which is also the case with human walking. Unfortunately, the design of LCW gaits is seemingly far more complicated than the design of ZMP-based gaits. In this section we propose a general method to design a stable bipedal LCW gait. This gait design is not based on predefined joint trajectories, but is a result of feedback control

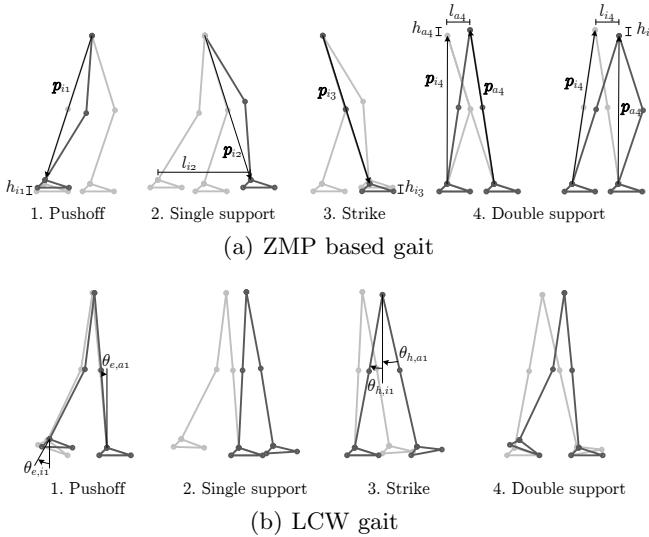


Fig. 2. Snapshots of the two different gaits

applied on the bipedal robot. We first present four controllers that are used in the different domains of bipedal walk.

- *Controlled symmetries*: To induce a passive behavior of the biped, this controller mimics the forces acting on the biped as if it would walk down a slope. The controller compensates for the actual gravitational forces, and at the same time it adds fictitious gravitational forces for the same robot configuration to mimic the dynamics of a passive dynamic walker of the same configuration as the considered bipedal robot [10], [9].

$$\tau_\beta = \mathbf{G}(\mathbf{q}) - \mathbf{G}(\mathbf{q} + \beta) \quad , \quad (7)$$

where $\tau_\beta \in \mathbb{R}^{n_j}$ is the torque applied to the n_j actuated joints and β is the angle of the fictitious slope.

- *Foot Scuffing Prevention*: An additional foot scuffing prevention controller is used to prevent the swing foot from scuffing. This controller calculates a vertical force which exponentially grows when the swing ankle approaches the ground. This force is virtually applied to the swing ankle by mapping it to swing knee and hip torques. To prevent the swing leg from slowing down, at the same time a fraction of this force is, in the same manner, virtually applied in horizontal direction to the swing ankle.

$$\tau_s = \mathbf{J}^T(\mathbf{q}_i) \begin{bmatrix} -\alpha_x e^{-\rho h_i(\mathbf{q}_i)} & -\alpha_z e^{-\rho h_i(\mathbf{q}_i)} \end{bmatrix}^T \quad , \quad (8)$$

where $\tau_s \in \mathbb{R}^{n_i}$ is the torque applied to the n_i joints in the swing leg, $\mathbf{J}(\mathbf{q}_i)$ is the Jacobian of the swing leg to map the force to the joint torques, h_i is the height of the swing ankle, and α_x , α_z and ρ are tunable controller parameters.

- *Ankle Controller*: The ankle controller is a spring damper controller which adds some compliance to the ankle:

$$\tau_e = K_e^p (\theta_e^r - \theta_e) - K_e^d \dot{\theta}_e , \quad (9)$$

where θ_e^r is the reference angle, θ_e is the actual ankle angle, and K_e^p and K_e^d are tunable controller parameters.

- *Step Size Controller*: The step size controller regulates the angle between the swing and stance leg, and in that sense also regulates the size of a step. The controller is again a spring damper controller:

$$\tau_h = K_h^p (\theta_h^r - \theta_h) - K_h^d \dot{\theta}_h , \quad (10)$$

where θ_h^r is the reference angle, θ_h is the actual ankle angle, and K_h^p and K_h^d are tunable controller parameters.

In this work, the parameters of the given controllers are determined by trial and error. In the future, analytical computation of these parameters will be considered. Here, we explain how the controllers are used to achieve stable bipedal walk in different domains illustrated in Fig. 2(b).

1. In this domain, only the ankle controller is applied in the swing ankle with the reference angle $\theta_e^r = \theta_{e,i1}^r > 0$ to achieve push off, while in the stance ankle it is applied with $\theta_e^r = \theta_{e,a1}^r = 0$ to push the stance leg straight up.
2. The controlled symmetries controller is used in this domain to mimic the forces as if the robot was on a shallow slope. Additionally, the ankle controller is used in the swing ankle to keep the swing foot horizontally with respect to the ground; $\theta_e^r = \theta_{e,i2}^r(t)$, where $\theta_{e,i2}^r(t)$ is the angle between the current swing foot orientation and the ground. The ankle controller is also used in the stance ankle with $\theta_e^r = \theta_{e,a2}^r = 0$ to keep the stance leg straight up. Furthermore, the scuffing prevention controller is used on the swing ankle to prevent the swing foot from hitting the ground.
3. Here, the controlled symmetries controller is used again. Additionally, the step size controller is used in the stance hip with $\theta_h^r = \theta_{h,i3}^r > 0$ and in the swing hip with $\theta_h^r = \theta_{h,a3}^r = -\theta_{h,i3}^r$ to force the biped to make a certain step with predefined step size. Furthermore, the ankle controller is used to add some compliance in the swing and stance ankles, both with $\theta_{e,i3}^r = \theta_{e,a3}^r = 0$.
4. Here, the controlled symmetries controller and the ankle controller are used. The ankle controller works with $\theta_e^r = \theta_{e,i4}^r = \theta_{e,a4}^r = 0$ for both the swing and stance ankle.

4 Stability Analysis

Among the existing methods for stability analysis, the ZMP and the Poincaré return map methods are mostly used. In this section, we show conservatism of these methods by means of simulations. According to the definitions given in [7], a bipedal walking is stable if it is carried out without falling. A biped falls when any other point than points on the feet come in contact with the ground. Hence, we consider situations where the robot is allowed to touch the ground only with its feet. Sitting and crawling fall out of our scope.

4.1 ZMP Gait Analysis

We calculate the ZMP to qualify stability of a given gait. The ZMP can be determined by the moment balance around a point p on the foot with position \mathbf{p}_p with respect to the base frame [12]:

$$\mathbf{M}_p = \dot{\mathbf{H}} - \mathbf{p}_c \times m_t \mathbf{g} + (\dot{\mathbf{P}} - m_t \mathbf{g}) \times \mathbf{p}_p , \quad (11)$$

where $\mathbf{M}_p = [M_x, M_y, M_z]^T$ is the moment acting in point p , $\dot{\mathbf{H}} = [\dot{H}_x, \dot{H}_y, \dot{H}_z]^T$ and $\dot{\mathbf{P}} = [\dot{P}_x, \dot{P}_y, \dot{P}_z]^T$ are the time derivatives of the angular and linear momenta of the biped with respect to the base frame, respectively, m_t is the total mass of the biped, $\mathbf{p}_c = [x_c, y_c, z_c]^T$ is the position of the center of mass, and \mathbf{g} is the gravitational vector. By definition, the position of the ZMP, $\mathbf{p}_{ZMP} = [x_{ZMP}, y_{ZMP}, z_{ZMP}]^T$, is the point \mathbf{p}_p for which $M_x = M_y = 0$. From this we can calculate the planar position of the ZMP with respect to the base frame:

$$x_{ZMP} = \frac{m_t g x_c - \dot{H}_y}{m_t g + \dot{P}_z} . \quad (12)$$

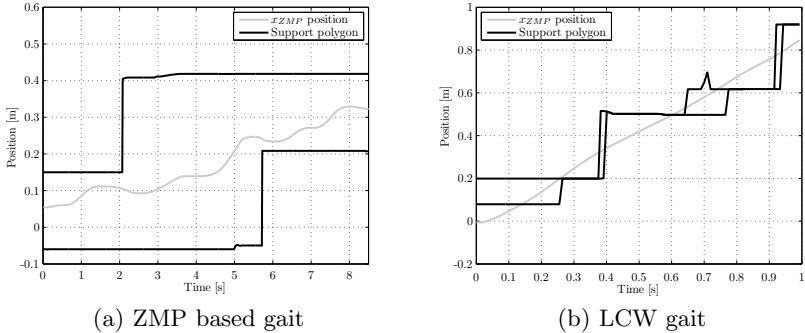
For small angular velocities, we use $\dot{H}_y = m_t (z_c \ddot{x}_c + x_c \ddot{z}_c)$ and $\dot{P}_z = m_t \ddot{z}_c$, in order to approximate (12) with the so called the cart-table model [8]:

$$x_{ZMP} = x_c - \frac{\ddot{x}_c}{g + \ddot{z}_c} z_c . \quad (13)$$

Now we can evaluate if the gaits illustrated in Figs. 2(a) and 2(b) are stable according to the ZMP criterion. For this, we simulate one step of the ZMP-based as well as the LCW gait. If the ZMP remains inside the support polygon, the ZMP criterion qualifies walking as stable. In Figs. 3(a) and 3(b), the ZMP trajectories for both gaits are plotted as functions of time, together with the support polygon limits. Please notice that for a planar biped, the support polygon is actually a line segment. These figures confirm what one would intuitively expect. The ZMP of the ZMP based gait stays inside the support polygon. According to the ZMP criterion this gait is stable. Nevertheless, the ZMP of the LCW gait does not stay inside the support polygon all the time. At time periods when the ZMP leaves the support polygon, the gait would be classified as unstable according to the ZMP stability criterion. This illustrative example let us realize that the ZMP stability criterion is too conservative to analyze the stability of an arbitrary gait. To the best of our knowledge, there is no similar illustration of conservatism of the ZMP criterion available in the literature.

4.2 Poincaré Return Map Gait Analysis

If a walking gait exhibits a cyclic pattern, then the biped realizing such a gait will return to the same state at the end of each cycle. One can consider a lower

**Fig. 3.** Position of the ZMP during one step

dimensional subspace \mathbf{S} of the system state-space, the Poincaré return map, which is intersected by the cyclic motion. The intersection point is called a fixed point. For the stable periodic walking gait, the system state-trajectories return to approximately the same state after every step. One can make a Poincaré map at, for example, every start of a step, just after heel strike. According to the terminology of bipedal locomotion [2], [3], this mapping of the cyclic nonlinear dynamics is called the stride function. The stride function determines a transition between the current state and the state after one cycle:

$$\mathbf{q}_{k+1} = \mathbf{S}(\mathbf{q}_k) . \quad (14)$$

Only if the motion is perfectly cyclic, then the state \mathbf{q} is a fixed point \mathbf{q}_f :

$$\mathbf{q}_f = \mathbf{S}(\mathbf{q}_f) . \quad (15)$$

The stability of the cyclic motion can be analyzed by perturbing the initial fixed point and checking if it returns to the fixed point after a finite number of cycles. Linearizing the stride function around these perturbations can tell us if the state will return to the fixed point:

$$\mathbf{S}(\mathbf{q}_f + \Delta\mathbf{q}) \approx \mathbf{q}_f + \mathbf{K}\Delta\mathbf{q} , \quad (16)$$

where $\mathbf{K} = \frac{\partial\mathbf{S}}{\partial\mathbf{q}}$ is the linear return matrix. This matrix determines if the state of the system returns to the fixed point for small perturbations. Namely, the motion is considered as stable if the eigenvalues of the matrix \mathbf{K} fall inside the unit circle. If this is the case, then it is expected that the system state monotonously converges to the fixed point after each cycle. The smaller the absolute values of the eigenvalues, the faster the convergence to the limit cycle. Because of complexity of the biped dynamics, it is virtually impossible to find the stride function analytically. Therefore, the linear return matrix should be estimated by a finite difference approximation [2]. By doing so, we can analyze the gaits illustrated in Figs. 2(a) and 2(b) using the Poincaré return map stability criterion. One step of each gait is simulated as many times as the number of states

Table 1. Eigenvalues of Poincaré return map

λ	ZMP	LCW
λ_1	0.3178	-0.4873
λ_2	$-0.0285 + 0.0730i$	$0.1326 + 0.0819i$
λ_3	$-0.0285 - 0.0730i$	$0.1326 - 0.0819i$
λ_4	$0.0545 + 0.0578i$	0.0222
λ_5	$0.0545 - 0.0578i$	-0.0159
λ_6	0.0127	0.0005
λ_7	$-3.1771e^{-4} + 4.2667e^{-4}i$	$1.5954e^{-5} + 4.5911e^{-5}i$
λ_8	$-3.1771e^{-4} - 4.2667e^{-4}i$	$1.5954e^{-5} - 4.5911e^{-5}i$
λ_9	$1.4851e^{-4}$	$-1.5233e^{-5}$
λ_{10}	$-2.7185e^{-5}$	$-1.9101e^{-5}$
λ_{11}	$1.2318e^{-5}$	$-3.1599e^{-6}$
λ_{12}	$-4.7939e^{-6}$	$8.8410e^{-7}$
λ_{13}	$-1.9989e^{-6}$	$-3.8626e^{-8}$
λ_{14}	$-3.3579e^{-7}$	$9.6776e^{-10}$

of the considered planar bipedal robot. At the beginning of every simulation, one state-coordinate is perturbed with respect to the initial state of the considered gait. From the results at the end of the step, we can calculate the linear return matrix and analyze the eigenvalues for each gait. The results that correspond to the two different gaits are presented in Table II. The given results show that all eigenvalues of each gait lie within the unit disc. Consequently, the designed gaits are cyclically stable according to the Poincaré criterion. However, in the derivation of the stride function, we use a linearization of the system dynamics that are highly nonlinear and of a switching type. As shown in [4], the Poincaré map for the switching type dynamics may experience further difficulties than we know from systems which are only nonlinear. A specific concern in this regard is whether the matrix \mathbf{K} is well-defined. The Poincaré stability criterion can be applied to cyclic gaits only. The class of the periodic gaits is rather restrictive, which implies conservatism of the Poincaré stability method. In this aspect, the ZMP criterion is less conservative than the method of Poincaré, since periodicity of the gait is not a necessary condition for application of the ZMP criterion.

5 Conclusion

We model a bipedal robot as a system with unilateral constraints. This approach allows for the proper integration of ground contact and tangential friction force effects. In addition, a new methodology for the systematic design of limit-cycle walking (LCW) gaits is proposed. Finally, the planar biped model is used for simulation of two different walking gaits, based on the zero moment point (ZMP) and LCW principles, respectively. By means of simulations, we show serious conservatism of the two mostly used criteria for stability analysis of bipedal walking: the ZMP criterion and the Poincaré return map method. Our results illustrate that fulfillment of the ZMP criterion is sufficient for stability, but this

criterion is applicable to just a portion of complete state-space of the bipedal robot. Using the Poincaré return map method, we demonstrate that violation of the ZMP criterion does not necessarily mean that stable walking is not possible. Unfortunately, the Poincaré method can only be used to analyze the stability of periodic gaits only, which is a serious limitation. Periodicity of a gait is not necessary for application of the ZMP criterion. Consequently, neither criterion does offer a general stability qualification that includes all possible gaits an arbitrary biped can make. This raises a concern that the existing stability criteria are still in infancy regarding general qualification of stability of bipedal walking. Consequently, further research is needed to achieve a general stability criterion.

References

1. Goswami, A.: Foot rotation indicator (FRI) point: A new gait planning tool to evaluate postural stability of biped robots. In: IEEE International Conference on Robotics and Automation, pp. 47–52 (1999)
2. Goswami, A., Espiau, B., Keramane, A.: Limit cycles in a passive compass gait biped and passivity-mimicking control laws. *Autonomous Robots* 4, 273–286 (1997)
3. Hobbelin, D., Wisse, M.: Limit cycle walking. *Humanoid Robots, Human-like Machines* (2007)
4. Leine, R., Nijmeijer, H.: Dynamics and bifurcation of non smooth mechanical systems. Springer, Berlin (2004)
5. Leine, R., van de Wouw, N.: Stability and convergence of mechanical systems with unilateral constraints. Springer, Berlin (2008)
6. Moreau, J.: Unilateral contact and dry friction in finite freedom dynamics. In: Nonsmooth mechanics and Applications, pp. 1–82 (1988)
7. Pratt, J., Tedrake, R.: Velocity-based stability margins for fast bipedal walking. In: First Ruperto Carola symposium: fast motions in biomechanics and robots (2005)
8. Siciliano, B., Khatib, O.: Springer handbook of robotics. Springer-Verlag New York Inc., Heidelberg (2008)
9. Sinnet, R., Ames, A.: 2d bipedal walking with knees and feet: a hybrid control approach. In: Joint 48th IEEE Conference on Decision and Control and 28th Chinese Control Conference, pp. 3200–3207 (2009)
10. Spong, M., Bullo, F.: Controlled symmetries and passive walking. *IEEE Transactions on Automatic Control* 50(7), 1025–1031 (2005)
11. Spong, M., Hutchinson, S., Vidyasagar, M.: Robot modeling and control. John Wiley and Sons, Inc., New York (2006)
12. Vukobratovic, M., Borovac, B., Surla, D., Stokic, D.: Scientific Fundamentals of Robotics 7: Biped Locomotion. Springer, New York (1990)
13. Westervelt, E., Grizzle, J., Chevallereau, C., Choi, J., Morris, B.: Feedback control of dynamic bipedal robot locomotion. CRC Press, New York (2007)
14. Wight, D.L., Kubica, E.G., Wang, D.W.L.: Introduction of the foot placement estimator: A dynamic measure of balance for bipedal robotics. *Journal of Computational and Nonlinear Dynamics* 3, 1–9 (2008)

An Approach to Close the Gap between Simulation and Real Robots

Yuan Xu, Heinrich Mellmann, and Hans-Dieter Burkhard

Institut für Informatik, LFG Künstliche Intelligenz, Humboldt-Universität zu Berlin,
Rudower Chaussee 25, 12489 Berlin, Germany
{xu,mellmann,hdb}@informatik.hu-berlin.de

Abstract. Numerous simulators have been developed over the years to assist robotics research in the development, testing, and evaluation. Nevertheless, there is still a big gap between the simulation and the reality. This makes it difficult to transfer methods and code. The 3D simulator — SimSpark is developed and used by a big community of AI researchers in RoboCup. But up to now there are only few applications to real robots. In this paper, we discuss the general possibilities how the SimSpark simulator can be used to support research in cognitive robotics and present applications on the humanoid robot Nao. As a result of our investigation we have developed a unified team playing both in Simulation League and Standard Platform League in RoboCup.

1 Introduction

The Robot World Cup (RoboCup) initiative is an attempt to foster artificial intelligence and intelligent robotics research by providing a standard problem where a wide range of technologies can be integrated and examined [8]. There are different leagues that make it attractive as an environment for researchers to focus on a set of specific problems on the way to the final goal — “By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team.”

At present, there are several different leagues, which are focus on different subtasks. However, there is a lot of work being repeated in the different leagues while solutions for the same and similar issues exist in another league. For example, self localization, biped locomotion and etc. It is useful to achieve synergy effects for the same challenges in different leagues.

Numerous simulators have been developed over the years to assist in the development, testing, and evaluation of robots. Every robotics research group uses simulators to develop their robots. Designing and implementing a good robot simulator is a difficult and time consuming task, so it makes sense to reuse the existing work. And also, the simulation teams want to apply their solutions to real robot. It makes sense to integrate simulation team and real robot team.

There are already some collaborations between researchers from different leagues. Dylla et al. [5] investigated to model soccer knowledge in a way that they are usable for multiple RoboCup soccer leagues. Mayer et al. [11] proposed

a road map to collaboration between Soccer Simulation League and Humanoid League. At the same time, the USARsim [15] simulator is used as bridging tool between the Rescue Real Robot League and the Rescue Simulation League.

In this paper, our investigations concerning simulation and real robot focus on Standard Platform League and Simulation League. Since 2008, the Standard Platform League and 3D Simulation League use the same robot — Nao from Aldebaran Robotics. Our team — Nao Team Humboldt started at 2008 in Standard Platform League, and also participate in 3D Simulation League now. We use a common Core of our program for both platforms. Actually our framework runs on Webots and log simulator, too. But throughout the paper we will concentrate on the real robots and SimSpark — the simulator of 3D Simulation League.

This paper is organized as follows. Section 2 describes the current state of the RoboCup Simulation League and Standard Platform League. The implementation of our team is described in Section 3. Section 4 gives some experimental result. Section 5 discusses the problems which we are working on, followed by conclusion and future work in section 6.

2 Current State of the Leagues

Simulation League and Standard Platform League both have long histories and big communities. In this section, we describe the current state of the two leagues.

2.1 3D Simulation League

Simulation league exists from the very beginning of RoboCup in 1997. The 2D simulator only runs on a relatively abstract environment. The participants concentrate mainly on coordination and cooperation of robot teams. Because of the simplified model of 2D simulation league, a three-dimensional physical simulation [9] was introduced in RoboCup 2004, but only sphere shaped agents ran around the field. Thereby increasing the realism of the simulated environment and making it more comparable to the other RoboCup league environments; but still being able to simulate more players on the field than other leagues. Thus giving rise to higher level research on larger groups of slightly more realistic soccer playing robots.

Since 2007, the humanoid robots are introduced into the 3D Simulation League. This opens up opportunities for research on lower level control of humanoid robots as well as higher level behaviors in humanoid soccer and getting closer to how humans play the game. The participants have to create soccer playing agents for a team which have to deal with the higher complexity of a human shaped body in the (simulated) physical environment. Hopefully, the results can be transferred to humanoid robotics more easily, and our research investigates these connections.

As a consequence of the changes to the 3D simulator, a (temporary) shift of problem solving to the more basic problems of body control could be observed.

This is related to restrictions concerning tactics: how to plan a team play when the basic skills do not allow, e.g., for a good pass.

At the same time, simulation allows for more players than with real robots. In 2010, the field size of the 3D Simulation League increased to 15×10 meters, and the games are played in 6 against 6, see Fig. 1. This means, coordination and cooperation between robots becomes important again like in 2D simulation. It is a very interesting two tired problem: to develop the skills for better tactics on one hand, and to realize tactics based on the actually available skills on the other hand.



Fig. 1. A 6 vs. 6 game is playing in the 3D Simulation League at RoboCup 2010 in Singapore

2.2 Standard Platform League

In the Standard Platform League (SPL) all teams use the same robotic platform. On one hand it gives scientists the opportunity to concentrate on the software development only. On another hand it provides a better possibility to compare the developed algorithm in the game, since no team may gain advantages by changing the hardware of the robot (e.g., using stronger motors). Fig. 2 shows a scene from a SPL soccer game.

Like in the Simulation League the agents (i.e. robots) operate fully autonomous. In particular there is no external control neither by human or by computer. Perception is based on different sensors (cf. Table 1) and calculated by the on board computer. Communication is possible via wireless LAN, but it maybe corrupted by environmental conditions. The on-board computer is also used for planning and control. The runtime system provides special middle ware for the communication between software and hardware.

Until 2008 the four-legged Sony Aibo robot was used as the common platform. Starting from 2008 the humanoid robot Nao produced by Aldebaran is used. Because of 10 years difference, the Nao can benefit from progress in technology.

But, the restriction of processing power is still a bottle neck, because reactions have to be done in real time, e.g., perception and control calculations should be finished during the cycles given by the frequency of images (30 frames per second). Otherwise, the robot would act on obsolete data. The runtime system uses a special middle ware, the Naoqi.

Of course, biped motion is much more complicated than quadruped motion, and skills like kicking and dribbling are still under development. For the Aibo, the RoboCup teams could provide a walk of about 50cm/second which was about three times faster than the original walk, this was mainly done using Machine Learning.



Fig. 2. A 3 vs. 3 game is playing in the Standard Platform League at the German Open 2010 in Magdeburg

2.3 Comparison of Two Leagues

Since Standard Platform League and 3D Simulation League use the same robot — Nao from Aldebaran Robotics, it provides a good opportunity to cooperate between two leagues, but there are also some difference between them. In this subsection, we investigate the difference and consistency of the two platforms.

First of all, the simulated robot in 3D Simulation League has some differences to real Nao robot, see Table 1. In the real Nao, the left hip and right hip are physically connected one motor so they cannot be controlled independently, but there are two motors in the simulated robot. The robot program has to communicate with NaoQi, but the network connection is established between agent program and simulator.

The vision of real Nao is based on CMOS camera, it can receive 640×480 pixels image in 30 FPS. The image processing is not needed in simulation up to now, since the robot receives abstract vision percepts from simulator directly. The vision percepts of each object contains distance and angle relative to the

Table 1. Comparison between real Nao robot and simulated robot in 3D Simulation League

	Standard Platform League	3D Simulation League
Degree of freedom	25	26
Kinematics	the same	the same
Joint control	angle	velocity
Vision	2 cameras	vision information
Accelerometer	3 axes	3 axes
Gyrometer	2 axes	3 axes
Force sensor in each foot	4 force sensitive resistors	6 dimensions force sensor
LEDs	yes	no
Loudspeakers & Microphones	yes	no
Sonars	yes	no
IR transmitter/receiver	yes	no
Platform interface	NaoQi	TCP/IP

camera coordinates. The real robot has to calculate related percepts by image processing. Using percepts, the real robots as well as the simulated robots have to calculate higher level information, e.g., for localization and speed of objects.

The head, body, hands and feet of other robots are given by the vision percepts in simulation. It provides much more information than real images, so it is possible that the robot can better recognize the situation of the environment and perform more complex tasks.

The simulated force sensor in the foots also provides more information than for the real robot, since it provides force vector and force point at the same time. The gyrometer has one axis more than in the real robot.

Besides the robot, the game in different leagues are organized in different ways, see Table 2. The field size and goal width in 3D Simulation League is bigger than for Standard Platform League. Each team has 6 robots in Simulation League while only 3 robots are used in Standard Platform League. The duration of one Standard Platform League game is twice long as simulation game. In comparison to Standard Platform League, the rules of 3D Simulation are closer to human soccer rules.

Table 2. Comparison of games in Standard Platform League and 3D Simulation League

	Standard Platform League	3D Simulation League
Field size	$6m \times 4m$	$15m \times 10m$
Goal size	$1.4m \times 0.8m$	$2.1m \times 0.8m$
Number of robots	3	6
Game	without stop	with kick in, goal kick and etc.
Duration	20 min	10 min

3 Implementation

In order to play both in Simulation 3D League and Standard Platform League, the architecture strives to seamlessly integrate simulation system with real robotic hardware, and allows simulated and real architectural components to function seamlessly at the same time. And also it should be able to integrate existing available infrastructures.

3.1 Outlines/Architecture

Since the Humboldt-Universität has a long history in RoboCup, there is a lot of experience and already existing code, especially from the GermanTeam [13] with its module based architecture. In oder to integrate different platforms, our project is divided into two parts: platform independent part and platform specific part. The platform independent part is the Core, which can run in any environment, and all the algorithms are implemented here. The platform specific part contains code which is applied to the particular platform. In the core part, several different modules are implemented under the module based architecture. In this section, we briefly describe our modules.

The recent perception model in 3D simulation league is based on already “preprocessed” information provided by the soccer server. For the real robots, we have developed basic methods as well, which might be useful in the future 3D league. Basic image processing as color classification is performed using a 80×60 pixel grid. Object detection (ball, goals, lines, robots) applies classical methods, e.g., region growing. The camera matrix is computed by the kinematic chain (alternative approaches using constraints are under investigation). Constraints are already used for navigation and world model as well as particle filters. A Kalman filter is applied to generate a local ball model. Global ball positions are communicated between players to coordinate team behavior.

Behavior is executed through a hierarchical state machine, known as XABSL (Extensible Agent Behavior Specification Language) [10]. The behavior code generation is supported by the related editor, debugging, and visualization tools. It allows layered usage of different behaviors, from very low-level motions to high-level cooperative team skills. Each option in the behavior tree decides the running time of it's direct active sub-options in case of being activated itself, though each one takes care of it's own timing. Furthermore, options have some internal state criteria, used as inter-option communication means, mainly utilized by sub-options and their super options. Each option can also have a possible transition condition, through which along with the option's state propagation, the active behavior-switching in any decision level can be safely accomplished. In Section 4 we describe some experiments how behavior can be designed in XABSL in order to be used simultaneously in simulation and on a real robot.

Motions are implemented by keyframe techniques and inverse kinematics with motion planning. Keyframe nets were developed through teaching and hand coding with the help of our motion editor framework. For parameterized dynamic

motions as omnidirectional walks and kicks we use and investigate inverse kinematics and the sensor feedback [12].

Debugging code can be switched on/off during runtime. Debug results are transferred over the network and monitored/visualized using RobotControl, a robot data monitoring and debugging tool, which is implemented in Java to be used on arbitrary computer systems, see Fig. 3. It is a good tool to analysis and debug one robot, but in order to analysis and develop the team behavior of a robot soccer team, we need to connect to all the robots at the same time. A related new tool — TeamControl is under the development now.

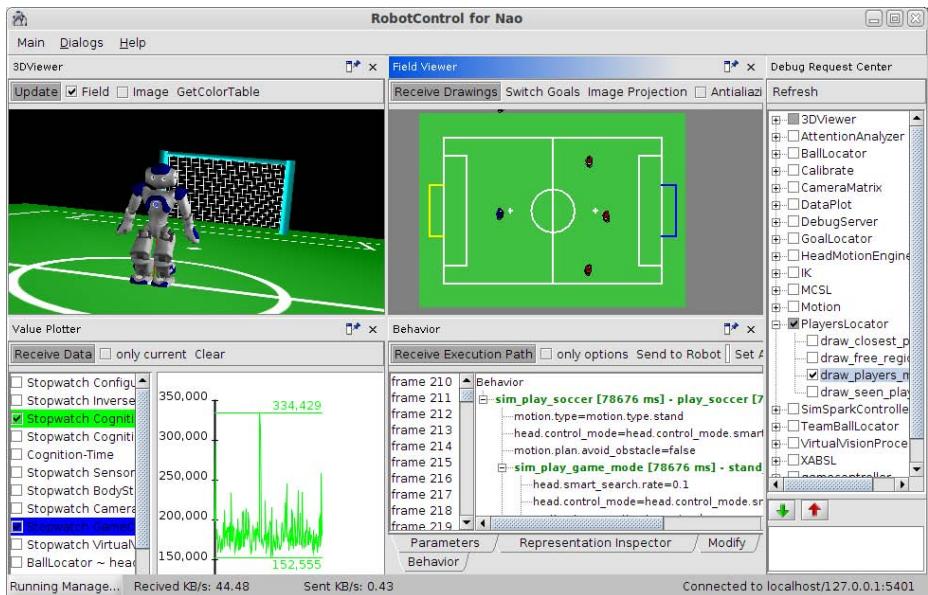


Fig. 3. The RobotControl program contains different dialogs. In this figure, the left top dialog is the 3D viewer, which is used to visualized the current state of robot; the left bottom dialog plots some data; the middle top dialog draws the field view; the middle bottom shows the behavior tree; and the right one is the debug request dialog which can enable/disable debug functionalities.

3.2 Unified Platform Interface

At present, our agent (robot control program) can run in 4 different platforms: real *Nao* robot, *Webots* simulator, *SimSpark* simulator and *log simulator*. An abstraction interface is implemented in the sense - think - act loop. With the unified platform interface(see Fig. 4), the core of our program is independent of the platform which it runs, and all the platform dependent codes are in separate platform implementations.

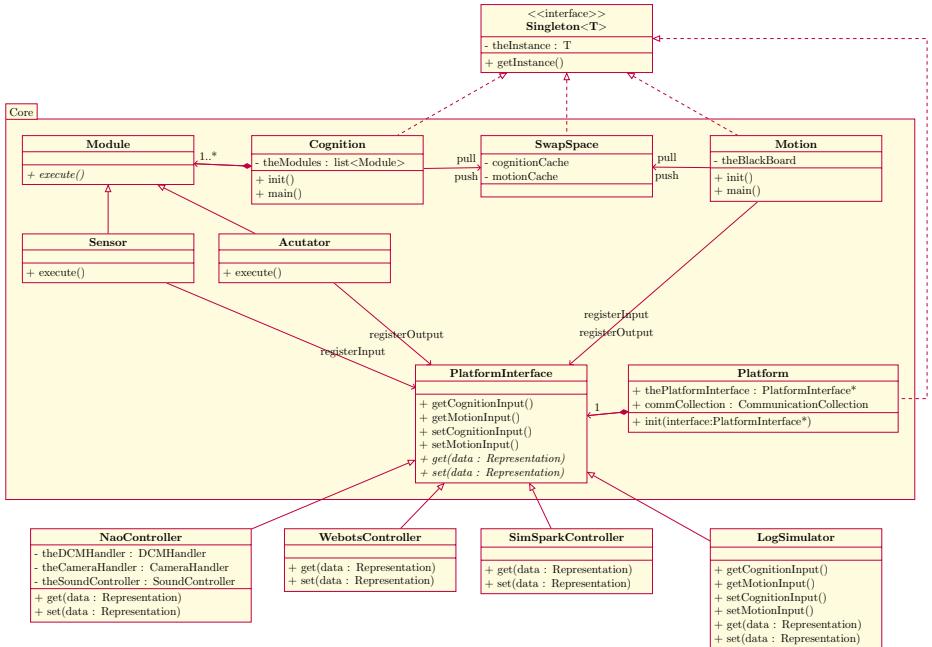


Fig. 4. Framework of unified platform interface, the core of program can run in different platforms with different modules

As mentioned in section 2, there are some difference between simulation and real robot. Comparing to real Nao robot, some devices are missing in the simulations, such as LEDs and sound speaker. In this case, the platform skips unsupported devices, and uses different modules for different platforms. For example, camera (image sensor) is not used in 3D simulation league. Thus, the controller can disable the image processing module, use the virtual *see* sensor of the simulator SimSpark to provide perceptions. Therefore, the core can run on different platforms and enable different modules for different sensors and actuators.

4 Experiments and Results

With the implementation of architecture and modules, our robot program can be executed on different platforms. In order to play both in 3D Simulation League and Standard Platform League, related experiments were performed and investigated, especially for behavior and motion.

The behavior of our robot program is build by XABSL (cf. 3.1). It is relatively easy to describe some behavior from soccer theory. For example, the role changing can be described by 4 basic states, see Fig. 5. But the decision of changing between states depends heavily on the characteristics of the robot and the environment, i.e. the conditions for decisions are different in different leagues. Currently, some parameters are chosen specially for different leagues.

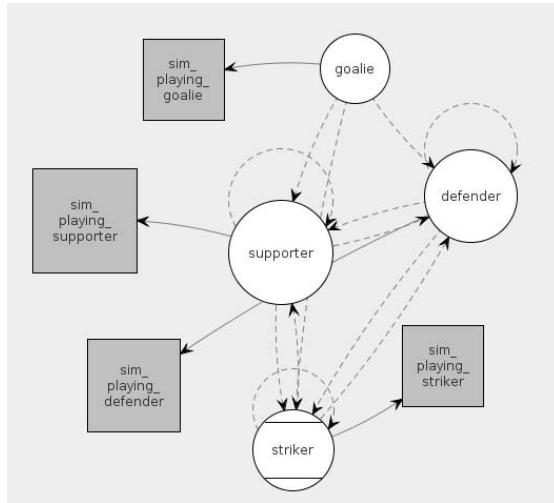


Fig. 5. The role changing behavior described by XABSL

The better alternative is creating some symbols by lower level control modules, which makes the decision processes more general. For example, the motion module provides parameters of motion skills like the walking speed. Then the behavior module can base its decisions on real values. We have successfully used related behaviors to play in both leagues, i.e. for skills like go to ball and kick, and dribble, respectively.

Our motions are implemented by key frame techniques and parameterized dynamic motions. For key frame motion, we adjust key frames for the different platforms. For parameterized dynamic motions, evolutionary approaches are used to optimize the parameters. For example, our biped walking is implemented for the real robot with some parameters which are tested on the real Nao. Basically, it also works in simulation, but it is not fast enough to play against other teams. We use evolution algorithm as well to optimize the 18 parameters of our walking in simulation. After 50 generations with 100 individuals in each generation, our robot runs at the same speed as the fastest team in simulation league.

With these experiments, our team successfully participated RoboCup German Open 2010, and won the 4th place in Standard Platform League and 1st place in 3D simulation league. It was the first time that the same source code played in two leagues: both simulation and real robot.

5 Discussion

We have developed a framework for running robots both in real and simulated settings. A main challenge is the usage of unified code as much as possible. Now, the performance of a control program in the simulation can be compared with on

a real robot. Differences occur systematically from differences between sensors and actuators.

Unrealistic assumptions about the statistics of the sensor input will result in differences on the higher level control. Sensory inputs in real robots are very noisy and biased data. The difference between the simulated sensory input and the sensory input that comes from a real world humanoid robot system can be directly recorded and compared. In this way we can get accurate statistics and can integrate the results into the simulated sensory environment. It is possible to establish feedback from the reality to the simulation league, and the related statistics can lead to stepwise improvement of the 3D simulator.

As another common experience, there are big gaps between simulation and reality in basic physics with consequences for low level skills in motions. An challenging question is the transferring of learned skilled from simulation to reality. We investigate the relationships and possibilities for the transferring of methods and code. Again, consequences can lead to better simulation tools, especially in the 3D Simulation League.

At the higher level, the aim is to use the same tactics in simulation and in reality. Simulation still provides better opportunities to develop and test coordination strategies (e.g., by larger fields with more players). Here, simulation will be the driving force for better play.

So far, we have developed walking gaits through evolution techniques in a simulated environment [7,6]. Reinforcement Learning was used for the development of dribbling skills in the 2D simulation [14], while Case Based Reasoning was used for strategic behavior [4,2]. BDI-techniques have been investigated for behavior control, e.g., in [13].

6 Conclusion

We investigated Standard Platform League and Simulation League for getting simulation and real robot closer. Our unified interface and modular architecture are presented. As a result, our team — Nao Team Humboldt uses a common Core of our program for both platforms. We successfully participated RoboCup German Open 2010, and won the 4th place in Standard Platform League and 1st place in 3D simulation league. It was the first time that the same source code played in two leagues: both simulation and real robot.

The key point to success is narrowing the gap between simulation and reality. There are some researchers who have already tried to achieve this, but there are only few successful results so far. At first, the relations (differences and similarities) between reality and simulations should be better understood. This will help to develop more realistic simulations. Simulations than can help to develop methods and code for real robots in an easier way. Our plan is to analyze data from sensors/actuators in simulation and from real robots at first and then to apply machine learning methods to improve the available model or to build a good implicit model of the real robot from the data.

Acknowledgments

The authors would like to thank other members of Nao Team Humboldt.

References

1. Berger, R.: Die Doppelpass-Architektur. Verhaltenssteuerung autonomer Agenten in dynamischen Umgebungen. Diploma thesis, Humboldt-Universität zu Berlin, Institut für Informatik (2006) (in German)
2. Berger, R., Lämmel, G.: Exploiting Past Experience. Case-Based Decision Support for Soccer Agents. In: Hertzberg, J., Beetz, M., Englert, R. (eds.) KI 2007. LNCS (LNAI), vol. 4667, pp. 440–443. Springer, Heidelberg (2007)
3. Burkhard, H.D.: Programming Bounded Rationality. In: Proceedings of the International Workshop on Monitoring, Security, and Rescue Techniques in Multiagent Systems (MSRAS 2004), pp. 347–362. Springer, Heidelberg (2005)
4. Burkhard, H.-D., Berger, R.: Cases in robotic soccer. In: Weber, R.O., Richter, M.M. (eds.) ICCBR 2007. LNCS (LNAI), vol. 4626, pp. 1–15. Springer, Heidelberg (2007)
5. Dylla, F., Ferrein, A., Lakemeyer, G., Murray, J., Obst, O., Röfer, T., Stolzenburg, F., Visser, U., Wagner, T.: Towards a league-independent qualitative soccer theory for robocup. In: Nardi, D., Riedmiller, M., Sammut, C., Santos-Victor, J. (eds.) RoboCup 2004. LNCS (LNAI), vol. 3276, pp. 29–40. Springer, Heidelberg (2005)
6. Hein, D.: Simloid – Evolution of Biped Walking Using Physical Simulation. Diploma thesis, Humboldt-Universität zu Berlin, Institut für Informatik (2007)
7. Hein, D., Hild, M., Berger, R.: Evolution of biped walking using neural oscillators and physical simulation. In: Visser, U., Ribeiro, F., Ohashi, T., Dellaert, F. (eds.) RoboCup 2007: Robot Soccer World Cup XI. LNCS (LNAI), vol. 5001, pp. 433–440. Springer, Heidelberg (2008)
8. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E.: RoboCup: The robot world cup initiative. In: Johnson, W.L., Hayes-Roth, B. (eds.) Proceedings of the First International Conference on Autonomous Agents (Agents 1997), pp. 340–347. ACM Press, New York (5–8, 1997), citeseeer.ist.psu.edu/kitano95robocup.html
9. Kögler, M., Obst, O.: Simulation league: The next generation. In: Polani, D., Browning, B., Bonarini, A., Yoshida, K. (eds.) RoboCup 2003. LNCS (LNAI), vol. 3020, pp. 458–469. Springer, Heidelberg (2004)
10. Lötzsch, M., Risler, M., Juengel, M.: Xabsl - a pragmatic approach to behavior engineering. In: Proceedings of IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS), Beijing, China, October 9–15, pp. 5124–5129 (2006)
11. Mayer, N.M., Boedecker, J., da Silva Guerra I, R., Obst, O., Asada, M.: 3d2real: Simulation league finals in real robots. In: Lakemeyer, G., Sklar, E., Sorrenti, D.G., Takahashi, T. (eds.) RoboCup 2006: Robot Soccer World Cup X. LNCS (LNAI), vol. 4434, pp. 25–34. Springer, Heidelberg (2007)
12. Mellmann, H., Xu, Y.: Adaptive motion control with visual feedback for a humanoid robot. In: Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (to appear, 2010)

13. Röfer, T., Brose, J., Göhring, D., Jüngel, M., Laue, T., Risler, M.: GermanTeam 2007 - The German national RoboCup team. In: RoboCup 2007: Robot Soccer World Cup XI Preproceedings. RoboCup Federation (2007)
14. Uc-Cetina, V.: Reinforcement Learning in Continuous State and Action Spaces. Ph.D. thesis, Humboldt-Universität zu Berlin (2009)
15. Wang, J., Balakirsky, S.: USARSim — A Game-based Simulation of the NIST Reference Arenas, University of Pittsburgh (May 2005),
<http://sourceforge.net/projects/usarsim/>

Accelerating Point-Based POMDP Algorithms via Greedy Strategies*

Zongzhang Zhang and Xiaoping Chen

University of Science and Technology of China, Hefei, China
`zzz@mail.ustc.edu.cn, xpchen@ustc.edu.cn`

Abstract. Many planning tasks of autonomous robots can be modeled as partially observable Markov decision process (POMDP) problems. Point-based algorithms are well-known algorithms for solving large-scale POMDP problems. Several leading point-based algorithms eschew some flawed but very useful heuristics to find an ϵ -optimal policy. This paper aims at exploiting these avoided heuristics by a simple framework. The main idea of this framework is to construct a greedy strategy and combine it with the leading algorithms. We present an implementation to verify the framework's validity. The greedy strategy in this implementation stems from some common ignored heuristics in three leading algorithms, and therefore can be well combined with them. Experimental results show that the combined algorithms are more efficient than the original algorithms. On some benchmark problems, the combined algorithms have achieved about an order of magnitude improvement in runtime. These results provide an empirical evidence for our proposed framework's efficiency.

Keywords: planning of autonomous robots, POMDP, point-based algorithms, framework, greedy strategy, reinforcement learning.

1 Introduction

Partially observable Markov decision process (POMDP) has been widely recognized as a powerful probabilistic model for planning of autonomous robots in uncertain environments. It is often challenging to exactly solve POMDP problems due to their computational complexity, however. In recent years, various efficient approximate algorithms have been proposed and have been successfully applied to various robotic tasks [12]. Among them, point-based algorithms are a family of particularly efficient algorithms [34]. They focus on sampling a set of representative beliefs that are reachable from the initial belief, and are adept at tackling large POMDP problems [3]. All point-based algorithms repeatedly take two steps. The first step is to sample a set of beliefs according to some rules.

* This work is supported in part by the Natural Science Foundations of China under Grant No. 60745002, and the National Hi-Tech Project of China under Grant No. 2008AA01Z150.

The second step is to perform an α -vector backup operation for each sampled belief in some order.

Several leading point-based algorithms have a wonderful property that an ϵ -optimal policy can be found theoretically for a POMDP problem given enough running time. To satisfy this property, these algorithms ignore some very useful heuristics. The main reason to do this is that these heuristics have some shortcomings, such as, sometimes easily trap algorithms into local optima. However, by avoiding these heuristics, algorithms lose a precious opportunity to further improve their performance.

This paper aims at taking advantage of these underused heuristics to accelerate the leading point-based algorithms. To address the local-optima issue, we propose a generic framework. The main idea of this framework is to construct a greedy strategy using these ignored heuristics and insert it as a component into the leading point-based algorithms (see Fig. II(a)). The combined algorithms use an ϵ -optimal strategy to guide search toward an optimal policy, and meanwhile, use a greedy strategy to explore some shortcuts via some extra heuristics (see Fig. II(b)). The use of a systematic search for finding a better policy with greedy action choices to exploit the best known decisions is reminiscent of exploration-exploitation tradeoff mechanisms in reinforcement learning [5]. However, to our knowledge, this idea hasn't been well applied into solving POMDP problems.

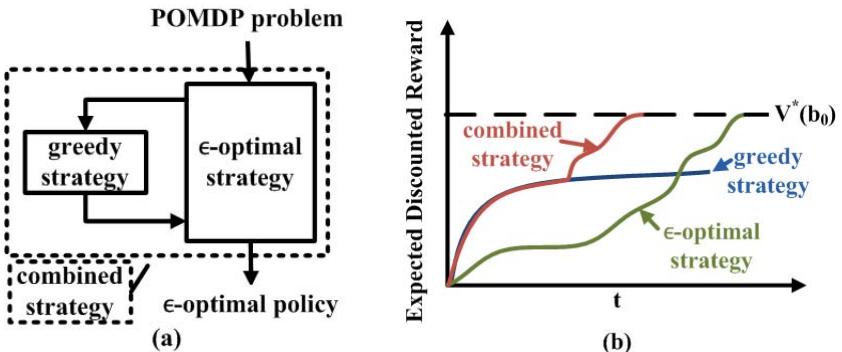


Fig. 1. (a) A generic framework, in which a combined strategy is a combination of an ϵ -optimal strategy and a greedy strategy. (b) The lines represent the process of finding a better policy (corresponding to a higher expected discounted reward) over time for different strategies. A combined strategy takes advantage of a greedy strategy to explore possible shortcuts, and hence has extra opportunities to speed up the convergence rate. V^* represents the optimal value function.

To verify this framework's utility, we designed a greedy algorithm using some common avoided or unnoticed heuristics among three leading point-based algorithms: heuristic search value iteration 2 (HSV2) [6], focused real-time dynamic programming (FRTDP) [3,7] and successive approximations of the reachable space under optimal policies (SARSOP) [4]. We called this greedy algorithm as the “second best policy-guided” search algorithm (SBPG). The three leading

algorithms are trial-based algorithms. They sample a set of beliefs by exploring forward in each trial. During forward exploration, they decide the next sampled belief via both action-selection strategy and observation-selection strategy. In general, the two strategies are, in fact, the functions of both the lower and upper bound over the optimal value function. To guarantee convergence, these algorithms adopt the same action-selection strategy, which only depends on the upper bound. As a result, the lower bound is ignored in the design of the action-selection strategy. However, the lower bound is very useful in finding a better policy. A direct insight is that the current best policy of these algorithms is obtained according to the lower bound, but not the upper bound.

SBPG originates from this insight. It has two novel features. First, it has a heuristic search strategy based on classification and error-minimization search techniques. Second, it has a greedy, often more efficient action-selection strategy, which mainly depends on the lower bound. Using these techniques, SBPG can greedily focus on sampling a small belief space, which is “promising” in terms of the significance and easiness of finding a better policy. In addition, SBPG can easily be combined with the three leading algorithms. Our experimental results indicate the combined algorithms SBPG+HSVI2, SBPG+FRTDP and SBPG+SARSOP, for most of the studied instances, significantly outperform the original algorithms HSVI2, FRTDP and SARSOP, respectively. These results provide a powerful empirical evidence for our proposed framework’s utility.

Our first contribution is a novel framework for speeding up point-based algorithms via greedy strategies. The second contribution is an implementation of this framework that includes a methodology for constructing a greedy strategy and combining it with the three leading algorithms, and a validation of the framework via empirical results.

2 Background and Related Work

In this section, we go over POMDP model and some point-based algorithms for POMDPs, and briefly describe HSVI2, one of the fastest point-based algorithms.

2.1 POMDP Model

POMDP provides a natural and expressive mathematical model for planning of autonomous robots with hidden state and uncertainty in action effects. A POMDP model can be officially defined by a tuple $\langle S, A, Z, Tr, \Omega, R, \gamma, b_0 \rangle$. In the tuple, S is a set of all possible states, A is a set of all possible actions, Z is a set of observations available to the agent, $Tr(s, a, s') = Pr(s'|s, a) : S \times A \times S \rightarrow [0, 1]$ is a state transition function, $\Omega(a, s', z) = Pr(z|a, s') : A \times S \times Z \rightarrow [0, 1]$ is an observation function, $R(s, a)$ is a reward function, γ ($0 \leq \gamma < 1$) is a discount factor, and b_0 represents the agent’s *initial belief*. A *belief* b is a probability distribution over the set of states S , where $b(s)$ gives the probability that the agent’s state is s . When the agent takes action a at belief b and receives

observation z , it will arrive at a new belief $\tau(b, a, z)$. $b_a^z = \tau(b, a, z)$ is defined using Equation 1:

$$b_a^z(s') = \eta \Omega(a, s', z) \sum_{s \in S} Tr(s, a, s') b(s). \quad (1)$$

Here, η denotes a normalizing constraint. Every POMDP problem has at least an *optimal policy* π^* that maximizes the *expected discounted reward* $V^\pi(b) = E[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)|b, \pi]$, where π denotes a *policy*, and s_t and a_t are the agent's state and action at time t , respectively. We denote Π as the set of all possible policies. The function $V^* = V^{\pi^*}$ is called the *optimal value function*, which is usually represented as a set of vectors. When V^* is available, π^* can be achieved using Equation 2:

$$\pi^*(b) = \arg \max_{a \in A} Q^*(b, a) = \arg \max_{a \in A} \left\{ R(b, a) + \gamma \sum_{z \in Z} Pr(z|b, a) V^*(b_a^z) \right\}. \quad (2)$$

Here, $R(b, a) = \sum_{s \in S} R(s, a) b(s)$, and $Pr(z|b, a)$ means the probability of receiving observation z after taking action a at b . We represent the lower and upper bounds of $V^*(b)$ as $\underline{V}(b)$ and $\bar{V}(b)$, and the lower and upper bounds of $Q^*(b, a)$ as $\underline{Q}(b, a)$ and $\bar{Q}(b, a)$, respectively. Generally speaking, it is intractable to get π^* for large POMDP problems given limited time. Hence, approximate algorithms concentrate on finding an approximate optimal policy π with high expected discounted reward. A policy π is called an ϵ -optimal policy when it satisfies $V^*(b_0) - V^\pi(b_0) \leq \epsilon$. An algorithm is called an ϵ -optimal algorithm if it always returns an ϵ -optimal policy, otherwise it is called a *greedy algorithm*. The strategy used in an ϵ -optimal algorithm and a greedy algorithm are called an ϵ -optimal strategy and a *greedy strategy*, respectively.

For every POMDP problem, beliefs that are reachable from the initial belief b_0 can be represented as an *and/or tree* [4]. We denote it as Δ_{POMDP} . Each node in the tree represents a belief. The root node in Δ_{POMDP} is b_0 . Beliefs that are reachable from b_0 by following some policy π make up an and/or subtree in Δ_{POMDP} [3]. We define the subtree corresponding to π as Δ . Without the risk of confusion, we also use Δ to represent the set of all nodes in the subtree, since a subtree consists of nodes. Note that the notation of Δ is important and will be mentioned several times in this paper. D_Δ is the depth of Δ , and $\Delta(i)$ is the set of nodes at depth i , where $0 \leq i \leq D_\Delta$.

2.2 Point-Based Algorithms

We can sort point-based algorithms into three camps. The first camp focuses on how to sample a set of beliefs B . HSVI2, FRTDP and SARSOP are three of the most successful algorithms in this camp. In addition, other algorithms, such as point-based value iteration (PBVI) [8], Perseus [9] and Forward search value iteration (FSVI) [10] are also well known and widely accepted. The second camp concentrates on finding an optimal order of performing backup operations. Except for the above point-based algorithms, some approximate algorithms which

require low computational cost are used to initialize \underline{V} and \bar{V} , respectively. Among them, the blind policy [11] and the fast informed bound (FIB) [11] have been widely accepted. HSVI2, FRTDP and SARSOP use the blind policy to initialize \underline{V} and use FIB to initialize \bar{V} . The initial bounds \underline{V}_0 and \bar{V}_0 obtained by these two methods are *uniformly improvable* [12], since they satisfy $\max_{a \in A} \{R(b, a) + \gamma \sum_{z \in Z} Pr(z|b, a) \underline{V}_0(b^z_a)\} \geq \underline{V}_0(b)$ and $\max_{a \in A} \{R(b, a) + \gamma \sum_{z \in Z} Pr(z|b, a) \bar{V}_0(b^z_a)\} \leq \bar{V}_0(b)$ for any b , respectively. A more detailed survey of point-based algorithms can be find in [3][11].

2.3 Hsvi2

Because of Hsvi2's simple structure, we describe Hsvi2 briefly (see Algorithm 1). Later, we discuss how to combine SBPG with Hsvi2. In Algorithm 1, d_b represents b 's depth in Δ_{POMDP} , $\text{excess}(b, d_b)$ is specified as $\bar{V}(b) - \underline{V}(b) - \epsilon \gamma^{-d_b}$. The procedure $\text{update}(b)$ updates both $\underline{V}(b)$ and $\bar{V}(b)$ by performing an α -vector backup operation and a Bellman update at b , respectively. Hsvi2 adopts the action-selection strategy that only depends on the upper bound (see line 7).

Algorithm 1. Hsvi2

Function $\pi = \text{Hsvi2}(\epsilon)$	Function $\text{trialRecurse}(b, \epsilon, d_b)$
1: Initialize the bounds \underline{V} and \bar{V} ;	5: if $\bar{V}(b) - \underline{V}(b) \leq \epsilon \gamma^{-d_b}$ then
2: while $\bar{V}(b_0) - \underline{V}(b_0) > \epsilon$	6: return ;
3: $\text{trialRecurse}(b_0, \epsilon, 0)$;	7: $a^* = \arg \max_{a \in A} \bar{Q}(b, a)$;
4: return π corresponding to $\underline{V}(b_0)$;	8: $z^* = \arg \max_{z \in Z} [Pr(z b, a^*) \cdot$ $\text{excess}(\tau(b, a^*, z), d_b + 1)]$;
	9: $\text{trialRecurse}(\tau(b, a^*, z^*), \epsilon, d_b + 1)$;
	10: $\text{update}(b)$;

3 Framework Definition

This section provides a definition of the framework used in this paper (see Framework 1). Trading off exploration and exploitation is one of central issues of reinforcement learning. In the past decades, some simple, ad hoc exploration strategies have been developed in the reinforcement-learning field. Among them, ϵ -greedy exploration, Boltzmann exploration, interval estimation and Rmax have been popular in practice [5]. These exploration methods are viewed as reasonable and computationally tractable approaches. However, greedy exploration methods have been only used to initialize \underline{V} and \bar{V} in the POMDP field. The main reason seems to be that previous POMDP researchers always pursue general useful heuristics to find an ϵ -optimal policy for all common POMDP problems. However, the demand of these general heuristics is harsh and unfair for many very useful heuristics with some shortcomings, such as, local optima. The framework is proposed to address this issue via the combination of an ϵ -optimal strategy and a greedy strategy. Note that the concrete representations of exploration and exploitation in reinforcement learning and our framework are different. However,

they have the common feature in exploring a better action strategy via heuristics which are eschewed in exploitation. In Framework 1, T_O and T_G represent the ϵ -optimal algorithm's running time and the greedy algorithm's running time, respectively. The $condition(T_O, T_G)$ can be defined in different forms. Equation 3 is its simplest form, which is similar to the ϵ -greedy strategy.

$$condition(T_O, T_G) = \frac{T_G}{T_O + T_G} < \epsilon. \quad (3)$$

Framework 1. Framework for Accelerating Point-Based POMDP Algorithms

- 1: Choose an original ϵ -optimal algorithm O ;
 - 2: Construct a greedy algorithm G ;
 - 3: **while** $\bar{V}(b_0) - \underline{V}(b_0) > \epsilon$
 - 4: call a main step of the original ϵ -optimal algorithm O ;
 - 5: **if** $condition(T_G, T_O) == \text{true}$ **then**
 - 6: call the greedy algorithm G ;
 - 7: **return** π corresponding to $\underline{V}(b_0)$;
-

As shown in Framework 1, the invocation of a greedy strategy is not limited to the combined algorithm's initialization. Therefore, the combined algorithm can use the greedy strategy to explore possible shortcuts over its "whole" running time. The advantage of such a framework is that the combined algorithm has more extra opportunities to accelerate the original ϵ -optimal algorithm if the greedy strategy has some wonderful property, such as, very low computational cost, or very useful heuristics.

4 Mathematical Foundation for a Greedy Strategy

We now describe an important theorem for constructing SBPG (see Sect. 5). First, we define a policy as the *current best policy* π_{best} and the *current second best policy* π_{second} if they, respectively, satisfy Equation 4 and Equation 5:

$$\underline{V}^{\pi_{best}}(b_0) \geq \underline{V}^\pi(b_0), \pi \in \Pi, \quad (4)$$

$$\underline{V}^{\pi_{second}}(b_0) \geq \underline{V}^\pi(b_0), \pi \in \Pi - \{\pi_{best}\}. \quad (5)$$

Following π_{best} 's definition, we know $\pi_{best}(b) = \arg \max_{a \in A} \underline{Q}(b, a)$. In fact, it is possible that many policies π_{best} and π_{second} exist. In these situations, we can select a unique π_{best} according to an appropriate (lexicographic) tie-breaking rule. In what follows, we assume π_{best} is unique to simplify the discussion and denote a set of all policies π_{second} that satisfies Equation 5 as Π_{second} . We represent the subtree that corresponds to π_* by Δ_* , where * can be "best", "second", "candidate" or "promising". Theorem 1 shows the relationship between π_{best} and π_{second} . We don't include theorem 1's proof due to space limitations.

Theorem 1. Given \underline{V} is uniformly improvable, there exist at least one π_{second} in Π_{second} and only one node b^* in both Δ_{best} and Δ_{second} such that $\pi_{second}(b^*)$ is in $\arg \max_{a \in A - \{\pi_{best}(b^*)\}} \underline{Q}(b^*, a)$ and $\pi_{second}(b') = \pi_{best}(b')$ for any b' in $\Delta_{second} - \{b^*\}$.

Now, we define a set of policies $\Pi_{candidate}$, which is a useful tool for constructing SBPG. A policy $\pi_{candidate}$ is in the set $\Pi_{candidate}$ if and only if it satisfies that there exists a node b^* in both Δ_{best} and $\Delta_{candidate}$ such that $\pi_{candidate}(b^*) \neq \pi_{best}(b^*)$ and $\pi_{candidate}(b') = \pi_{best}(b')$ for any b' in $\Delta_{candidate} - \{b^*\}$. Let's look at this definition, we can easily conclude that there exists at least one π_{second} in $\Pi_{candidate}$, and the definition of $\pi_{candidate}(b^*)$ seems to be confusing. Here, we first ignore this confusion since the intent becomes clear in Sect. 5.2. All policies $\pi_{candidate}$ in $\Pi_{candidate}$ are different in their definition of what b^* is and what $\pi_{candidate}(b^*)$ is. Since there are $|\Delta_{best}|$ possible b^* and $|A| - 1$ possible actions $\pi_{candidate}(b^*)$ at every b^* , $\Delta_{candidate}$ has $|\Delta_{best}|(|A| - 1)$ policies. All policies $\pi_{candidate}$ in $\Pi_{candidate}$ have a remarkable property: if $\underline{Q}(b^*, \pi_{candidate}(b^*))$ is greater than $\underline{Q}(b^*, \pi_{best}(b^*))$, then a better policy will be found. Note that SBPG's objective is to speed up the process of finding a better policy, it hence focuses on re-evaluating a set of “promising” policies' reward in $\Pi_{candidate}$.

As we see, Δ_{best} is a tree with infinite depth, and therefore, Δ_{best} may consist of an infinite number of nodes. To focus on the key part for $V^{\pi_{best}}(b_0)$, we prune the vast majority of nodes in the tree by adding a constraint condition: $\text{excess}(b, d_b) > 0$ for all b in Δ_{best} . Here, the definition of $\text{excess}(b, d_b)$ is different, which depends on the algorithm that SBPG is combined with. When SBPG is combined with HSVI2 or SARSOP, it is defined as $(\bar{V}(b) - \underline{V}(b)) - \epsilon\gamma^{-d_b}$, whereas it is $(\bar{V}(b) - \underline{V}(b)) - \epsilon/2$ when SBPG is combined with FRTDP. In the next section, we use this definition as Δ_{best} , but not the prior one.

5 Framework Implementation via a Greedy Strategy

This section provides an algorithmic implementation of Framework 1. Sect. 5.1–5.3 describe how to construct a greedy algorithm SBPG using some common ignored heuristics in HSVI2, FRTDP and SARSOP. Sect. 5.4 describes how to combine it with the three leading algorithms. Here, we first briefly describe the exploration mechanism in SBPG.

In a running race, the most “promising” player for surpassing the currently No.1 ranked player is the current No.2 player. SBPG's exploration mechanism stems from this principle. In HSVI2, FRTDP and SARSOP, policy's reward evaluation ($\underline{V}^\pi(b_0)$) by the lower bound is not exact, and sometimes very rough. Therefore, π_{second} 's reward re-evaluation is a good way of finding a better policy. The main difficulty is to locate b^* 's position in Δ_{second} (see Theorem 1). Fortunately, it is not a big obstacle since π_{second} 's property shown in Theorem 1 offers a guidance for selecting some “promising” policies from $\Pi_{candidate}$. Following the definition of $\Pi_{candidate}$, whether a policy in $\Pi_{candidate}$ is “promising” depends on its b^* and the definition of $\pi(b^*)$. Sect. 5.1 provides a heuristic method via classification and error-minimization search to find a set of “promising” nodes

b^* . In Sect. 5.2, we use the second best action selection in defining $\pi(b^*)$. Sect. 5.3 introduces a sampling strategy that re-evaluates “promising” policies’ reward.

5.1 Heuristic Search via Classification and Error-Minimization

A “promising” node plays a significant and easy role in finding a better policy. SBPG selects the “promising” nodes from Δ_{best} via the following strategy.

Classification. Each set $\Delta_{best}(i)$ has a weight to measure its importance in finding a better policy. For a node $\tau(b, a, z)$ at the i^{th} depth of Δ_{best} , let its lower bound improve δ , then the backup operation at its parent node b will bring an improvement of at least $\gamma\delta Pr(z|b, a)$ at b , further, the backup operations along the path from its parent node to the root node b_0 will bring $\underline{V}(b_0)$ ’s improvement of at least $\gamma^i\delta Pr(b|b_0, \pi_{best})$, where $Pr(b|b_0, \pi_{best})$ represents the probability of arriving at b from b_0 by following the policy π_{best} . Let the lower bound of all nodes in $\Delta_{best}(i)$ improve δ , then the backup operations from every node’s parent node to b_0 will improve the lower bound at least $\gamma^i\delta \sum_{b \in \Delta_{best}(i)} Pr(b|b_0, \pi_{best})$ at b_0 . Therefore, SBPG uses $\gamma^i \sum_{b \in \Delta_{best}(i)} Pr(b|b_0, \pi_{best})$ as $weight(\Delta_{best}(i))$.

According to the above definition of $weight$, we can conclude that $weight(\Delta_{best}(0)) = 1$, and $0 < weight(\Delta_{best}(i)) \leq 1$, for $1 \leq i \leq D_{\Delta_{best}}$. When SBPG is called, it decides whether to select a “promising” node from each $\Delta_{best}(i)$ using $weight(\Delta_{best}(i))$ as its selection probability. To avoid randomness, SBPG uses an equivalent deterministic algorithm to make decision.

Error-Minimization Search. When SBPG has decided to choose a node from $\Delta_{best}(i)$, the error-minimization search strategy will be useful to select a “promising” node from $\Delta_{best}(i)$. For each node in $\Delta_{best}(i)$, SBPG computes its heuristic value H using Equation 6:

$$H(b) = \begin{cases} Pr(b|b_0, \pi_{best})(\bar{V}(b) - \underline{V}(b)) & \text{if } a_s^b \text{ exists} \\ 0 & \text{otherwise.} \end{cases} \quad (6)$$

Here, $Pr(b|b_0, \pi_{best})$ represents the significance that $\underline{V}(b)$ ’s improvement will bring into $\underline{V}(b_0)$ ’s, and $\bar{V}(b) - \underline{V}(b)$, to some extent, represents the easiness of improving $\underline{V}(b)$. Therefore, $H(b)$ is a trade-off between the significance and easiness. By the error-minimization search strategy, SBPG chooses the node with the highest non-zero heuristic value from $\Delta_{best}(i)$. The strategy is inspired by a similar search strategy in an efficient online POMDP algorithm [13].

5.2 Second Best Action Selection

According to Theorem 1, we should define $\pi(b^*) = \arg \max_{a \in A - \{\pi_{best}(b^*)\}} \underline{Q}(b^*, a)$. However, this action selection sometimes guides search towards a suboptimal action branch. We address it by a refined action selection a_s^b in Equation 7:

$$a_s^b = \arg \max_{a \in A_s^b - \{\pi_{best}(b)\}} \underline{Q}(b, a). \quad (7)$$

Here, A_s^b represents $\{a \in A | \bar{Q}(b, a) > \underline{Q}(b, \pi_{best}(b))\}$. We call a_s^{b*} as “ b ’s second best action”. The second best action selection uses the upper bound \bar{V} to clear away suboptimal action branches, just like the α - β pruning technique in game theory. Therefore, it is a good way of leveraging off the lower and upper bounds.

5.3 Second Best Policy-Guided Sampling and Updates

According to Sect. 5.1 and Sect. 5.2, we can obtain a set of “promising” policies $\Pi_{promising}$ from $\Pi_{candidate}$, whose elements are different in b^* and $\pi(b^*)$. Now, we need to re-evaluate each $\pi_{promising}$ in $\Pi_{promising}$. Algorithm 2 first re-evaluates the subtree rooted at $Q^{\pi_{promising}}(b^*, a_s^{b*})$ (see line 1-2 in Algorithm 2), in the hope that $Q^{\pi_{promising}}(b^*, a_s^{b*})$ is greater than $Q^{\pi_{best}}(b^*, \pi_{best}(b^*))$. If so, it updates each node along the path from b_0 to b^* in reversed order. Thus a better policy will be found. Here, D_{max} represents the maximal sampling depth from b^* . We increase D_{max} if SBPG hasn’t found a better policy during a trial (see line 3-4 in Algorithm 3). Algorithm 3 shows a sketch of SBPG. Note that SBPG defines its action selection mainly according to the lower bound (see line 1-2 in Algorithm 2 and the definition of $\pi_{candidate}$ in Sect. 4), whereas HSVI2, FRTDP and SARSOP’s action selection only depends on the upper bound. This is the deep reason that SBPG can be well combined with the three leading algorithms.

Algorithm 2. samplingAndUpdates

```
Procedure samplingAndUpdates( $b^*, a_s^{b*}$ )
1: for  $i \leftarrow 0 : D_{max}$  do
2:   update( $b$ ), for all  $b$  in  $\{\mathbb{b} \in \Delta_{promising}(d_b + D_{max} - i) | excess(\mathbb{b}, d_b) > 0\}$ ;
3: if  $\underline{V}(b^*)$  has been improved then
4:   update each node along the path from  $b_0$  to  $b^*$  in reversed order;
```

Algorithm 3. SBPG

```
Procedure SBPG( $\epsilon$ )
1: select a set of “promising” policies  $\Pi_{promising}$  from  $\Delta_{best}$ ;
2: samplingAndUpdates( $b^*, a_s^{b*}$ ), for all  $\pi_{promising}$  in  $\Pi_{promising}$ ;
   //Each  $\pi_{promising}$  has unique  $b^*$  and  $a_s^{b*}$ .
3: if a better policy hasn’t been found then
4:    $D_{max}++$ ;
```

5.4 Combination of SBPG and Leading Algorithms

Algorithm 4 presents the SBPG+HSVII2 algorithm. In a similar way, the combined algorithms SBPG+FRTDP and SBPG+SARSOP can also be obtained. The three combined algorithms can easily be modified into anytime algorithms. In this paper, we define simply $condition(T_{SBPG}, T_*) = \frac{T_{SBPG}}{T_{SBPG} + T_*} \leq \frac{1}{2}$, where * represents one of HSVI2, FRTDP and SARSOP.

Algorithm 4. SBPG+HSVII2

Function $\pi = \text{SBPG+HSVII2}(\epsilon)$
1: Initialize the bounds \underline{V} and \bar{V} ;
2: $D_{max} = 1$;
3: **while** $\bar{V}(b_0) - \underline{V}(b_0) > \epsilon$
4: trialRecurse($b_0, \epsilon, 0$);
5: **if** condition(T_{SBPG}, T_{HSVII2}) == true **then**
6: SBPG(ϵ);
7: **return** π corresponding to $\underline{V}(b_0)$;

6 Empirical Evaluations

In this section, we present empirical results of our combined algorithms and compare them with HSVI2, FRTDP and SARSOP, respectively.

6.1 Experimental Setup

We implemented both SBPG+HSVII2 and SBPG+FRTDP based on zmdp-v1.1.7, in which HSVI2 and FRTDP have been included. For SBPG+SARSOP and SARSOP, we use appl-v0.3. The experimental platform is AMD Athlon(tm) 64 X2 Dual Core Processor 3600+ 2.00GHz, 2GB memory. The operation system is Ubuntu Linux 9.04. Table 1 lists all benchmark problems we have tested.

Table 1. Benchmark problems

Problem	$ S $	$ A $	$ Z $	Problem	$ S $	$ A $	$ Z $
TagAvoid	870	5	30	LifeSurvey1	7,001	7	28
RockSample_5_5	801	10	2	RockSample_5_7	3,201	12	2
RockSample_7_8	12,545	13	2	RockSample_10_10	102,401	15	2

6.2 Results

We compare the performance of the combined algorithms with the original algorithms in terms of the reward and the time using via Table 2. Now, we describe how an algorithm’s reward and time are obtained on a benchmark problem. Firstly, we ran every algorithm for a maximum of half an hour. Every algorithm returned a policy. We evaluated the expected discounted rewards of these policies using sufficiently large number (more than 10,000) of simulation runs, and then selected the minimal and maximal reward among them. We empirically set a threshold as $(\text{minimal reward} + \text{maximal reward}) / 2$ for the benchmark problem. Secondly, we ran each algorithm again and evaluated its policy for every “evaluated time interval”. The evaluated time interval is dependent on the POMDP problem’s size. The term of “time” records the first time that an

algorithm obtains a policy whose reward is greater than the threshold. And the term of “reward” is the corresponding policy’s reward and the reward’s 95% confidence interval.

Table 2. Performance comparison on benchmarks (e.t.i. = evaluated time interval)

Method	Reward	Time (s)	Method	Reward	Time (s)
TagAvoid e.t.i. = 1s			LifeSurvey1 e.t.i. = 200s		
SBPG+HSV12	-6.09±0.09	13	SBPG+HSV12	93.68±0.14	1200
HSV12	-6.05±0.09	591	HSV12	92.22±0.14	1800
SBPG+FRTDP	-6.04±0.10	6	SBPG+FRTDP	94.39±0.15	200
FRTDP	-6.09±0.12	46	FRTDP	94.30±0.15	600
SBPG+SARSOP	-6.02±0.10	1.5	SBPG+SARSOP	94.42±0.15	200
SARSOP	-6.06±0.12	6	SARSOP	94.40±0.15	800
RockSample_5_5 e.t.i. = 0.5s			RockSample_5_7 e.t.i. = 10s		
SBPG+HSV12	19.23±0.07	1.0	SBPG+HSV12	24.62±0.06	60
HSV12	19.23±0.07	19.5	HSV12	24.61±0.07	460
SBPG+FRTDP	19.23±0.06	1.0	SBPG+FRTDP	24.60±0.05	90
FRTDP	19.23±0.06	3.5	FRTDP	24.62±0.06	480
SBPG+SARSOP	19.24±0.10	1.5	SBPG+SARSOP	24.61±0.10	40
SARSOP	19.23±0.10	4.0	SARSOP	24.61±0.12	360
RockSample_7_8 e.t.i. = 200s			RockSample_10_10 e.t.i. = 200s		
SBPG+HSV12	21.51±0.13	600	SBPG+HSV12	20.39±0.10	1800
HSV12	21.37±0.12	1800	HSV12	19.47±0.09	1800
SBPG+FRTDP	21.51±0.11	800	SBPG+FRTDP	20.86±0.09	1200
FRTDP	21.49±0.10	800	FRTDP	20.11±0.12	1800
SBPG+SARSOP	21.49±0.11	800	SBPG+SARSOP	20.49±0.11	200
SARSOP	21.28±0.11	1800	SARSOP	20.49±0.11	600

7 Conclusions and Future Work

This paper presents a novel framework and an implementation of this framework for accelerating point-based POMDP algorithms. This implementation mainly constructs a greedy strategy SBPG via some common ignored heuristics in three leading point-based POMDP algorithms. It is important to note that the heuristics used in constructing a greedy strategy must be very useful, and the method of using them is key to our framework’s performance. In our implementation, the lower bound over the optimal value function not used in the three leading algorithms’ action-selection is very useful. The second best policy guided exploration mechanism in SBPG provides a promising way to accelerate these algorithms. As a future work, we will examine constructing more greedy strategies to speed up current point-based POMDP algorithms, and provide further empirical evidences for our proposed framework’s efficiency.

Acknowledgments. Thanks to Feng Wu, Michael L. Littman, Trey Smith, Stéphane Ross, and David Hsu for ideas and discussions.

References

1. Hoey, J., von Bertoldi, A., Poupart, P., Mihailidis, A.: Assisting Persons with Dementia during Handwashing Using a Partially Observable Markov Decision Process. In: Proceedings of International Conference on Vision Systems (2007)
2. Hsu, D., Lee, W.S., Rong, N.: A Point-Based POMDP Planner for Target Tracking. In: Proceedings of IEEE Conference on Robotics and Automation, pp. 2644–2650 (2008)
3. Smith, T.: Probabilistic Planning for Robotic Exploration. Ph.D. Dissertation, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA (2007)
4. Kurniawati, H., Hsu, D., Lee, W.S.: SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. In: Proceedings of Robotics: Science and Systems (2008)
5. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research* 4, 237–285 (1996)
6. Smith, T., Simmons, R.: Point-Based POMDP Algorithms: Improved Analysis and Implementation. In: Proceedings of Uncertainty in Artificial Intelligence (2005)
7. Smith, T., Simmons, R.: Focused Real-Time Dynamic Programming for MDPs: Squeezing More Out of A Heuristic. In: Proceedings of the Twenty-First Conference on Artificial Intelligence, pp. 1227–1232 (2006)
8. Pineau, J., Gordon, G., Thrun, S.: Point-Based Value Iteration: An Anytime Algorithm for POMDPs. In: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, pp. 1025–1032 (2003)
9. Spaan, M.T.J., Vlassis, N.: Perseus: Randomized Point-Based Value Iteration for POMDPs. *Journal of Artificial Intelligence Research* 24, 195–220 (2005)
10. Shani, G., Brafman, R.I., Shimony, S.E.: Forward Search Value Iteration for POMDPs. In: Proceedings of Twentieth International Joint Conference on Artificial Intelligence, pp. 2619–2624 (2007)
11. Hauskrecht, M.: Value-Function Approximations for Partially Observable Markov Decision Processes. *Journal of Artificial Intelligence Research* 13, 33–94 (2000)
12. Zhang, N.L., Zhang, W.: Speeding Up the Convergence of Value Iteration in Partially Observable Markov Decision Processes. *Journal of Artificial Intelligence Research* 14, 29–51 (2001)
13. Ross, S., Pineau, J., Paquet, S., Chaib-Draa, B.: Online Planning Algorithms for POMDPs. *Journal of Artificial Intelligence Research* 32(1), 663–704 (2008)

Author Index

- Ando, Noriaki 168, 180, 192
Andrist, Sean 473
Ansell, Matthew 26
Asfour, Tamim 109

Bachiller, Pilar 251
Bachmann, Ferry 121
Balakirsky, Stephen 336
Bardella, Andrea 436
Beck, Daniel 300
Berns, Karsten 156
Bertram, Torsten 288
Biggs, Geoffrey 168, 180, 192, 350
Billington, David 204
Birk, Andreas 374
Bohg, Jeannette 109
Brugali, Davide 324
Bruyninckx, Herman 15
Burkhard, Hans-Dieter 533
Bustos, Pablo 251, 263
Buys, Koen 15

Calderita, Luis 251
Chen, Ian Yen-Hung 350
Chen, Xiaoping 545
Cho, Joomyun 217
Christensen, Henrik I. 336
Cintas, Ramón 251
Codd-Downey, Robert 26
Corominas Murtra, Andreu 461
Costantini, Diego 133
Cummins, Chris 75

DallaLibera, Fabio 362
Danieletto, Matteo 436
De Laet, Tinne 15
Diankov, Rosen 109
Digor, Elena 374
Dillmann, Rüdiger 109
Drews, Sebastian 387
Drumwright, Evan 38
Durst, Phillip J. 75

Erlandson, Alexander J. 473
Estivill-Castro, Vladimir 204

Ferrein, Alexander 229, 300
Frank, Sebastian 399
Freese, Marc 51
Friedmann, Martin 63
Fritzson, Peter 5

Gates, Burhman 75
Gerkey, Brian P. 1
Gini, Giuseppina 449
Gini, Maria 473, 485
Goodin, Chris 75

Hara, Isao 275
Haschke, Robert 144
Haumann, Dominik 133, 399
Heß, Robin 87
Hexel, René 204
Hoffmann, Frank 288
Hsu, John 38

Ikemoto, Shuhei 362
Ishiguro, Hiroshi 362

Jasiobedzki, Piotr 26
Jenkin, Michael 26

Kanda, Takayuki 2
Khatib, Oussama 3
Khelif, Abdelmajid 133
Kim, Hyun 217
Knoll, Alois 324
Koenig, Nathan 38
Kolhe, Pushkar 336
Kostić, Dragan 521
Kotoku, Tetsuo 168, 180, 192, 350
Kotthäuser, Tobias 97
Kramer, Tom 336
Kuffner, James 109
Kułakowski, Konrad 241
Kunz, Jürgen 411
Kurowski, Stefan 497

Lakemeyer, Gerhard 300
Lange, Sven 387
Lens, Thomas 411, 497

- León, Beatriz 109
 Listmann, Kim 399
 MacDonald, Bruce 350
 Magid, Evgeni 423
 Malenfant, Jacques 312
 Manso, Luis 251, 263
 Martínez, Jesús 263
 Matsuhira, Nobuto 51
 Matsusaka, Yosuke 275
 Matyasik, Piotr 241
 Mellmann, Heinrich 533
 Menegatti, Emanuele 362, 436
 Mertsching, Bärbel 97
 Mina, Marco 436
 Minato, Takashi 362
 Mirats Tur, Josep M. 461
 Mogre, Parag S. 133
 Mohammed, Abdul A. 473
 Moisio, Sami 109
 Morales, Antonio 109
 Moro, Federico 449
 Nanjanath, Maitreyi 473
 Narayanan, Krishna K. 288
 Ng, Ho-Kong 26
 Niemueller, Tim 300
 Nijmeijer, Henk 521
 Nüchter, Andreas 374
 Núñez, Pedro 251
 Ozaki, Fumio 51
 Pagello, Enrico 362
 Pakanati, Anuraag 485
 Pasemann, Frank 121
 Petersen, Karen 63, 133
 Posada, Luis Felipe 288
 Pretto, Alberto 436
 Priddy, Jody 75
 Proctor, Fred 336
 Protzel, Peter 387
 Przybylski, Markus 109
 Puche, Gustavo 109
 Radkhah, Katayon 497
 Ragipindi, Aravind 473
 Reinl, Christian 133
 Rempis, Christian 121
 Ritter, Helge 144
 Rock, Andrew 204
 Rodic, Aleksandar 449
 Rogovchenko, Olena 312
 Romero-Garcés, Adrián 263
 Sachidananda, Vinay 133
 Sanfeliu, Alberto 461
 Schilling, Klaus 87
 Schlegel, Christian 324
 Schmidt, Daniel 156
 Sharma, Ankur S. 473
 Shell, Dylan 38
 Singh, Surya 51
 Smits, Ruben 15
 Sohn, Joochan 217
 Steck, Andreas 324
 Steinbauer, Gerald 229
 Thomas, Verena 121
 Trulls, Eduard 461
 Tsubouchi, Takashi 423
 Tudorie, Claudia Raluca 509
 Ulbrich, Stefan 109
 van Zutven, Pieter 521
 von Stryk, Oskar 63, 411, 497
 Weitnauer, Erik 144
 Wettach, Jens 156
 Willert, Volker 399
 Wünsche, Burkhard 350
 Xu, Yuan 533
 Zanella, Andrea 436
 Zanuttigh, Pietro 436
 Zefran, Milos 449
 Zhang, Zongzhang 545