Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it-bots

Master Thesis Report

Task Planning, Execution and Monitoring for Mobile Manipulators in Industrial Domains

Author:
Oscar Lima Carrion
oscar.lima@smail.inf.h-brs.de
Matrikel Nr: 9021273

Advised by:
Prof. Dr. Paul G. Plöger
paul.ploeger@h-brs.de
Prof. Dr-Ing. Gerhard Kraetzschmar
gerhard.kraetzschmar@h-brs.de
M.Sc. Iman Awaad
iman.awaad@h-brs.de

Bonn-Rhein-Sieg University of Applied Sciences
Department of Computer Science

April 2016

Declaration of Authorship

I, Oscar Lima, declare that this Master thesis report titled, "Task Planning, Execution and Monitoring for Mobile Manipulators in Industrial Domains" and the work presented in it are my own. I confirm that:

- This work was done while in candidature for a master degree at this University.
- Where any part of this report has previously been submitted for a degree or any other qualification at this University or any other institution.
- Portions of this work were presented in a conference paper with title "Task Planning with Costs for Industrial Robotics Domains" which is under review for publication in RoboCup International Symposium 2016 (July 4, 2016) at the time of thesis submission.
- Where I have consulted the published work of others, it is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this report is entirely my own work.
- I have acknowledged all main sources of help.
- Where the report is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Bonn-Rhein-Sieg University of Applied Sciences

Abstract

Department of Computer Science

Master Thesis Report

Task Planning, Execution and Monitoring for Mobile Manipulators in Industrial Domains

by Oscar Lima Carrion

Decision making is a necessary skill for autonomous mobile manipulators working in industrial environments. Task planning is a well-established field with a large research community that continues to produce new heuristics, tools and algorithms that enable agents to produce plans that achieve their goals.

In this work we demonstrate the applicability of satisficing task planning with action costs by integrating a state-of-the-art planner, the Mercury planner, into the KUKA youBot mobile manipulator and using it to solve basic transportation and insertion tasks. In contrast to optimal planners which minimize total action costs in a plan, satisficing planners minimize plan generation time, yielding sub-optimal solutions but in less time compared to optimal planners. The planner uses a delete-list relaxation heuristic to quickly prune the search space and generate satisfactory solutions.

The developed planning framework is modular, re-usable and well documented. It brings together two major standards in robotics and task planning: ROS and PDDL, similar to ROSPlan. Unlike ROSPlan, this work is able to plan with cost information. Moreover, it is more modular, enabling the community to use the various components at their own discretion. Finally, while ROSPlan allows only one re-planning strategy, our framework enables users to quickly implement their own strategies.

The resulting system demonstrates that the agent's behavior is optimized, it is able to flexibly handle unexpected situations, and it can robustly handle failures by re-planning when needed. It is also easier to maintain and extend. The work presented here also highlights the benefits of conducting a domain analysis to gain the maximum benefit from the use of a given planner and domain.

Acknowledgements

I would like to express my gratitude to my supervisors Prof. Dr. Paul Plöger and Prof. Dr-Ing. Gerhard Kraetzschmar for the useful comments, remarks and engagement through the learning process of this master thesis.

I would like to specially thank Iman Awaad for her guidance in the topic as well for the support on the way.

Many thanks to the University Robocup team, who have willingly shared their precious time during the process of implementation and testing.

Many thanks for proof reading of this work to Tan Chun Kwang and Santosh Thoduka.

Thanks to Emo Robot project, who provided me with student job for almost two years during the Master course, and from which I have learned plenty.

Many thanks to Distribuidora Metalica S.A. de C.V. (Mexico company) for the provided economic support.

Special thanks to my father, Fernando Lima Gonzalez which encouraged me to follow my dreams and supported me during my studies.

Finally I would like to thank my loved ones: Mother, Family and Friends, who have supported me throughout entire process, both by keeping me harmonious and helping me putting pieces together.

Contents

List of Figures

List of Tables

Abbreviations

ADL	**A**ction **D**escription **L**anguage
AI	**A**rtificial **I**ntelligence
BMT	**B**asic **M**anipulation **T**est
BNT	**B**asic **N**avigation **T**est
BTT	**B**asic **T**ransportation **T**est
CBT	**C**onveyor **B**elt **T**est
CLIPS	**C** **L**anguage **I**ntegrated **P**roduction **S**ystem
CPU	**C**entral **P**rocessing **U**nit
CTT	**C**ompetitive **T**ransportation **T**est
DWA	**D**ynamic **W**indow **A**pproach
GUI	**G**raphic **U**ser **I**nterface
IBM	**I**nternational **B**usiness **M**achines (company)
ICAPS	**I**nternational **C**onference on **A**utomated **P**lanning and **S**cheduling
IPC	**I**nternational **P**lanning **C**ompetition
IK	**I**nverse **K**inematics
KB	**K**nowledge **B**ase
KDL	**K**inematics and **D**ynamics **L**ibrary
LED	**L**ight **E**mitting **D**iode
LUH bots	**L**eibniz **U**niversity **H**annover bots
NASA	**N**ational **A**eronautics and **S**pace **A**dministration
OMPL	**O**pen **M**otion **P**lanning **L**ibrary
PDDL	**P**lanning **D**omain **D**efinition **L**anguage
PTT	**P**recision **P**lacement **T**est
ROS	**R**obot **O**perating **S**ystem
ROSPlan	**R**obot **O**perating **S**ystem **P**lanning
SMACH	**S**tate **MACH**ine
STRIPS	**S**tanford **R**esearch **I**nstitute **P**roblem **S**olver
SVM	**S**upport **V**ector **M**achine
UCPOP	so**U**nd and **C**omplete **P**artial **O**rder **P**lanner
URDF	**U**niversal **R**obot **D**escription **F**ormat

VAL Plan **VAL**idation System

Chapter 1

Introduction

1.1 Motivation

Although robot manipulators have been used in industry since 1961, the use of autonomous mobile manipulator applications in these days are limited, yet mobile manipulators are considered as essential components of the Factory of the Future [1].

Kiva robot [2] is one example of a robot currently working in industry. Traditionally, objects are transported inside distribution centers using conveyor belt systems or by human operated machines (for example forklifts). With the Kiva systems approach, items are stored on portable storage units which are transported by robots.

At Amazon warehouses, people work together with robots on a daily basis. A swarm of robots manipulate shelves inside the warehouse and brings them closer to human operators that fill the racks with orders. A button is pressed once the human task is finished and the robot takes away the full shelve, to be replaced by a empty one. The complicated logistic problem is tackled by using simple solutions that are highly functional. A wide range of talent is needed to accomplish such challenges, from mechanical engineers, software development to production experts, all of them sharing one common goal, to increase the productivity [2].

"In March 2012, Amazon.com acquired Kiva Systems for 775 million USD, the second largest acquisition in its history"[1].

The system works with software agents that run on the robots computer and on a computer at the picking stations. The agents exchange information but act independently, each one trying to optimize its own task. Programmers use heuristic methods such as

[1] https://en.wikipedia.org/wiki/Amazon_Robotics

greedy search that make non-optimal solutions, but good enough for the robot to get the job done [2].

In fact, during the development of Kiva, engineers were looking for projects involving mobile robots and *"became fascinated with videos of RoboCup, the international robotic soccer championship"* [2].

RoboCup is an international competition well known in research as a means to promote robotics and artificial intelligence research.

The RoboCup @Work league *"aims to foster research and development that enables use of innovative mobile robots equipped with advanced manipulators for current and future industrial applications, where robots cooperate with human workers for complex tasks ranging from manufacturing, automation, and parts handling up to general logistics"* [2].

This work uses the RoboCup @Work competition as a use case to test our task integration approach and re-factors already existing code from our University @Work team, the b-it-bots[3].

1.2 Problem statement

Although our robot can perform many individual actions, all the logic for complex behavior is currently performed via finite state machines (FSM).

FSM are suitable to model a large number of problems, however, our skills as humans are limited to accurately program large and complex FSM.

Therefore there is there is a need to consider other decision making approaches to enable intelligent behavior on the robot to autonomously perform decision making.

The problem tackled by this work is the integration of a state of the art planner into a real robotic system.

To accomplish that several sub-problems need to be solved first:

A planning domain model derived from a industrial environment needs to be designed. We propose, as a start, to model the domain based on transportation and insertion tasks, but keep it flexible enough to be easily extensible to other industrial related tasks such as stack, paint, etc.

[2] `http://www.robocupatwork.org/`
[3] `https://mas-group.inf.h-brs.de/`

One of the challenges that this work faces is the selection or design of a fault tolerant planning framework that can make use of an existing planner, making a bridge between the robot skills and the generated plan. Execution and monitoring components need to be developed for this purpose.

1.3 Structure of this work

The remainder of the report is structured as follows:

Chapter 2, state-of-the-art or related work starts describing the international planning competition (IPC), then the last IPC 2014 different tracks are presented. In particular this work is interested and uses information coming from deterministic IPC track, its sub-tracks are also described in detail. Particularly interesting for this work is the sequential satisficing category, so it is further explained. Next the official input language for IPC planners is described: Planning Domain Definition Language (PDDL), how this language was formed and the different PDDL versions among with the evolution of the language is covered. Mercury planner is presented next as a satisficing planner, and its inner working is briefly explained. The next section is dedicated to the explanation of the platform on which Mercury planner is implemented, the Fast Downward planning system. We then describe an existing planning architecture for robotics: ROSPlan, the methods implemented there and the provided tools. Next we explore the approach followed by other teams in the RoboCup @Work league and in particular the one used by LUH bots: graph based search planning. The next section explores the approach followed by RoboCup Logistics League (RCLL) and in particular the CaroLogistics team from Aachen, Germany. The evolution of the competition is described and the middleware used in their robots: Fawkes. Afterwards the behavior engine is presented, state machines and their ruled based approach written in NASA CLIPS. Afterwards, similarities between ROS and Fawkes are mentioned, this last one (Fawkes) widely used in RoboCup competitions, for instance in soccer middle size and @Home league. The chapter ends by briefly explaining some of the tools used by NASA for space exploration and the planning techniques that they deploy for some of their space missions.

Chapter 3 describes the current skills of our robot as a background needed to understand the details of the interaction between planning and a existing robot architecture based on finite state machines (FSM). It starts by describing briefly the event based prefered design pattern, then describes the navigation system, global planner, local planner, base alignment component and recovery behavior. Next the perception functionality is presented, with just the information relevant from the planning point of view. This refers to the fact that the perception algorithm currently used is able to detect many objects

at the same time, so the planning domain needs to adapt and describe this feature. The next section describes the basic manipulation components used by the robot to perform actions such as pick objects or stage in the rear robot platform. The chapter ends by describing the previously existing basic transportation test (BTT) state machine, as the starting point of this work which is later refactored into small re-usable and easy to maintain state machines.

Chapter 4 first describes the motivation behind using task planning to approach the task integration problem addressed in this thesis, then an explanation is provided about the performed search among IPC planners and in particular the decision to select a planner from the sequential satisfactory IPC category. Then the rationale behind the selection of the Mercury planner is explained and discuss the approach followed to integrate the planner into the system. Afterwards we comment on the advantages and limitations of ROSPlan (planning framework) and why it was not used as planning framework but instead we decided to develop our own solution. Then a description of the planning components is presented along with the proposed architecture that integrates them and finally we describe the planning coordinator algorithm (state machine) that integrates the planning functionality.

Chapter 5 aims to test the planning framework previously described. We start by explaining the experimental setup, such as the environment and the computational resources that were utilized, etc. Next the parameters used to configure the planner are exposed and the navigations costs proposed to be used by the planner. The results from a 26 hour experiment are presented, they show scalability of the planner, which tells how well does it respond to a task that is increasing gradually in complexity. The chapter ends by testing different cost values on the planner to get the optimal cost range and as a result of the experiments some recommendations on how to select the costs are issued.

Chapter 6 starts with the lessons learned, followed by the contributions of this work and finishes with the improvements that could be done to the planning framework as ideas for future work.

Chapter 2

Related work

The field of artificial intelligence has worked hard in the last 50 years and currently there are many algorithms and available techniques that can provide solutions for robotics in a efficient and logical way. These AI techniques include Turing machines, search based algorithms, task planning techniques, ruled based agents, finite state machines etc. The focus of this work is on task planning, therefore we explore more about this field than other approaches.

2.1 International planning competition

Nearly every two years, the International Planning Competition (IPC) is hosted at the International Conference on Automated Planning and Scheduling (ICAPS). The IPC is widely considered to be a milestone in the planning community [3]. In this competition, researchers are able to present and compare their algorithms with others. This allows researchers to discuss challenges and exchange source codes. Additionally, this competition helps the planning community measure the overall progress of the field and establish a standardized collection of planning problems as a baseline [4].

The first edition took place in 1998 in Yale University, USA with five contestants on two tracks (STRIPS and ADL). The competition goals were to generate excitement in the planning community and to measure top-quality planning algorithms[1]. The competition was close and failed to find a clear winner. There is slight difference between ADL and STRIPS; ADL is considered an improved version of STRIPS, it allows context-dependent action effects and quantified preconditions.

[1] `http://ipc98.icaps-conference.org/`

2.1.1 IPC 2014

The latest IPC (Eighth International Planning Competition) 2014 was held in Portsmouth, USA with the following tracks:

Deterministic track - Planners are given all information needed to create a solution and no randomness is involved. According to the call for participation[2] *"The objectives of the competition are to provide a forum for empirical comparison of planning systems"*

Learning track - Is designed to perform a comparative evaluation between learning and non-learning planners[3]. Three awards are given in the learning track: overall, basic solver, and best learner.

Probabilistic Planning Tracks - According to lecture slides of the research group foundations of artificial intelligence in Freiburg University *"Probabilistic planning is an extension of nondeterministic planning with information on the probabilities of nondeterministic events"*[4]. Planners are provided with probability information about a particular domain and are asked to produce solutions based on probabilistic state transition functions.

2.1.2 Deterministic track of IPC 2014

Considered to be the biggest part of IPC, the deterministic part was host for 66 participants coming from different regions (such as Australia, Canada, Czech Republic, Finland, France, Germany, Iran, Israel, New Zealand, Spain, Switzerland, United Kingdom, Venezuela, USA) grouped in 31 different teams competed with a total of 67 planners[5], some of them were variations of the same planner.

IPC 2014 Deterministic part announced 4 tracks but at the end only 3 took place:

Sequential - Objective is to minimize action cost.

Agile - Objective is to minimize CPU time.

Temporal - Objective is to minimize total time (makespan or schedule length).

The "sequential" track includes categories like "satisfactory", "optimal" and "multi-core". Figure 2.1 below illustrates all the IPC 2014 tracks.

[2] `https://helios.hud.ac.uk/scommv/IPC-14/cfp.txt`
[3] `http://www.cs.colostate.edu/~ipc2014/`
[4] `http://gki.informatik.uni-freiburg.de/teaching/ss05/aip/s05.pdf`
[5] `https://helios.hud.ac.uk/scommv/IPC-14/repository/booklet2014.pdf`

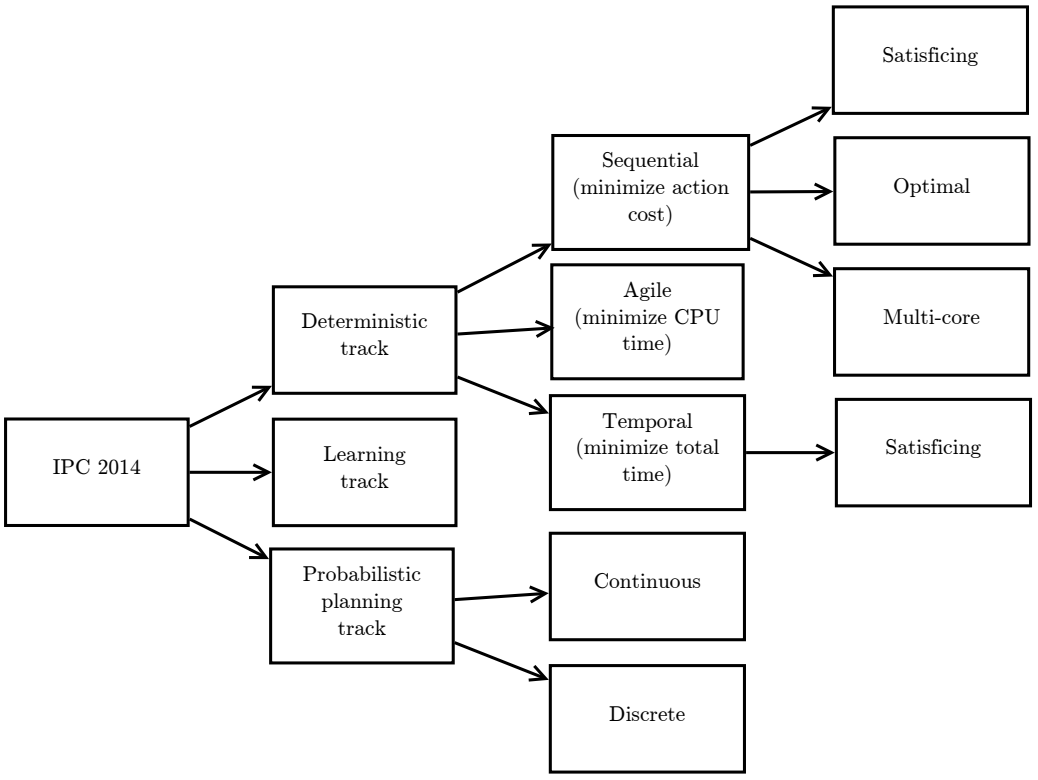


FIGURE 2.1: IPC 2014 tracks.

Each category aims to test different planning aspects, for example the objective of deterministic agile is to minimize CPU time while deterministic sequential aims to minimize total action cost. This work focus on the sequential satisfactory category, which aims to generate quick plans of reasonable quality.

2.1.3 Sequential satisficing track

The term satisficing, introduced by [5] in 1956, is a combination of satisfy and suffice, Simon Herbert used the term to describe the behavior of organisms which are not always looking for optimal solutions in nature but they apply a satisficing answer to a problem under circumstances in which an optimal solution is hard to find. They look for *"a path that will permit satisfaction at some specified level of all of its needs"*[5].

The goal of satisficing planning is to generate a quick (often non optimal) solution to the planning problem with reasonable quality.

In this category of the IPC 2014 (sequential satisficing track) a total of 43 planners registered facing a total of 23 different domains[6].

[6] `https://helios.hud.ac.uk/scommv/IPC-14/planners_actual.html`

Task domains are very diverse and are applied to various scenarios. For example, one task is to transport packages between cities, while another might be to serve drinks at a bar. It even includes assembly problems, like tetris.

No previous knowledge about the competition domains is given to the planners, and in fact are kept secret. Planner were given with the opportunity to fix bugs in their code during the competition.

It is an important requirement for the competitors in IPC to give the organizers the right to post the source code of their planners on the official website[7], additionally they have to publish a paper describing the approach.

A repository for all IPC 2014 planners and problems is open and available for download on the IPC 2014 website [8].

The scoring function used at the IPC focus on speed and plan quality and offers one way to evaluate the planners; however the community is encouraged to run their own experiments and draw their own conclusions.

2.2 Problem Domain Definition Language (PDDL)

The problem domain definition language (PDDL) was developed in 1998 by top planning researchers such as Drew McDermott, Malik Ghallab, Manuela Veloso, Adele Howe and others, as an attempt to standardize input to planning languages. Inspired by Stanford Research Institute Problem Solver (STRIPS), Action Description Language (ADL), partial order planner (UCPOP) and other planner formalism (see Figure 2.2), PDDL was introduced in the first International Planning Competition (IPC)[6].

STRIPS[7] developed in 1971, without doubt a milestone in planning, was able to generate novel solutions to planning problems and exposed a syntax for problem description which became a standard for many years in the AI community. Edwin Pednault (currently working at IBM T. J. Watson Research Center) observed in 1987 that the expressive power of STRIPS can be improved by allowing the effects of an operator to be conditional[8]. This idea lead to the language ADL[9] (action description language) which is considered an improvement of STRIPS.

Some of the advantages of having a common input standard to planning problems are code reusability, and the possibility to have a direct fair comparison between different planning algorithms.

[7] `https://helios.hud.ac.uk/scommv/IPC-14/`
[8] `https://helios.hud.ac.uk/scommv/IPC-14/errPlan.html`

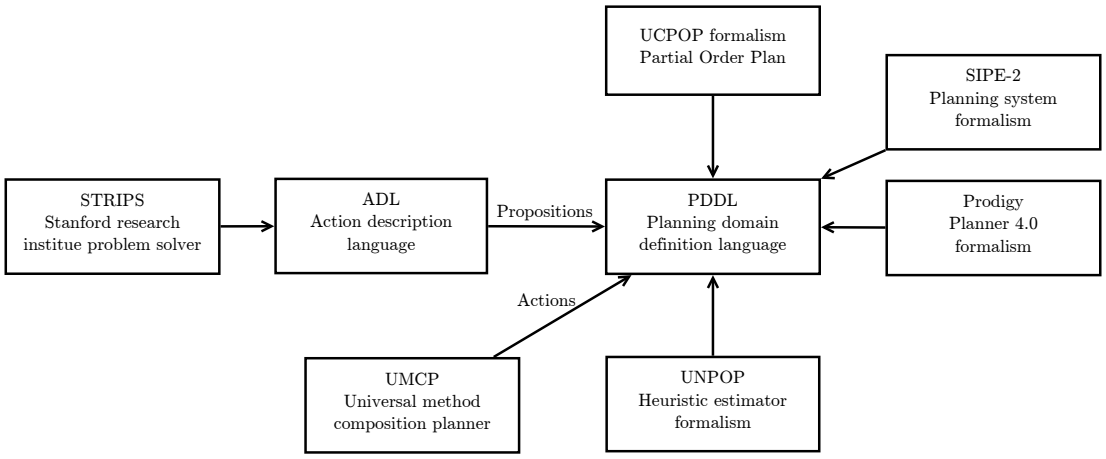


FIGURE 2.2: PDDL is composed by taking ideas from different planners.

During the development of PDDL language, most planners at that time required some "advice" or small annotations which were encoded inside the problem description, PDDL was designed to describe what they called the "physics" of the domain, providing no additional help to the planners [6].

The PDDL language has evolved with every IPC competition and different versions were released in the last 18 years:

PDDL 1.2 - Used in 1st and 2nd IPC (1998 & 2000), initial version. It supported basic STRIPS style actions, conditional effects, domain axioms, safety constrains, hierarchical actions, etc.

PDDL 2.1 - Used in 3rd IPC (2002), Introduces numeric fluents, plan metrics, durative and continuous actions.

PDDL 2.2 - Used in 4th IPC (2004), Introduces derived predicates and timed initial literals, no big changes in this review.

PDDL 3.0 - Used in 5th IPC (2006), Introduces state trajectory constraints and preferences. Includes features for recent developments in planning at that time.

PDDL 3.1 - It is the most recent version, it was used in 6th, 7th 8th IPC (2008, 2011, 2014), introduces object fluents (functions could be of any object type). There is a change in expressiveness of the language semantically speaking.

There are some variants of PDDL, for example the New Domain Definition Language (NDDL) created by NASA[9] in 2002, used in projects such as EUROPA[10].

Another extensions of PDDL include for example the Multi Agent Planning Language (MAPL). It was created in 2003 and introduces non propositional state variables and

[9] `https://www.nasa.gov/`

a temporal model with modal operators. Another example includes the Probabilistic PDDL (PPDDL) as an extension created in 2004 and 2006 to express problems and domain definitions in the 4th and 5th probabilistic track of IPC.

Example PDDL 3.1 code from IPC 2014 is shown below in listing 2.1.

```
(define (domain transport)
  (:requirements :typing :action-costs)
  (:types
        location target locatable - object
        vehicle package - locatable
        capacity-number - object
  )

  (:predicates
     (road ?l1 ?l2 - location)
     (at ?x - locatable ?v - location)
     (in ?x - package ?v - vehicle)
     (capacity ?v - vehicle ?s1 - capacity-number)
     (capacity-predecessor ?s1 ?s2 - capacity-number)
  )

  (:functions
     (road-length ?l1 ?l2 - location) - number
     (total-cost) - number
  )

  (:action drive
    :parameters (?v - vehicle ?l1 ?l2 - location)
    :precondition (and
        (at ?v ?l1)
        (road ?l1 ?l2)
      )
    :effect (and
        (not (at ?v ?l1))
        (at ?v ?l2)
        (increase (total-cost) (road-length ?l1 ?l2))
      )
  )
)
```

Listing 2.1: Partial example code from IPC 2014 (transport domain)

The complete transport domain is available in the following website[10].

2.3 Mercury planner

Mercury is a sequential satisficing planner developed by Michael Katz and Jörg Hoffmann from Saarland University (Saarbrücken, Germany) implemented in the Fast Downward planning system[11] (see Section 2.4). Mercury participated in sequential satisficing and agile tracks of the IPC 2014 and won 2nd place in the sequential satisfactory track. It was also granted with the innovative planner award, given to planners who expose novel techniques in deterministic AI planning.

The planner starts with a initial greedy best first search and once a solution is found it performs multiple iterations of heuristic search with a weighted A* algorithm.

[10] `https://helios.hud.ac.uk/scommv/IPC-14/repository/demo-instances.zip`

Mercury planner introduces a partial delete-relaxation heuristic called Red-Black. Red variables take the relaxed semantics while black variables take the regular semantics[12].

Relaxed problem heuristics were first proposed by Heuristic Search Planner (HSP)[13] in 1997 and improved later on by famous Fast Forward planner (FF)[14] in 2001.

The idea behind relaxed problem heuristics is simple, in order to reduce the search space, the algorithm ignores the negative effects of operators (delete list) to construct a simpler graph which could be searched in polynomial time. This causes odd fictional situations, for example an agent would be allowed to be in many places at the same time, however this concept helps to generate powerful heuristics that are used later on by the planners during the final search.

The Red-Black heuristic extends the delete list relaxation concept by ignoring the negative effects of "some" variables which are called Red, while the rest of the variables (called Black variables) follow the usual rules[12].

The method for finding black variables is called "the paint strategy" and is domain dependant. The delete relaxation heuristic or Red Black has played an important role in the success of satisficing planning systems[15].

According to IPC 2014 website[11] mercury planner supports the following (PDDL) representations: predicate, object fluent, typed, untyped, schematic, grounded; the following conditions: negative, action description language (ADL); effects: conditional, universal and support for derived predicates. A comparison regarding PDDL supported features between mercury and top 5 IPC 2014 planners can be consulted in table 2.1.

	ibacop	mercury	miplan	jasper	uniform
predicate representations	yes	yes	yes	yes	yes
object fluent representations	no	no	no	yes	no
typed representations	yes	yes	yes	yes	yes
untyped representations	yes	yes	yes	yes	yes
schematic representations	yes	yes	yes	yes	yes
grounded representations	yes	yes	yes	yes	yes
negative conditions	yes	yes	yes	yes	yes
ADL conditions	yes	yes	yes	yes	yes
universal effects	yes	no	yes	yes	yes
derived predicates	no	no	yes	yes	yes

TABLE 2.1: Top 5 IPC 2014 planner PDDL supported features comparison.

[11] `https://helios.hud.ac.uk/scommv/IPC-14/repository/table_seq_sat.pdf`

2.4 Fast Downward planning system

Fast downward is a deterministic progression planner based on heuristic search. The original version of the planner was developed by Helmert[16] and was first introduced in the 4th IPC in 2004.

The Fast Downward planner first converts the planning problem (expressed in the PDDL language), into a data structure convenient for itself. However, we do not review the planner, but instead we cover the planning framework this planner implements. According to J. Hoffman[17], Fast Downward planning framework[12] has evolved into the main platform for implementation and evaluation of planning algorithms.

Here is a list of some planners from IPC that are implemented in the Fast Downward platform:

- Fast Downward Autotune
- Fast Downward Stone Soup
- LAMA 2011
- BJOLP
- LM-Cut
- Merge and Shrink
- Selective Max
- Mercury planner
- and more...

Downward planning system pipeline consists of three major steps: translate, knowledge compilation (preprocess) and search (see Figure 2.3)

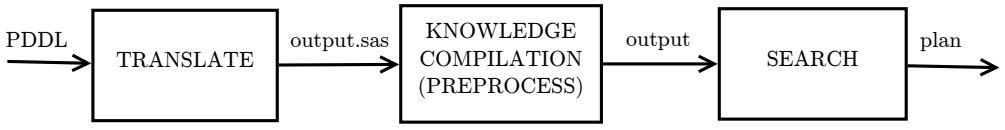


FIGURE 2.3: Fast Downward planning system pipeline.

Translate - To parse and convert PDDL specification into a non binary form which is convenient for planning approaches and performs grounding of axioms and operators. The output of the translator component is a multi-valued planning task[11].

[12] `http://www.fast-downward.org/`

Knowledge compilation (preprocess) - Generates 4 different data structures: domain transition graph, causal graph, successor generator and axiom evaluator[11].

Search - Planner specific, a search must be performed based on some heuristic. For example, in mercury planner it starts with an initial greedy best first search and then it performs multiple iterations of Black-Red heuristic search with a weighted A*.

2.5 ROSPlan

Developed at King's College London (KCL) University by M. Cashmore, M. Fox et al[18], with funding from EU projects PANDORA (288273) and Squirrel (610532), KCL ROSPlan introduces a framework with a generic method for task planning using Problem Domain Definition Language syntax (PDDL). It targets robotic platforms based on the robot operating system (ROS)[13].

It exposes a Knowledge base, an automatic PDDL problem generator, a graphic user interface to display current semantic state of the world and a plan dispatcher (scheduler) based on actionlib messages.

By using this framework a robot can benefit from task planning algorithms to perform planned actions in simulation or real world.

There are challenges that arise when dealing with the integration of task planning and real robotic platforms. For example, given a domain model and a initial state, the system needs to match the current environment with its internal representation. Furthermore actions coming from the planner based upon an abstract model of the world must be made concrete and executed by low level controllers, plans need to be executed with a strategy that allows the eventual possibility of plan execution failure due to the dynamics of the environment or upon request of the user by changing the desired goals.

According to ROSPlan authors their main contribution is to have provided with an open standard and implementation of an integrated task planning and execution framework that brings together standard components in planning and robotics (PDDL and ROS).

2.6 RoboCup @Work league

While most of the teams in the league are using finite state machines to create behaviors on their robots, the LUHbots @Work team[19] from Leibniz University of Hanover use

[13] `http://www.ros.org`

a graph-based search task planning approach with a greedy algorithm that minimizes different costs.

The costs taken into account are:

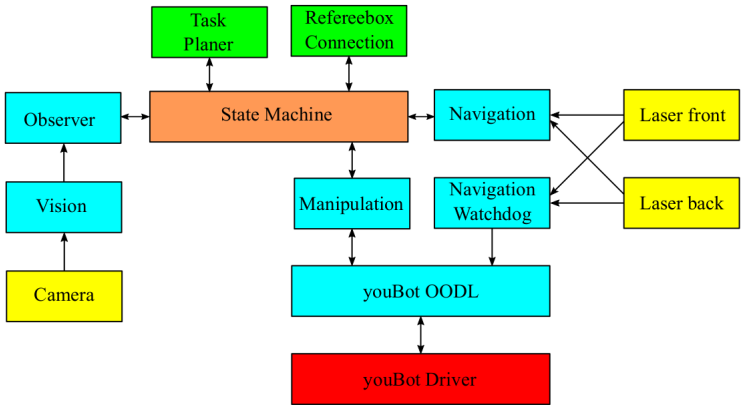
1. Expected execution time for each action
2. Likelihood of success.
3. Navigation costs based on the pre-computed distances between service areas

The behavior of their robot is controlled by a state machine. In the first states the Refereebox connection component gets triggered and waits for a task specification with information about initial situation of the world and desired configuration.

After receiving the task the system converts the information into a suitable representation for their system and then triggers the planner to generate a solution. The execution of the plan is called via state machine that interacts with the different components of the robot (Navigation, Manipulation, Perception, etc). (see Figure 2.4)

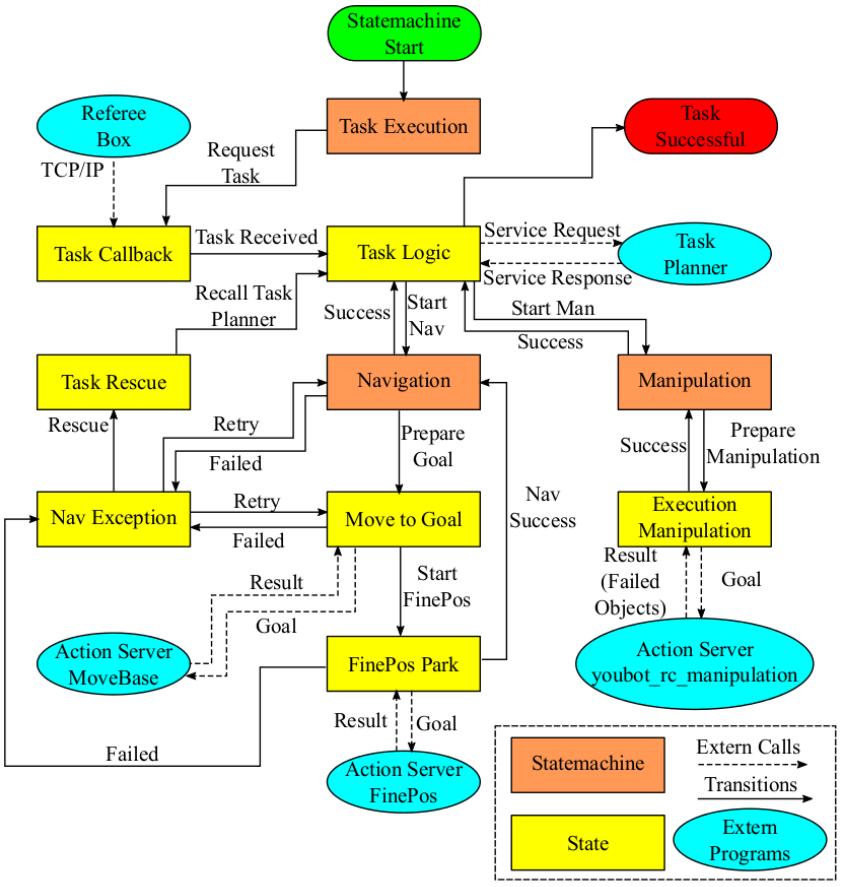
The @Work LUH-Bots main state machine diagram is shown in figure 2.5 and is organized in sub-state machines of task planning, task execution, navigation and manipulation. The state machine is based on action client servers; it sets goals in navigation and manipulation components and receives feedback about execution status.

According to the authors, their state machine is designed to recover from robot failure. It uses recovery mechanisms applied on the current task by retrying or postponing the action.



Credit: *LUHbots RoboCup@Work 2016 Team Description Paper [19]*

FIGURE 2.4: Robocup@Work LUH bots team, software architecture.



Credit: *LUHbots RoboCup@Work 2016 Team Description Paper [19]*

FIGURE 2.5: Robocup@Work LUH bots team, main state machine.

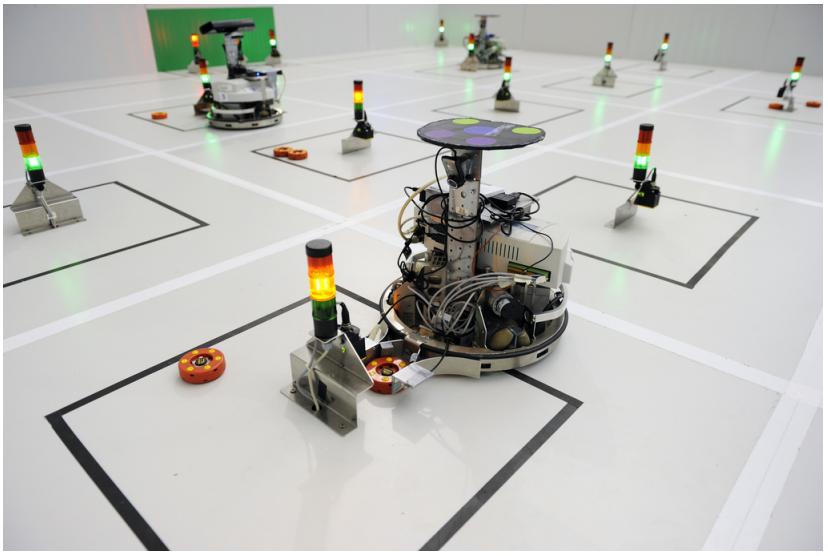
2.7 RoboCup logistics league

2.7.1 Introduction

RoboCup logistics league (RCLL)[14] started as a demonstration during the competition of 2009[15] in Graz (Austria) and is part of the major leagues since 2012; it focuses on in-factory logistics applications.

In this competition robots are interacting in a simplified factory like environment (see Figure 2.6) to achieve production goals. The robots need to fetch raw materials from storage and transport them between different workstations. In each step the task is to process the piece until it transforms into a final product.

Each team consists of three robots which are modified versions of the standard's league platform "Robotino", manufactured by Festo company [16].



Credit: *http://www.robocup2013.org/*

FIGURE 2.6: RoboCup Logistics league simplified factory like environment, Eindhoven 2013.

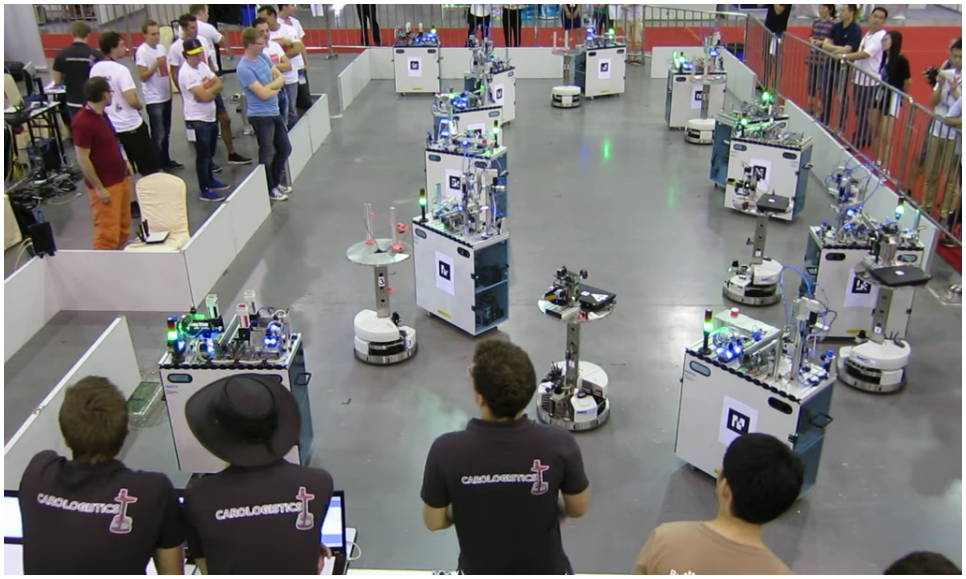
Robots in this competition are to collaborate with each other in a partially known environment. Their major tasks are exploration, production and execution/monitoring. In the exploration phase the robots need to gather information about the workstations that are available in the factory, after that they must complete the production chain as often as possible, minimizing idle time and dealing with incomplete knowledge.

[14] `http://www.robocup-logistics.org`
[15] `http://www.robocup2009.org/`
[16] `http://www.festo-didactic.com/int-en/learning-systems/education-and-research-robots-robotino/`

2.7.2 Logistics league evolution

RoboCup Logistics league started with a hockey challenge in the first edition in 2009 however the organization committee considered that it was too close to the already existing RoboCup soccer league and wanted something related to industrial applications. In 2010 the first logistics demonstration took place in Istanbul (Turkey) with focus on industrial production but with a simplified environment (see Figure 2.6).

The league has taken a major step since China 2015 competition and instead of moving pucks on the floor now the robot operates between stations based on a production system provided by Festo (see Figure 2.7).



Credit: *Carologistics: https://www.youtube.com/watch?v=MMs3DuzRG4s*

FIGURE 2.7: RoboCup Logistics league factory like environment, China 2015.

In 2014 a Gazebo[17] based simulator was developed by Zwilling[20] with the intention to be used in logistics league[21]. It can simulate many robots in a 3D environment with a multi-level abstraction layer (MLA), this offers the possibility to access low-level sensor data or to extract higher-level information through accessor plugins, it provides high-level control software interface for actuation and this feature in particular (according to the authors) aims to attract new users from the planning community[20].

[17] `http://gazebosim.org/`

2.7.3 Carologistics RoboCup team

The Carologistics RoboCup Team from Aachen (Germany), four time champion in the official Robocup logistics competition[18], released their full software stack used for winning the RCLL 2014. The open source software, is based on the Fawkes Robot software framework (see Section 2.7.4) and contains components for localization, navigation and perception along with basic behaviors (skills) using Lua-based behavior engine and the complete task-level executive based on CLIPS[19] (see Section 2.7.8).

2.7.4 Fawkes: A robotics software framework

Fawkes is a open source software framework for real-time robotic applications released in 2009. Similar to ROS it provides infrastructure and building blocks to create and run applications with a multi-thread approach. The first base system was created in February 2007[20] to account for game speed in soccer robots, and since then Fawkes has continuously been improved and extended with focus on the middle size RoboCup League[21]. However is also used in other RoboCup leagues (@home and logistics league).

It was developed mainly at Aachen University (Germany) by the Knowledge-Based Systems Group with large contributions from RoboCup teams AllemaniACs[22], Carologistics[23] and a large number of developers spread across two continents.

Support for Nao robot[24] was added in early 2008 with cooperation from Aachen, Cape Town, and Graz University of Technology (Germany, South Africa, and Austria)[22].

The Fawkes framework is written in C++, targeting Linux and Unix based systems with efforts towards having a clear structure and documentation.

2.7.5 Fawkes behavior engine

The behavior engine is basically a framework to develop, execute and monitor skills. It is an interface between low level systems and the high level agent.

In Fawkes they have identified a pattern in multi-agent robotic systems regarding their reasoning process which inspired them to create a “main loop” function. The loop

[18] `https://www.robocupgermanopen.de/,http://www.robocup.org/`
[19] `https://www.fawkesrobotics.org/projects/llsf2014-release/`
[20] `https://www.fawkesrobotics.org/about/history/`
[21] `http://www.robocup.org/robocup-soccer/middle-size/`
[22] `https://robocup.rwth-aachen.de/`
[23] `https://www.carologistics.org/`
[24] `https://www.aldebaran.com/en/cool-robots/nao`

consists of three steps, it starts with a vision subsystem that perceives the environment followed by a reasoning component that makes a decision about what to do and in the last step instructions are sent to the controllers for executing actions.

The Fawkes design emphasizes a centralized world model and primitive actions that can be used at a planning level; all data used by the plugins (or components) is centralized and stored in a "blackboard" which shares the information within the network.

Monitoring happens hidden to the agent which only receives a binary notification of action success or failure.

The Architecture considers three behavior levels, at the low level the robot components such as vision systems (i.e. object detection), navigation (i.e. localization), manipulation (to manage arm movement) and world model (to store all the knowledge from the agent).

The agent behavior is encoded at the middle-level in "skills" mainly as a combination of monitored low-level components coordinated by a state machine written in Lua[25].

Finally the Fawkes behavior engine considers the agent at the deliberation (top) level along with the task description and the skill decision making mechanism which can come from a state machine, a planning algorithm, etc.

2.7.6 Lua, the chosen programming language for skills in Fawkes

Fawkes authors believe Lua to be a superior programming language compared to Python when it comes to model behavior and in particular for state machine development (traditionally state machines are written in Python - SMACH ROS[26]).

Lua is an embeddable, light weight yet powerful and fast language. It uses a table as major data structure while other languages such as python use lists, array, tuples.

Lua has been used in many projects, for example in NASA Space shuttle hazardous Gas detection system[27] and AI games (CryEngine and World of Warcraft).

According to Fawkes architect and lead developer Tim Niemueller on a talk given in Willow Garage on Sep 30th[28] on average Lua is twice as fast as Python 3.

By using Roslua[29] the Fawkes behavior engine (written in Lua) can be used in ROS based systems[21], however unmaintained since 2011.

[25] `http://www.lua.org/`
[26] `http://wiki.ros.org/smach`
[27] `http://lua-users.org/wiki/LuaUses`
[28] `https://vimeo.com/16713496`
[29] `http://wiki.ros.org/roslua`

2.7.7 Fawkes vs ROS, differences and similarities

While ROS software architecture organizes the code in nodes, Fawkes components are written using plugins. Functionality is provided via aspects where you pass direct pointers for easy information access. This allows a quicker reaction of the system which specially soccer robots can benefit from; for example if the soccer robot reacts to a ball a few milliseconds late, it might miss it.

ROS and Fawkes are communication middlewares for robotic applications and they both follow a component based approach. They provide pre-built ready components for typical tasks (i.e. Navigation, Manipulation).

Both frameworks are able to to exploit modern multi-core technology as well as running across multiple machines (although this is slightly easier in ROS).

Fawkes was designed to have a low latency on closely integrated system while ROS (being a loosely distributed system) tries to use as much computational power as possible.

2.7.8 Carologistics CLIPS-based task coordination

In competition the three logistic robots are cooperatively performing a production task, communicating information about their intentions but also exchanging information.

According to the authors this continuous update of information suggests an incremental reasoning approach[23].

The robots need to deal with dynamic production chains in which no information about what needs to be produced is known in advance, furthermore the workstations are subject to failure and the referee can decide to temporary put the stations down for some time.

Most task planning approaches break down due to the presence of non-deterministic events[24] and the fact that the knowledge about the world is incomplete most of the time. Therefore the team followed a rule-based approach which they have implemented in CLIPS, a tool for building expert systems created by NASA in 1985 (see Section 2.8.3).

The (CLIPS) rules interact closely with a world state representation stored in a knowledge base. The stored information includes for example the locations and type of workstations in the factory (gathered during exploration phase), game information such as orders, machine status, remaining time and gripper state (open-closed). The sanity of the World model is periodically checked for inconsistencies and repaired if possible.

CLIPS condition-action rules are based on a first order logic chain system using a graph-based rete algorithm (pattern matching) with Lisp[25] syntax.

The Carologistics RoboCup team have around 220 rules and 50 facts in their system with more than 200 000 fact changes in a 10 min game; however rules are hierarchical and only one is executed at a time.

The skills in the behavior engine (see Section 2.7.5) get triggered by rules which act based upon the stored knowledge. In CLIPS it is possible to match facts against each other to trigger skills[21].

An example of a rule in CLIPS from RoboCup logistics league winter school 2015, (Carologistics, Aachen University in Germany) is shown below in listing 2.2.

```
1 (defrule <rule-name> [<comment>]
2 [<declaration>]          ; Rule properties
3 [<conditional-element>]*
4
5 =>
6
7 <action>*
8 )
```

Listing 2.2: Example of a rule description in CLIPS

During competition it is often the case that the connection between the agents gets interrupted, however the decision making mechanism is decentralized and each robot acts on its own based upon the latest available shared knowledge.

2.7.9 Lua based Hybrid State Machines (HSM)

In Fawkes every skill is implemented as a separate Lua module and are modeled using hybrid state machines. They have three different return value: running, finalized and failed.

A hybrid state machine (HSM) is a directed graph where the nodes represent the states and the edges are transitions among these states. The difference between traditional state machines and hybrid ones is mainly that they have access to global variable information common to the state machine. They also allow for continuous and discrete process changes.

2.8 NASA Planning and Scheduling Group

The Planning and Scheduling Group from NASA has developed techniques for automated planning, scheduling and control[30].

Some of their core technologies are mentioned below.

2.8.1 PLEXIL

"*All space missions require execution systems to perform commands and monitor the environment*"[26], the spacecrafts are controlled primary by humans but they have systems which are continuously monitoring and reporting important events, it would be impossible for a small team to attend all subsystems in real time. For this purpose, a system called PLEXIL (plan execution interchange language) has been developed by NASA, and is one of their core planning technologies available for download trough their website[31]. PLEXIL is a programming language used to represent plans and comes together with an execution engine that efficiently implements the PLEXIL language and provides an interface to controlled systems.

2.8.2 Space planning and EUROPA

Europa is a framework to model and solve scheduling, constraint programming and planning problems. Is free and available under NASA's open source agreement (NOSA). It is designed to be configurable and extendable, the system can do among other things: "*limited reasoning with infinite and dynamic domains*" and "*reason about resource production and consumption*"[10]. This last statement is particularly interesting for robotic platforms which perform always under limited resources constrains.

2.8.3 CLIPS

C Language Integrated Production System (CLIPS), is a public domain software built for expert systems. It was developed starting 1985 at the NASA-Johnson Space Center in Houston Texas. CLIPS presents an object oriented language for writing expert systems. Like other languages CLIPS deals with rules and facts to operate.

[30] `https://ti.arc.nasa.gov/tech/asr/planning-and-scheduling/`
[31] `http://ti.arc.nasa.gov/tech/asr/planning-and-scheduling`

Chapter 3

Background

In this chapter we present concepts and knowledge required to understand the rest of the thesis. We first describe the robot platform and then its available skills.

3.1 Robot platform

The robot platform used in this work is the KUKA youBot [27] (see Figure 3.1).

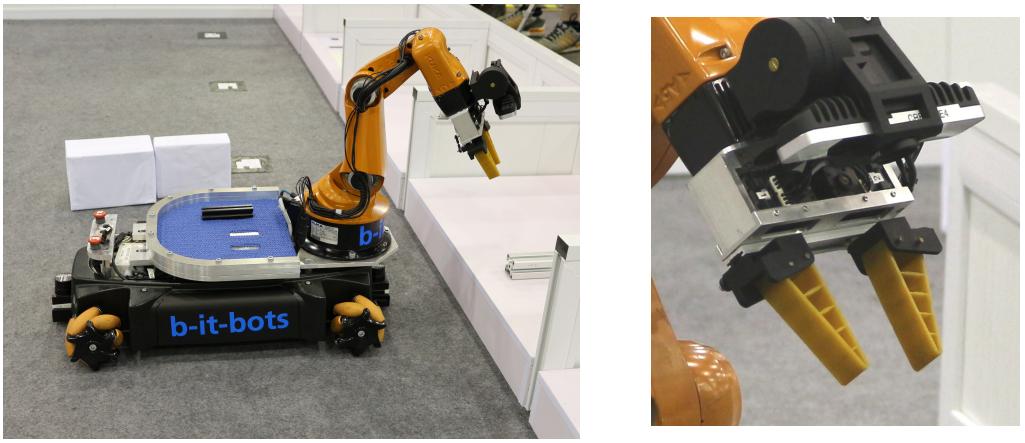


FIGURE 3.1: b-it-bots KUKA youBot and custom gripper.

It is equiped with an omnidirectional base with four swedish wheels, one manipulator of five degree of freedom and a custom two finger gripper with flexible fingers and current feedback motors. The robot has an intel SR300 realsense depth camera mounted on the arm and a custom grasp monitor based on infrared light.

The robot also has two Hokuyo URG-04LX laser range finders mounted on the front and back of the robot, each configured with a openning angle of 180 degree leaving a small blind spot are on the sides of the robot (see Figure 3.2).

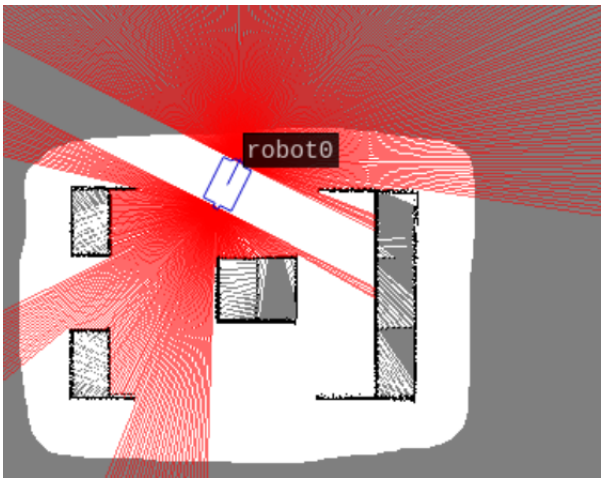


FIGURE 3.2: Simulation of the KUKA youBot robot, showing the specifics of the laser scanner readings.

The robot is used by the University team, b-it-bots[1] for education, research and to compete in the international RoboCup @Work competition[2].

3.2 Robot skills

In this section we briefly describe the components in our robot. This is important to understand the task integration work that is presented later in this thesis. It is important to mention that the contents of this chapter are mostly description of components developed by members of the b-it-bots team and the author of this thesis does not claim authorship on them.

3.2.1 Event based architecture

As described in chapter 1 our mobile manipulator KUKA youBot uses ROS[28] middleware as the chosen software framework. We decided to follow an event based architecture instead of traditional service calls (see Figure 3.3).

The reason behind it is because service calls in ROS might block certain processes, while event-based architecture can monitor outcomes from different components easily. Furthermore for simplicity while using/testing the components (in ROS it is easy to create string publishers and subscribers). This does not mean that we don't use service calls at all, but new components are developed by following the event architecture conventions established by the b-it-bots team.

[1] `https://mas-group.inf.h-brs.de/?page_id=23`
[2] `http://www.robocupatwork.org/`

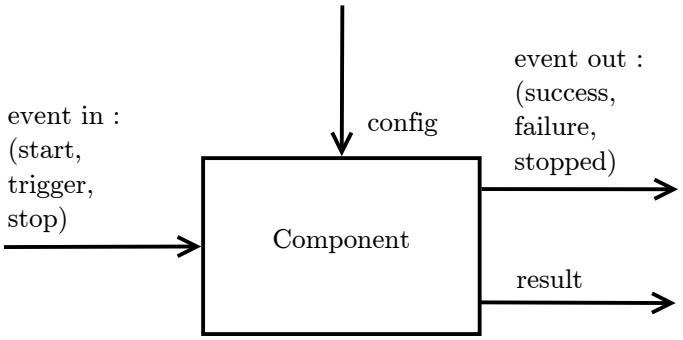


FIGURE 3.3: Event based example component.

3.2.2 Navigation

The "move_base"[3] component from the ROS navigation stack is configured and extended as described below.

Global planner - Based on global planner ROS package[4], orientations were added to a existing path produced via A* algorithm by dividing it into three sections. The first and third section exhibit omni-directional behavior while the second section drives in differential mode (see Figure 3.4). The global planner is called and executed by a local planner when a new navigation goal is received by move_base, but also gets called (but not executed) upon request via service call, this feature can be used for example to calculate a particular path length between two poses, taking into account the geometric constrains of the environment.

Local planner - Based on the ROS implementation[5] of the Dynamic Window Approach (DWA)[29] , the function of the local planner is to generate velocities for the mobile base to follow the global plan (path as array of poses) and towards the goal. The DWA algorithm generates random velocities in x, y and theta and forward simulates them for a small period of time (see Figure 3.5).

Trajectories that collide with any obstacles are discarded and remaining ones are scored based on path bias (how close they follow the path), goal bias (how close they drive the robot towards the goal) and oscillations related score (to prevent trajectories that create oscillations in the robot).

The current implementation of the local planner lacks support to score global plan orientations. This prevents the mobile base from making full use of the global plan and the omni-directional platform. As a workaround we adapted the DWA parameters to

[3] `http://wiki.ros.org/move_base`
[4] `http://wiki.ros.org/global_planner`
[5] `http://wiki.ros.org/dwa_local_planner`

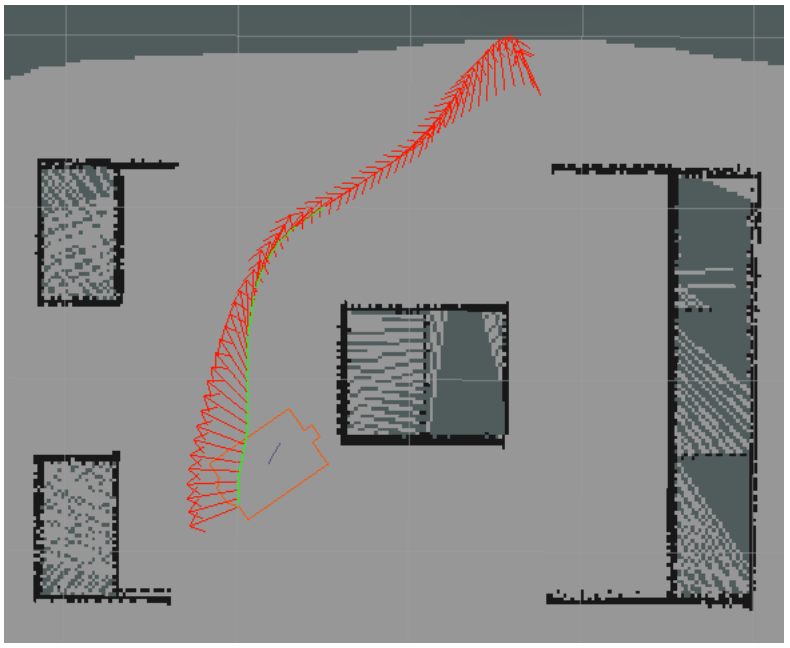
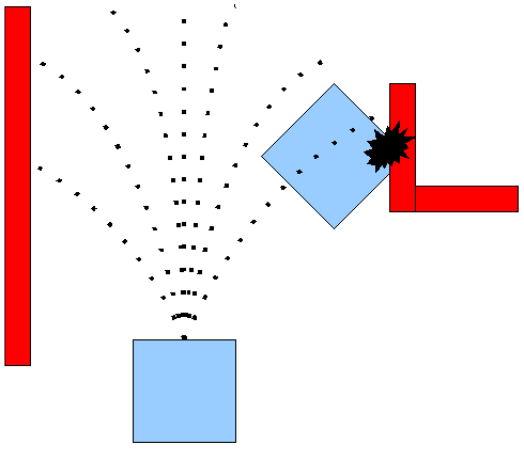


FIGURE 3.4: Custom global planner, example path with orientations expressed as array of poses.



Credit: *http://wiki.ros.org/dwa_local_planner*

FIGURE 3.5: Dynamic Window Approach random velocity generation.

restrain lateral velocity range, with the disadvantage of producing a behavior similar to a differential drive base.

Base controller - This component moves the base to a goal pose using a control algorithm without taking into consideration obstacles in the environment, this is useful to perform small base motions for example when the arm does not reach an object or to fine adjust the base to a particular pose.

Two controllers are available, map and odometry based, using the available localization information (odometry or map and particle filter). It performs a movement of the base

at a certain distance, for example moving 30 cm to the front or left, etc. (see Figure 3.6).

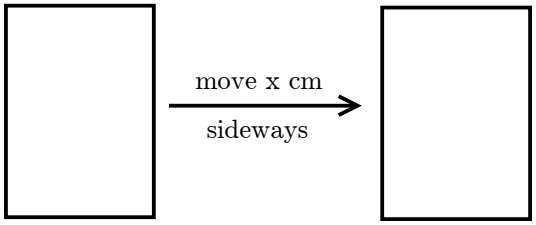


FIGURE 3.6: Relative base controller example:
moving in -y direction (sideways) a certain "x" distance.

Base alignment - This component is useful to align the mobile base towards service areas. It fits a line to laser scanner readings with linear regression and aligns the robot to the line. (see Figure 3.7)

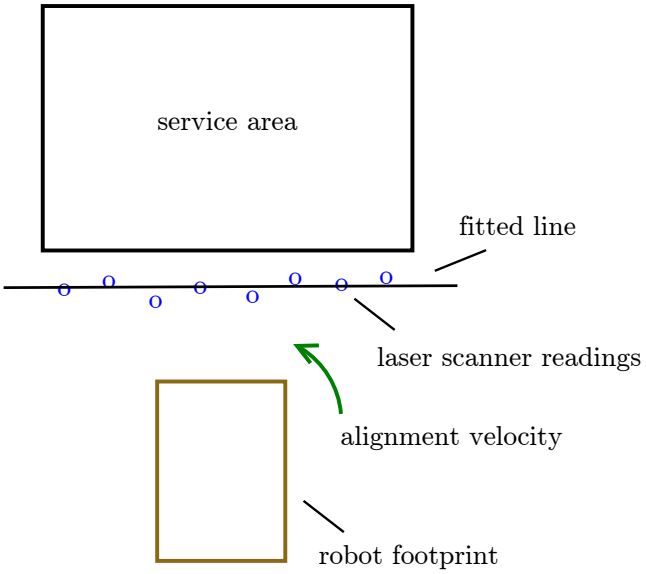


FIGURE 3.7: Base alignment towards service area.

Force field recovery behavior - An algorithm based on smARTLab[6] code release[7] was implemented, it drives the robot away from obstacles when it gets stuck.

Obstacles, which are encoded in the costmap, are treated as a force vector acting on the center of the robot (see Figure 3.8). The force vectors of obstacles within a specified radius are summed to obtain a resultant repulsive force (see Figure 3.9).

[6] `http://wordpress.csc.liv.ac.uk/smartlab/`
[7] `https://github.com/smARTLab-liv/smartlabatwork-release`

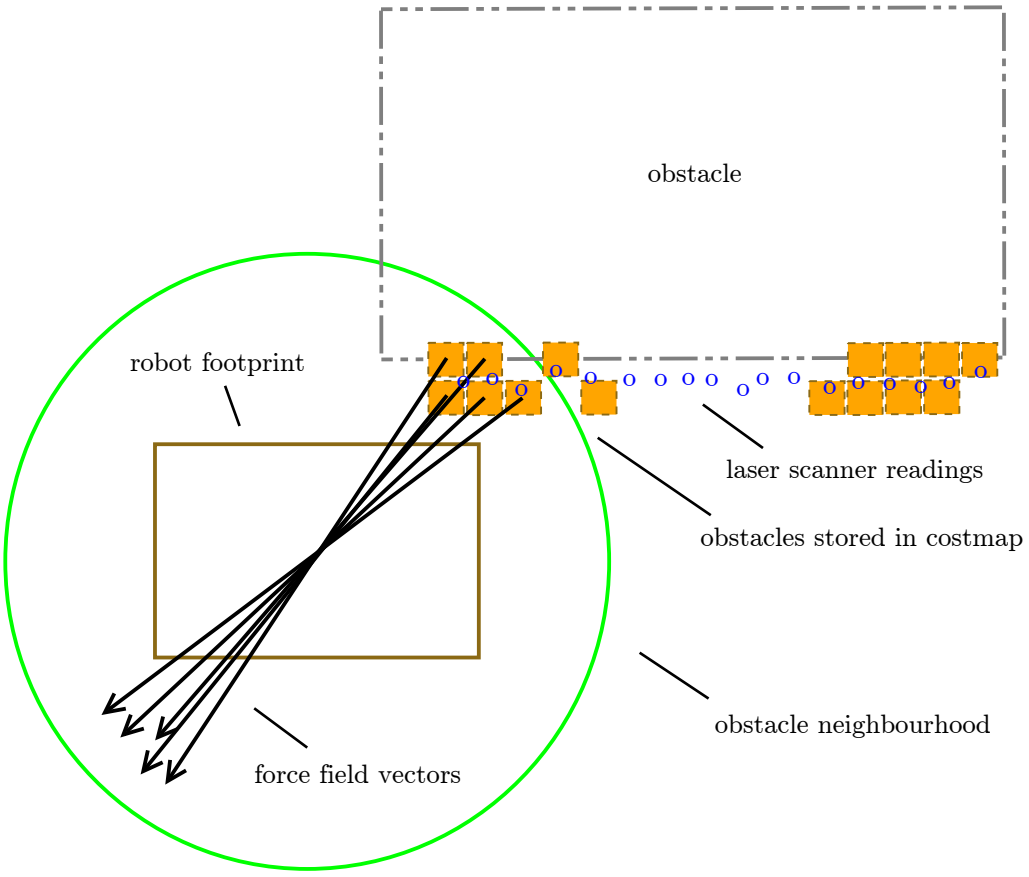


FIGURE 3.8: Force field vectors constructed from costmap information.

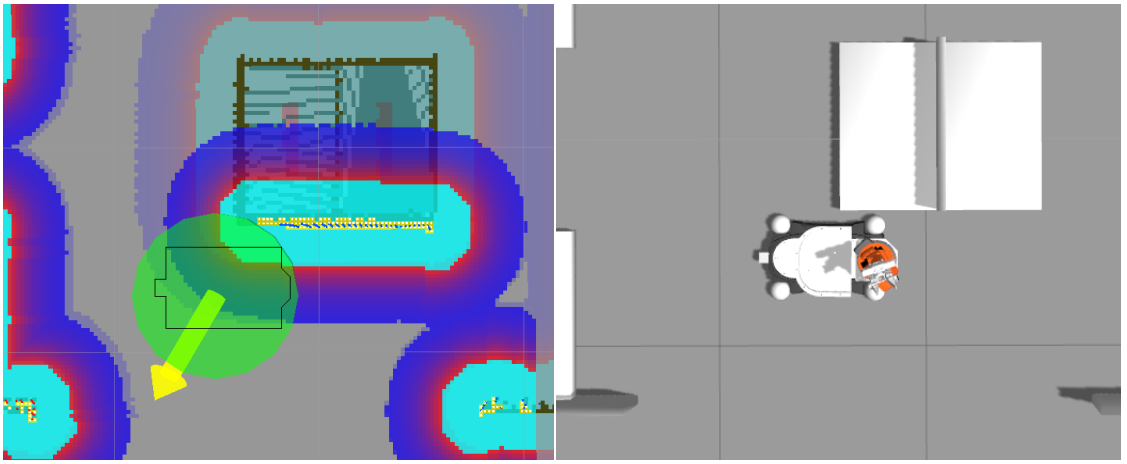


FIGURE 3.9: Force field recovery behavior for the mobile base getting away from obstacles in simulation.

In ROS Navigation a monitoring system is constantly checking for the progress of the base, if the quota is not fulfilled the recovery mechanisms get triggered.

Later on this work we will talk about the planner, which calls a navigation action with a timeout. The system will try to reach the goal and if it fails, it recovers and tries again (as long as there is time left).

3.2.3 Perception

The object perception functionality is divided into two main components: segmentation and recognition. The robot needs to recognize objects that are placed on a horizontal workspace. The robot aligns itself to the workspace and moves its arm such that the objects are in the field of view of the arm-mounted 3D camera.

The camera captures a pointcloud of the scene and a plane fitting algorithm is used to find the horizontal surface that supports the objects. Points above the supporting surface are then segmented and Euclidean clustering is applied on them. Each cluster represents an object that needs to be recognized.

Each cluster is classified using a previously trained support vector machine (SVM) based on features such as size and colour. The output of the object perception component is a list of recognized objects and their 3D poses.

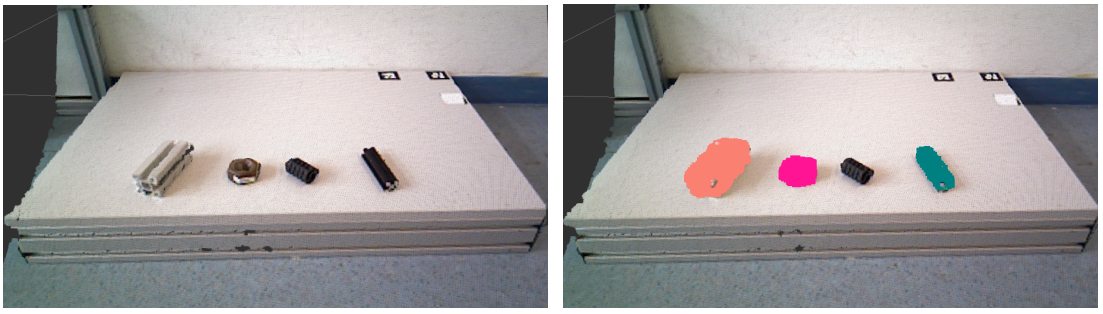


FIGURE 3.10: Original scene pointcloud and detected clusters.

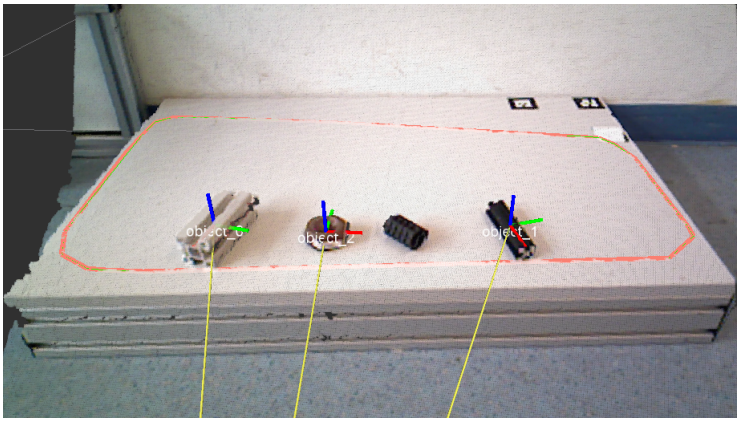


FIGURE 3.11: Object recognition example, three objects were recognized, one was not detected.

Figure 3.11 shows an example which exposes the limitations of the perception component. In this example three objects were recognized from a total of four. Failure could be attributed to many different factors such as the pointcloud quality, illumination conditions or the angle at which the objects are observed, etc. However previous experiments

(not discussed in this thesis) show that object perception might find the object if the robot tries to observe the object from a different angle by moving the arm or base and try again.

3.2.4 Manipulation

Manipulation components are based on MoveIt![8], a state of the art software for mobile manipulation which integrates the Open Motion Planning Library (OMPL), custom inverse kinematics of the arm, Universal Robot Description Format (URDF) and Kinematics and Dynamics Library (KDL) from Leuven University[9].

Modules in MoveIt! prevent the manipulator from planning a trajectory that collides with itself. This is achieved by factoring in the physical dimensions of the manipulator, which is encoded in the URDF file. Additionally, point cloud data can also be integrated for online collision detection with objects in the environment.

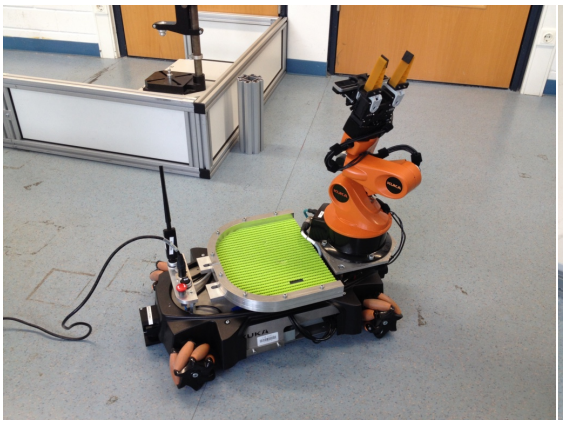
Some of the manipulation skills that we consider relevant for this work are described below.

Pre-recorded arm joint configurations - Several arm joint configurations are recorded and saved to move the end effector to predefined poses for specific tasks. This is useful for example to stage picked objects in the rear platform or to move the arm-camera towards the workspace. Some useful pre-recorded poses are shown for reference in Figure 3.12 along with their function, shown in table 3.1.

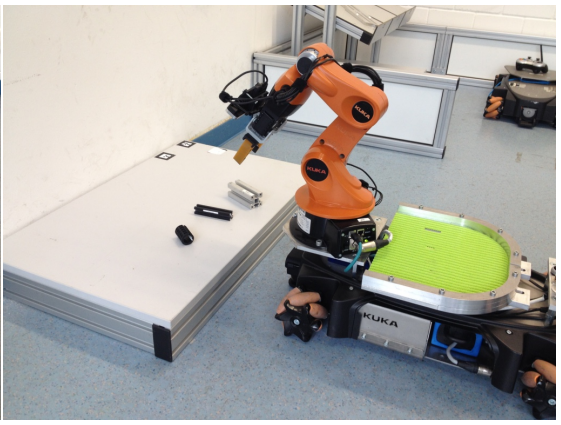
Arm configuration	Function
folded	to keep the arm within the robot footprint (used for navigation)
look at workspace	move the camera to the field of view of objects
platform left pre	prepare arm to stage/unstage an object from left platform
platform left	unstage/stage object in left platform
platform middle pre	prepare arm to stage/unstage an object from middle platform
platform middle	unstage/stage object in left platform
platform right pre	prepare arm to stage/unstage an object from right platform
platform right	unstage/stage object in left platform

Table 3.1: Pre-recorded arm joint configurations.

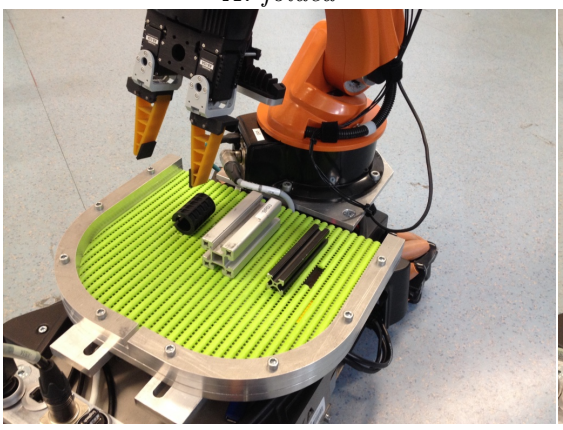
[8] `http://moveit.ros.org/`
[9] `http://www.orocos.org/kdl`



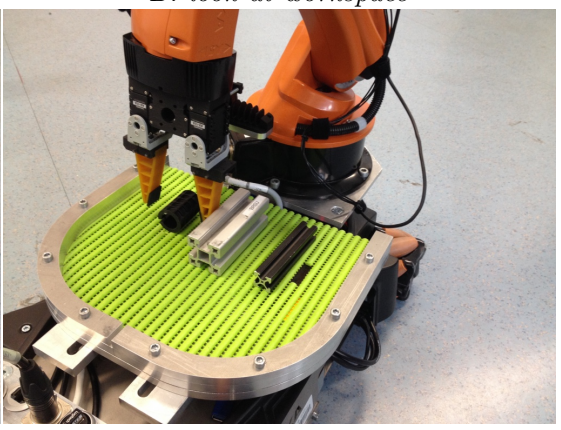
A. folded



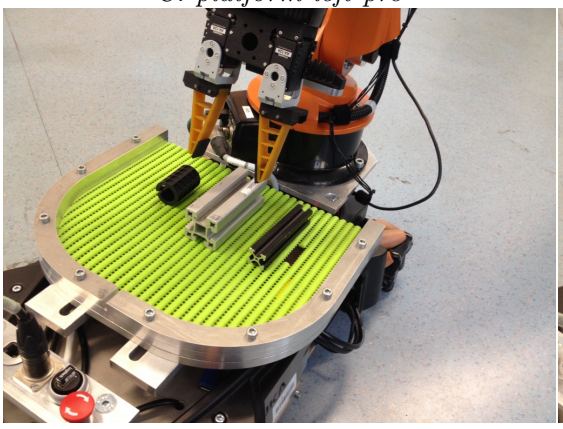
B. look at workspace



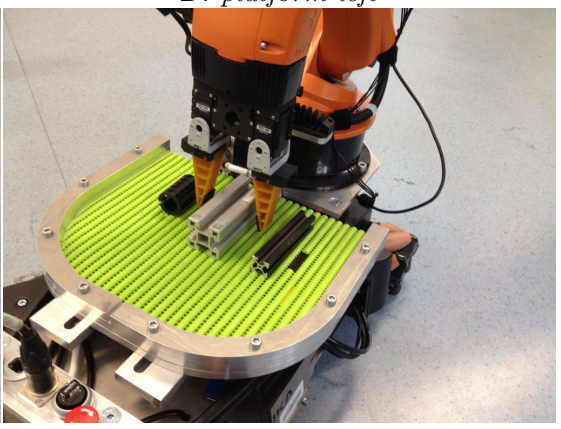
C. platform left pre



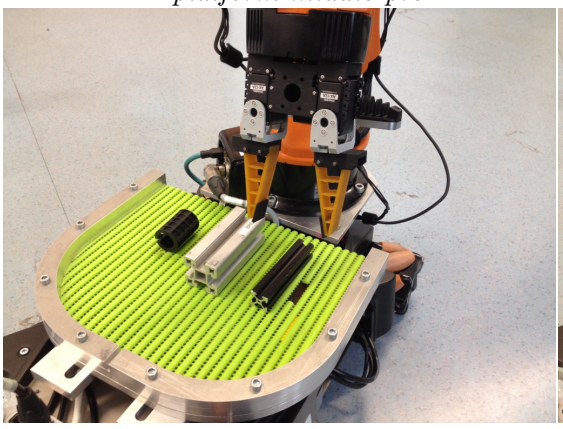
D. platform left



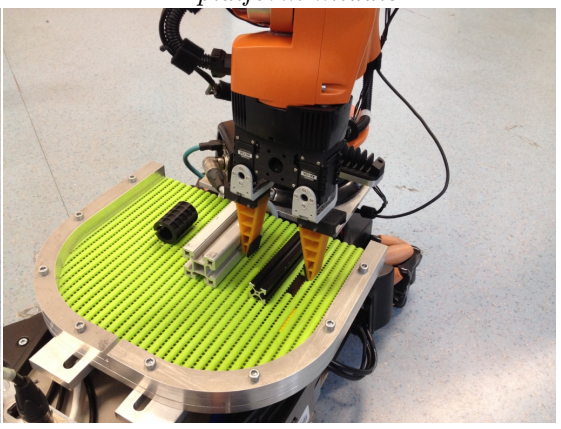
E. platform middle pre



F. platform middle



G. platform right pre



H. platform right

FIGURE 3.12: Pre-recorded arm joint configurations.

Pregrasp planner - This component is used to compute a feasible grasp joint configuration, based on a target pose provided by object perception (see Section 3.2.3) to move the end effector of the manipulator towards the desired pose. The problem arises from the fact that object perception component returns the transformation of the center of the object and this pose usually is not reachable by the arm.

Details of the pregrasp planner are not relevant for this work, instead we mention that a pick action could be formed by calling pregrasp planner component along with the move arm planned (using MoveIt!) and finally closing the gripper.

Grasp monitor - This component is based on a optic sensor (photo-transistor and infrared LED) which light gets blocked when a object is inside the gripper (see Figure 3.13). It confirms whether and object was grasped or not by the robot.

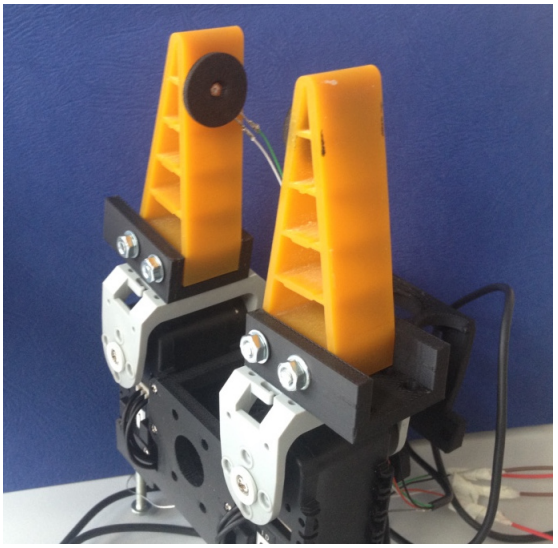


FIGURE 3.13: Grasp monitor installed on gripper, based on optic sensor.

3.2.5 Basic Transportation Test (BTT) state machine

Prior to this thesis the approach followed by the b-it-bots team to exhibit complex behavior in the robot was done via finite state machines written in ROS SMACH[10].

This section describes the basic transportation test (BTT) state machine that was used by the b-it-bots team during RoboCup China 2015 competition.

The top level state machine is shown in Figure 3.14, "GO AND PICK" sub-state machine is not expanded for simplicity, but is shown instead in Figure 3.15.

[10] `http://wiki.ros.org/smach`

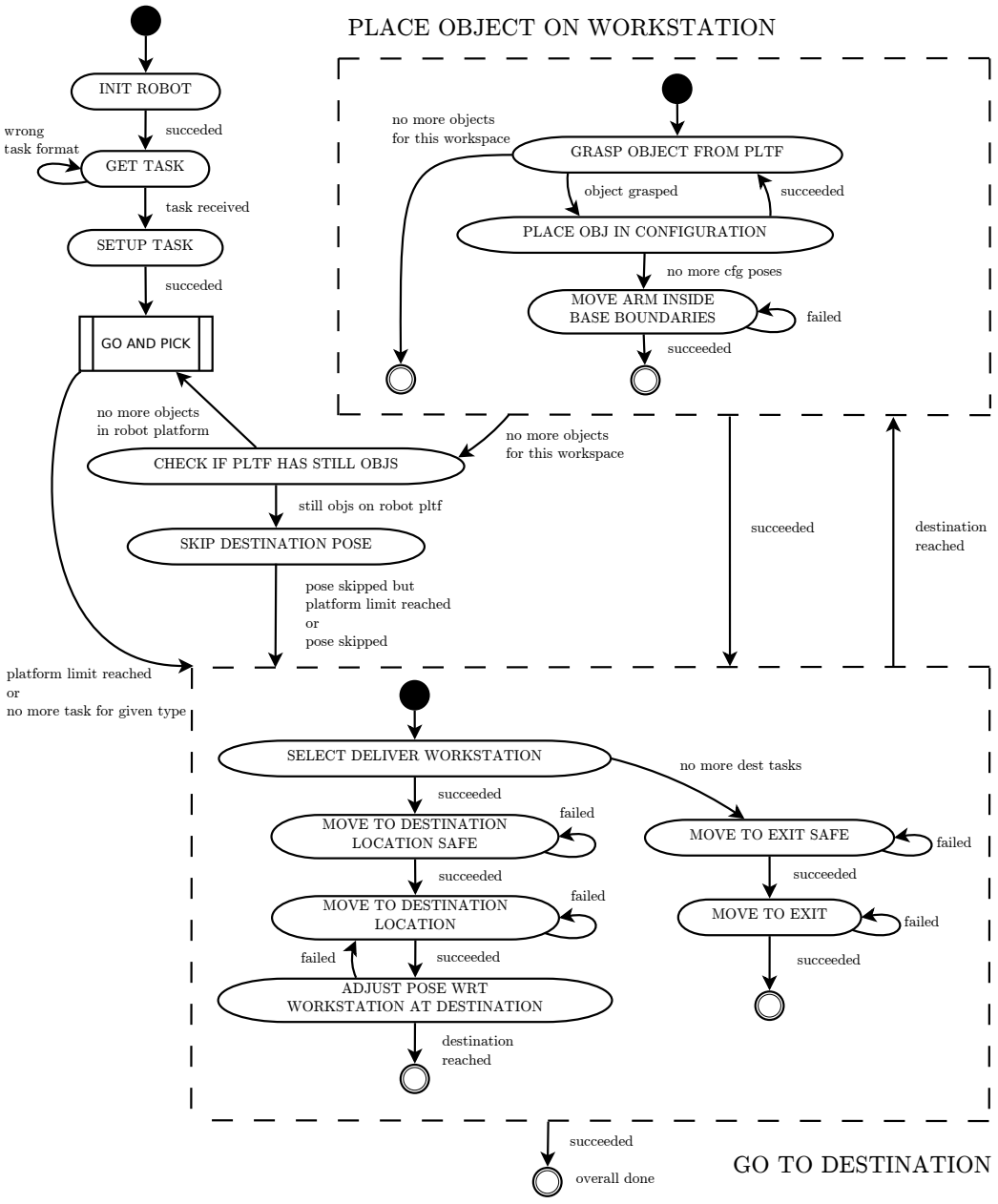


FIGURE 3.14: Top level Basic transportation test state machine.

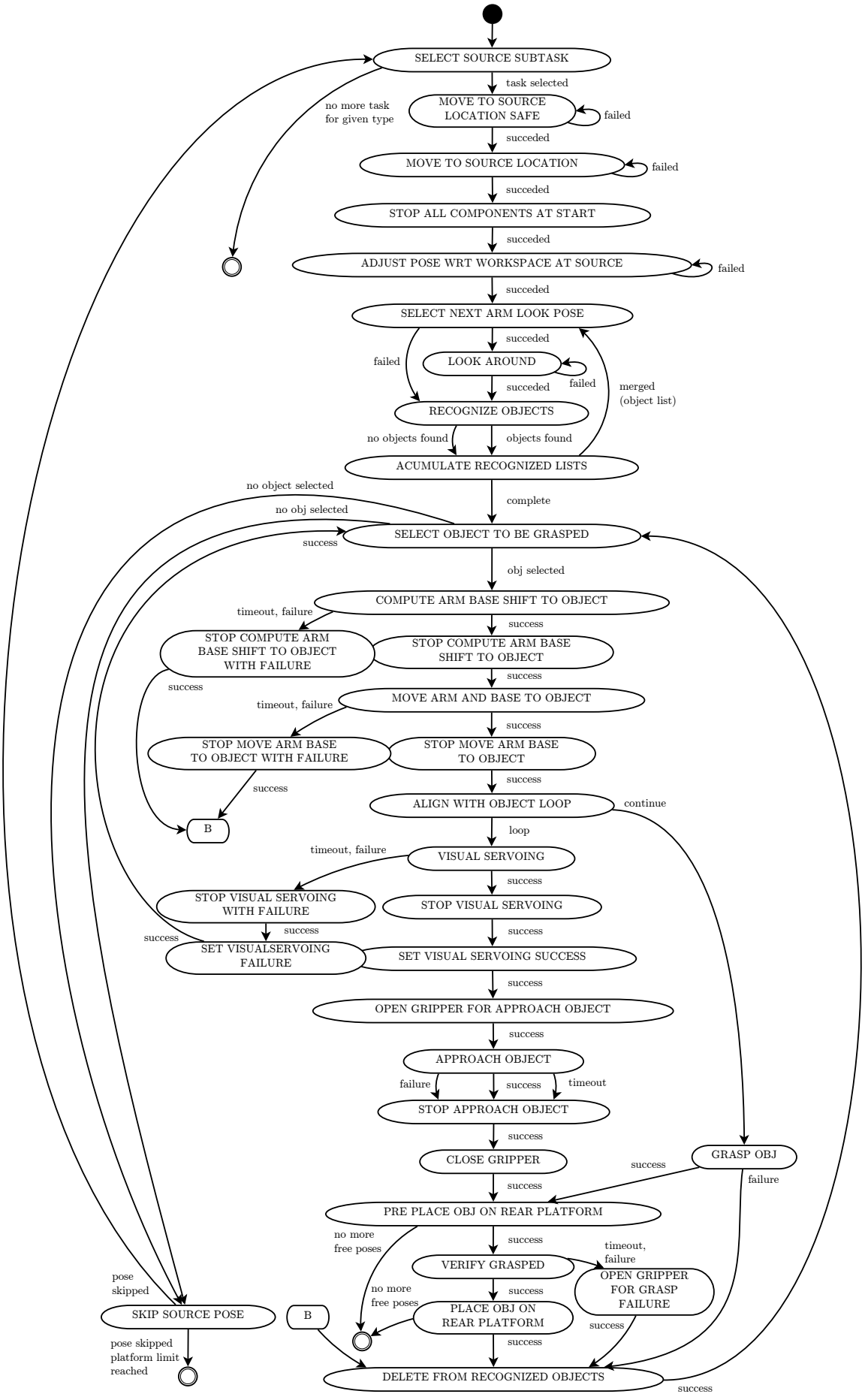


FIGURE 3.15: *"go and pick"* sub-state machine.

The basic transportation test (BTT) state machine (see Figure 3.14) starts by moving the arm inside the robot footprint and gripper is opened (INIT ROBOT), afterwards the robot waits for a task specification from referee box and parses the received instructions into sub-tasks (GET and SETUP TASK). For reference an example of a simple BTT task is shown below.

```
1 BTT<initialsituation(<S1,o1>);goalsituation(<S5,o1>)>
```

LISTING 3.1: Example of a basic transportation test task specification

Currently the following tasks are admissible:

- Basic Navigation Test (BNT)
- Basic Manipulation Test (BMT)
- Basic Transportation Test (BTT)
- Precision Placement Test (PPT)
- Competitive Transportation Test (CTT)
- Conveyor Belt Test (CBT)

A wrong task format state transition will trigger if the task is not one in the previous list. The robot then iterates over the sub-tasks (platforms with objects) by going and picking objects from a list which is stored as a variable in the state machine. After the robot platform limit has been reached a sub-state machine called "GO TO DESTINATION" gets called to move the base to deliver stored objects. Next the "PLACE OBJECT ON WORKSTATION" sub-state machine delivers the objects into the service areas or workstations and checks if there is still objects in the robot platform which need to be delivered to a different place. If there are no more objects on the robot platform then goes for picking again repeating this loop until task list is empty.

After having transported all required objects between locations the robot exits the competition arena.

As an example of the robot performing a basic transportation test, the reader might want to watch the following video[11], which shows the last b-it-bots RoboCup competition in Hefei (China, 2015). Furthermore, a description of the BTT state machine (Figures 3.14 and 3.15) (state by state) can be consulted in appendix A.

[11] `https://www.youtube.com/watch?v=PBI9_VxqQ_0`

Chapter 4

Approach and Implementation

Prior to this thesis, the b-it-bots team [30], as well as the majority of teams in @Work competitions, used SMACH finite state machines (FSM) [31] to enable the necessary complex behavior in their robots. FSM can be used to model many problems, however, as the task grows in complexity, the size of the FSM grows as well. This makes it difficult for human designers to maintain, extend and understand.

Other approaches such as the one followed by CaroLogistics (see Section 2.7.3) suggest the use of rule based systems, however the scenarios faced by them are far more complex that the challenges present in @Work league in terms of planning. Compared to RoboCup Logistics league, the @Work league is based in a deterministic static environment, where the location of all the objects are known and will not be moved by a third party during the competition. Logistics league deals with a multi-agent system in which up to six robots can be at the same time inside the arena, (three per team), while in @Work usually only one robot performs the task, however the task itself is more complex in terms of perception and manipulation.

We consider that deterministic task planning is a reasonable choice to exhibit complex behavior in @Work league because the environment is fully observable, however in our opinion, the challenge is in the complexity of the task that the robot needs to perform, which opens many possibilities for a real robot system to fail.

Deterministic planning approaches can be used to control the operational logic of single agents (or small groups of agents) in a structured and logical way.

Furthermore, there exists a well-formed planning research community, where they constantly improve heuristic search algorithms in order to make planning possible in real-time.

While rule based approaches such as CLIPS (see Section 2.7.8) respond better in terms of scalability (they can handle complex situations), they might lead to sub-optimal behavior of the system. On the other hand, task planning can provide optimal results on small problems, but suffers from large search spaces (does not scale well) on larger problems.

4.1 International planning competition

As mentioned in Section 2.1, the International Planning Competition (IPC) has become a milestone in planning where several state of the art planners are compared and shared with the planning community. The decision was made to search for a planner within this competition as it offers open source implementations of several state of the art planners among with a research paper describing their approach.

RoboCup @Work competition is based on a deterministic environment, therefore the planner needed for our framework is found in deterministic planning IPC branch, the biggest part of IPC.

Our work in this thesis is based on the sequential satisfactory category of the IPC deterministic branch, as mentioned in 2.1.3, sequential satisficing refers to the ability of the planner to produce quick plans of reasonable quality based on heuristic search to prune the search space. This feature makes satisficing planning suitable for robotic applications where a rapid response is expected from the system.

Because satisficing planning is in general sub-optimal, experiments have been conducted (in chapter 5) to further investigate on the quality of the produced plans for our domain.

4.2 The Planner

The results from the last IPC in 2014 are compared via quality score and are displayed in table 4.1 (some domains were omitted for space reasons, but if needed full table can be consulted here[1]).

[1] `https://helios.hud.ac.uk/scommv/IPC-14/repository/results.tar`

	Tetris	Barman	Cave	Childsnack	Citycar	Hiking	Maintenance	Parking	Transport	Visitall	Total
ibacop2	4.02	16.38	7	14.98	6.95	18.04	16.71	5.28	7.01	13.32	166.21
ibacop	6.28	16.27	7	15.29	7.32	18.65	16.81	1.74	9.94	14.18	162.73
mercury	14.38	13.94	3	0	3.99	16.46	5.06	15.64	20	19.88	153.04
miplan	7.46	16.54	7	18.22	4.69	18.14	16.62	11.08	0	8.18	150
jasper	9.52	19.78	8	0	8.89	17.19	9.28	12.91	7.59	15.18	144.89
uniform	11.52	17.83	7	1.23	12.74	17.01	9.36	9.74	9.25	19.56	143.25
cedalion	3.2	16.85	7	0.71	7.71	18.74	14.15	4.29	5.14	19.49	137.34
arvandherd	14.63	18.12	7	5.57	19.39	19	13.17	0.69	4.53	0.68	137.1
fdss-2014	9.87	11.64	7	1.36	5	17.44	16.63	10.86	4.06	8.78	127.89
dpmplan	1.8	16.57	7	18.46	5.82	16.28	15.07	0	0	3.41	125.5
use	3.76	15.12	0	0	1.65	8.79	15.03	3.65	6.27	14.8	107.14
nucelar	6.6	15.89	7	3.66	7.49	17.24	16.63	1.41	0	3.38	101.44
rpt	9.41	0	3	2.98	3.11	17.35	10.82	4.7	3.57	0.06	98.25
bfs-f	11.03	8.89	7.94	0	0	2	7.72	19.94	3.57	18.99	96.11
bifd	8.67	0.62	3	1.16	3.16	15.25	7.83	5.38	3.52	0	86.99
dae yahsp	0	0.35	0	9.36	0	8.12	0	0	9.02	17.26	64.19
freelunch	3.74	0	3	17.43	0	1.91	15.26	0	0.02	2.51	61.21
yahsp3-mt	0.67	6.6	0	0	0	3.81	0	1.37	10.74	10.73	58.5
yahsp3	0.54	1.33	0	0	0	4.8	0	0	8.03	17.08	48.11
planets	0	0	5.97	0	3.96	0.77	0	0	0	0	24.95

TABLE 4.1: Quality score sequential satisfactory IPC 2014 results

Results from last IPC 2014 (table 4.1) show that the best planner is ibacop2 and ibacop (with ibacop2 being a variation of ibacop). We observe that the presented domains offer plenty of diversity, from the Tetris domain (stacking pieces of a puzzle), to the Barman (serving drinks). The most similar domain to the task we want to perform in this thesis (insertion and transportation tasks) is the transport domain.

In the transport domain, a truck needs to deliver packages between cities which are connected to each other by roads, although not exactly the same but in our test (basic transportation test) the robot is asked to transport objects between different service areas in a fully connected environment (no road concept is needed).

The best planner available in the transport domain in the sequential satisfactory category of IPC 2014 was Mercury planner[12]. It had a score of 20 points, which is almost double the points of that it's nearest competitor, yahsp3 with 10.73 points (see Table 4.1).

We therefore select Mercury planner[12] being the most suitable planner for transportation tasks. More about the inner workings of the Mercury planner can be consulted in Section 2.3, the following section discusses its integration with the robot.

4.3 ROSPlan limitations

As described in Section 2.5, ROSPlan provides an open source planning framework implementation. We decided to reuse some of their components, such as the knowledge base, and parts of the planning problem generator component.

We have identified two main limitations in their approach, first being related to the lack of support for cost information, and second, is the way the main planning loop is implemented.

They have defined a fixed re-planning strategy based on mismatch between the observed world (monitoring) and internal knowledge representation of the robot (knowledge base), however this strategy cannot be changed and is hard-coded deep in the planning system.

The ROSPlan framework divides the functionality in two nodes: the knowledge base and the planning system. To encourage code re-usability, components need to be split, for example plan validation component should be able to be reused without having the whole system or a different re-planning strategy should be able to be implemented.

For the reasons above, we decided to implement our own planning framework. This framework reuses the knowledge base from ROSPlan, as well as, a knowledge base rqt

visualization tool and the planning problem generator class. The planning problem generator is extended to include costs.

4.4 Component description

In this section we describe the individual planning components used for the integration of the selected planner into the existing robot architecture.

Referee box parser - Receives a task specification coming from Referee box as a string (see example in Listing 3.1) and converts this information into objects and facts which describe the initial state of the world, this information gets uploaded to the knowledge base. This also includes the desired state of the world - for example, where objects should be transported to.

Knowledge Base - It is used to store objects, facts and goals. It's original version was extended with a Knowledge uploader and a Knowledge base analyzer. The Knowledge uploader reads a PDDL problem definition from file and uploads all information in it to the knowledge base. The Knowledge base analyzer answers two questions: "are there unfinished goals in the knowledge base?" and "is there new knowledge?" This is done through the event_out topic.

World model - This component stores information about the environment in the form of different files (see Figure 4.1), the domain model contains all available operators and expresses the physics of the system (rules of the world). The basic facts are information about our robot in particular, for example that it has one gripper or that it can only store three objects at the same time in the rear platform. The costs file gathers all available action cost information, for now we only have navigation costs but in future more costs can be added, for example perception costs. Locations file contains pre-recorded navigation points of interest as x, y, theta coordinates with friendly names, for example S1, S2, START and EXIT. The map, stored as a .pgm image file, is the end product of SLAM process which contains geometric information about the world, stored in a occupancy grid matrix. The plan metric (also stored in world model) refers to the strategy that the planner will follow during search. For our case the metric is to minimize the total cost. An example of all World model files can be seen in appendix B.

Path length calculator - Receives a navigation path (array of poses, see Figure 3.4) between two locations, generated by the global planner (see Section 3.2.2) and calculates the path length, which is the sum of the Euclidean distance between all poses.

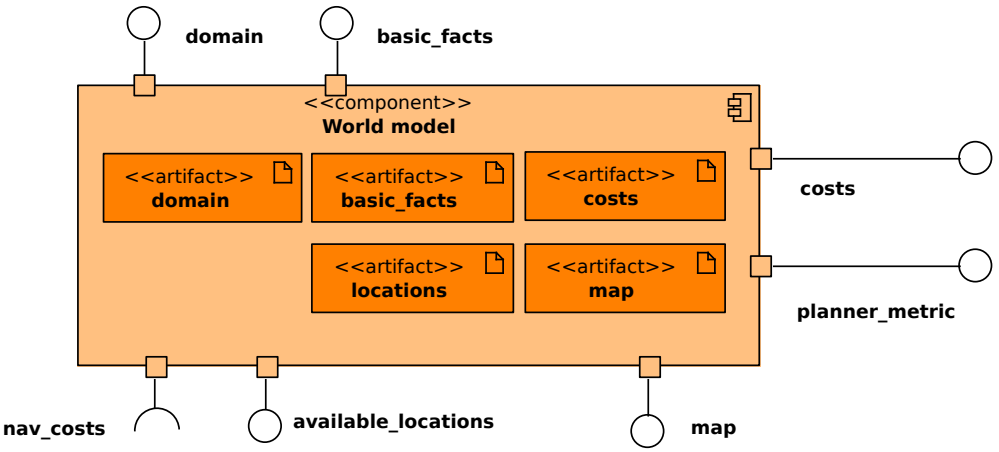


FIGURE 4.1: World model component diagram.

Navigation cost generator - This component takes as input the available recorded locations from the World model (points of interest) and produces a file containing all path lengths between known locations in the environment. To obtain this information, the global planner (see Section 3.2.2) gets called several times to produce paths between known locations (see Figure 3.4) and triggers the path length calculator to receive the path length. The interaction between the above mentioned components is depicted in Figure 4.2.

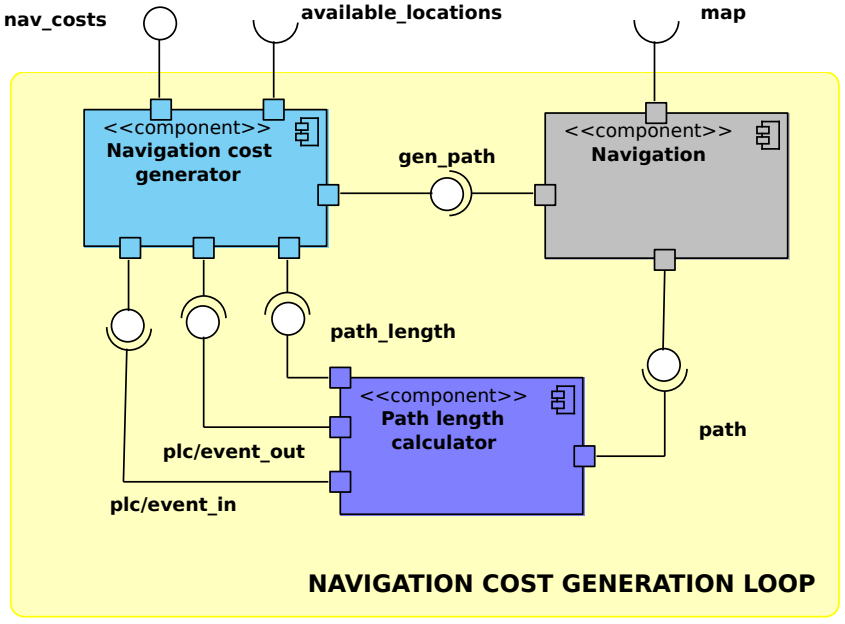


FIGURE 4.2: Component diagram for navigation costs calculation.

Planning problem generator - This component generates automatically a PDDL planning problem definition from a knowledge base snapshot. It takes as input objects, facts, goals, costs and plan metric and generates a problem in PDDL version 3.1. The vast

majority of the code of this component was taken from [18] but it was extended to support costs.

Planner - Although Mercury planner[12] was selected, the interface is designed so that any planner from IPC can be used. The planner component receives the domain and a planning problem and generates a plan.

Plan validation - Using VAL [32], the automatic planning validation tool used in IPC the plan is tested to confirm if it solves a planning problem. To make the validation possible it takes as input the domain model, planning problem and the plan and produces as output a binary result stating whether if the plan solves the proposed goals.

Plan parser - The plan parser is compatible only with IPC planners, however if another parser is needed in future the user can make a new parser and insert it easily in the planning architecture, which is flexible and configurable. The function of the plan parser is to read the plan (file) and convert it into a vector of actions.

Scheduler - ROS actionlib[2] provides a standardized interface to communicate with preemptable tasks. Shown in Figure 4.3, the scheduler receives an action vector and iterates over the received actions one by one, creates action clients with arguments from the plan and calls an action lib interface that executes each action. If any of the action reports failure, the execution is stopped and a binary flag, indicating failure, is triggered.

Based on the outcome of the executed actions the scheduler updates the knowledge by adding or removing predicates accordingly.

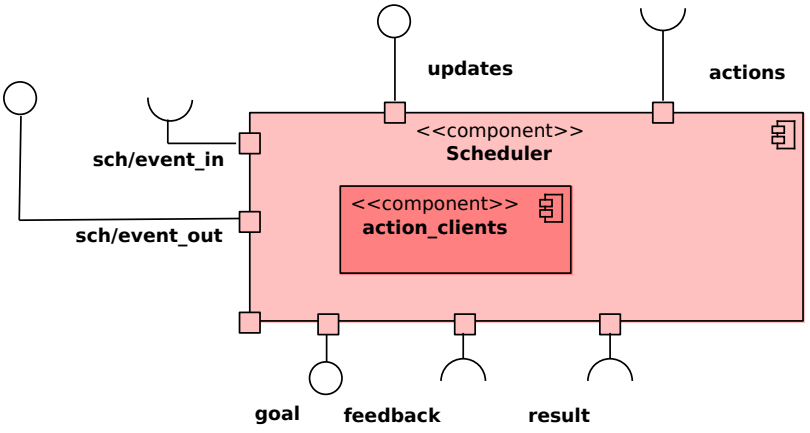


FIGURE 4.3: Component diagram for scheduler.

[2] `http://wiki.ros.org/actionlib`

4.5 Domain model and action servers

The planning domain model was designed based on the previous existing basic transportation test (BTT) state machine (see Section 3.2.5). It was done by refactoring or splitting the previously big state machine into small, reusable and easy to maintain state machines which execute high level robot actions.

The complete domain definition in PDDL 3.1 format can be seen in Appendix B.1 for basic transportation tasks, the insert operator is presented as an extension of the transportation domain in Appendix B.2 as a proof on concept that demonstrates that is simple to add new functionality from the planning point of view (an action server needs to be also created).

The available actions in the combined (transportation with insertion) domain are:

- move base safe
- perceive location
- pick
- stage
- unstage
- place
- insert

By performing this actions the robot is able to accomplish all transportation and insertion tasks that are required for competition. New actions can be added on top of this existing ones to extend the work presented in this thesis such as stack objects or paint objects, etc.

All actions are implemented in SMACH[31] state machines wrapped around action lib[3].

In the ROS actionlib package, a goal message is specified with specific arguments and a feedback is received periodically. When the action execution is finished it provides a custom result that can be of any data type or struct. In our case the provided result is a binary *failed* or *succeeded* response along with the predicates that need to be updated in the knowledge base to synchronize the robot belief with changes in the real world.

Robot actions are described below with in detail.

[3] `http://wiki.ros.org/actionlib`

4.5.1 Move base safe action server

This action starts by moving the arm within base boundaries to avoid arm collision, then fetches destination from the generated plan and calls the navigation component to reach destination with an increased linear and angular tolerance. A direct controller is used to fine adjust the base towards the final goal and fail or success is send to action lib interface (see Figure 4.4).

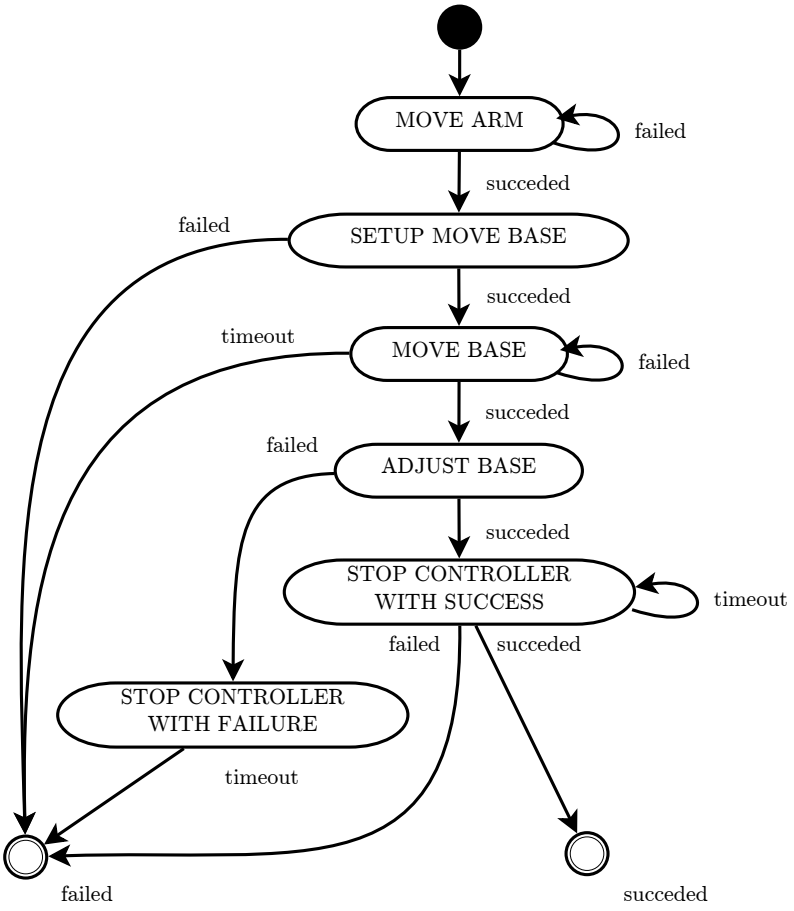


FIGURE 4.4: Move base safe action state machine diagram.

4.5.2 Perceive location action server

First moves the arm to look at workspace joint configuration (see Figure 3.12) to capture the scene in the field of view of the arm mounted camera, then triggers object recognition pipeline (see section 3.2.3) to identify the objects on the platform. Finally a list of recognized objects gets generated and is stored by a component called object selector. The object selector has memory and stores previously recognized objects but upon request, it can also forget about them. The object selector is also used by the pick action to get the pose of the object.

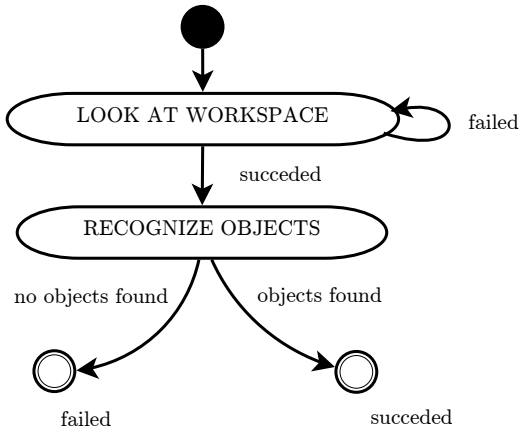


FIGURE 4.5: Perceive location action state machine diagram.

4.5.3 Pick object action server

The pick action starts by fetching the object to be picked from the plan, then opens the gripper and a query is sent to the object selector component to get the pose of the object. If the object was not previously identified by perception pipeline, a failure will occur and the execution of the action is aborted. However if the object is in the list, the object selector will publish the requested object pose. Using the pregrasp planner (see 3.2.4) the manipulation component will sample poses around the object pose until it finds an inverse kinematic (IK) solution for one of them. If IK is available, then arm motion planning is called to generate a path between initial and final joint configuration. Afterwards the generated arm plan is sent for execution, moving the manipulator towards the object. The next state is to close the gripper to grasp the object and a monitoring process confirms if the gripper was indeed closed by reading the angular joint values from the gripper fingers. Next step is to move the arm to a intermediate pose called "HOLD" and finally the grasp monitor (see Section 3.13) confirms if the gripper has an object in the gripper. The state machine diagram for pick action can be seen below in Figure 4.6

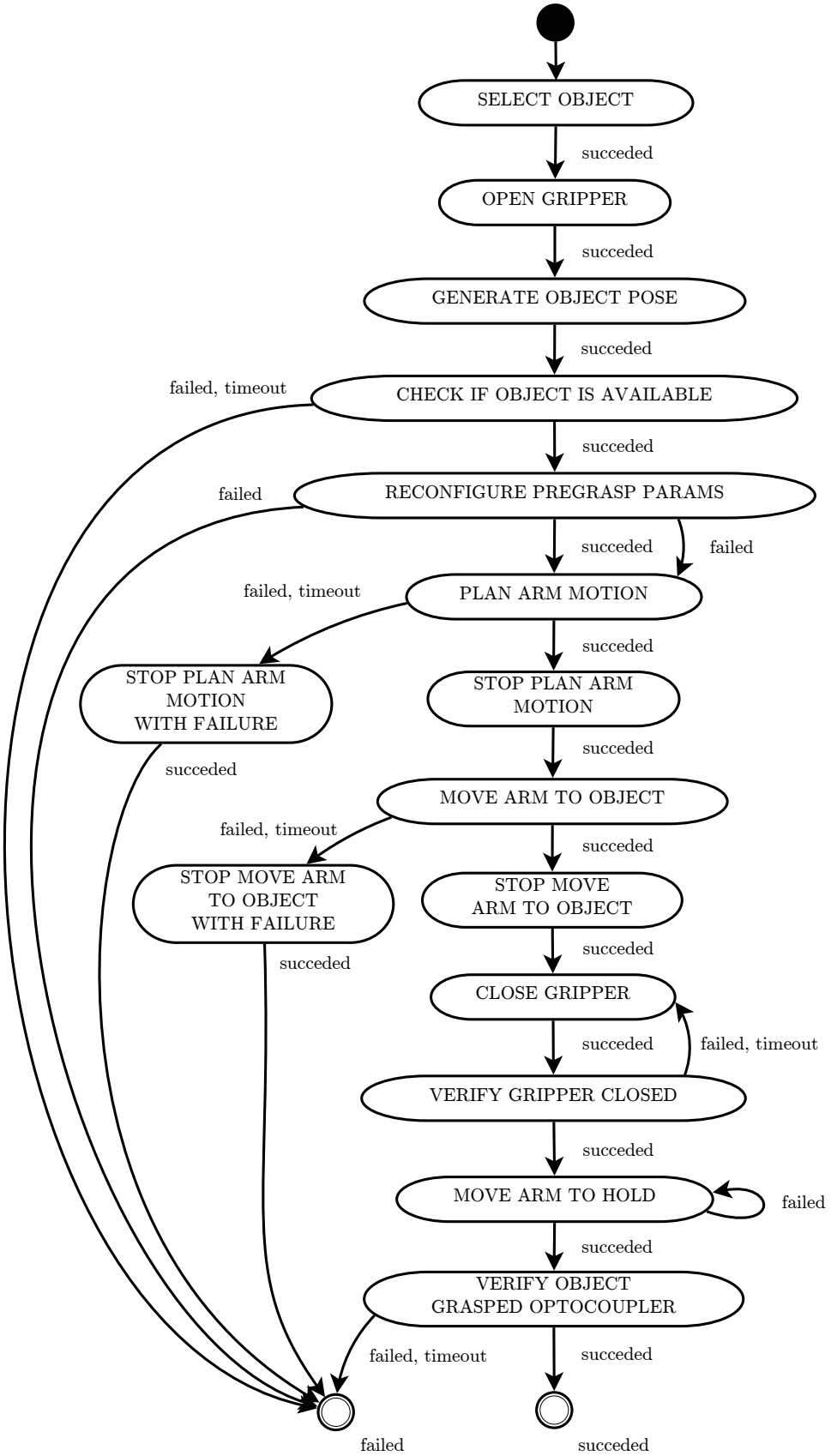


FIGURE 4.6: Pick object action state machine diagram.

4.5.4 Stage object action server

The stage object action assumes that the gripper is holding an object which needs to be stored in the rear robot platform. It first moves the arm to pre stage arm joint configuration, for example: pre_platform_middle, pre_platform_left, pre_platform_right (see Figure 3.12), information which is taken from the plan. Next state moves the arm to stage configuration, for example: platform_middle, platform_left or platform_right and opens the gripper, it finishes by moving the manipulator to a intermediate joint configuration called "HOLD" (see arm configurations in Figure 3.12).

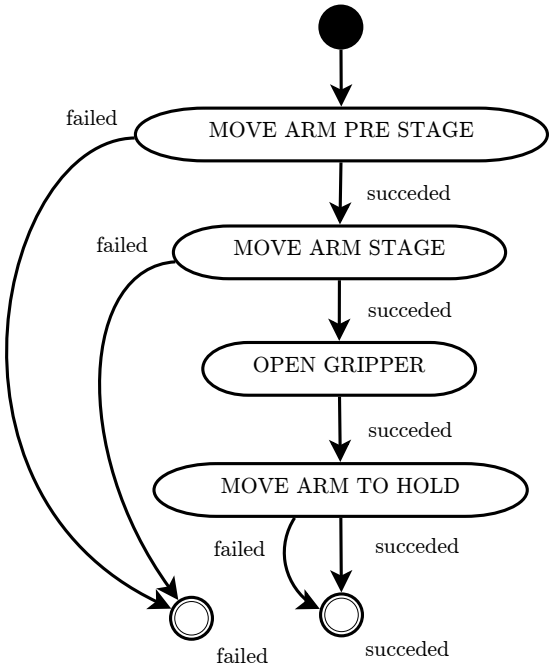


FIGURE 4.7: Stage object action state machine diagram.

4.5.5 Unstage object action server

Opposite to stage object action (see Section 4.5.4) unstage object action assumes that the gripper is free and that the robot has at least one stored object in it's rear platform, it then unstores the object, holding it for a while, waiting for a place, insert or stage action to occur. Unstage action state machine is shown in Figure 4.8

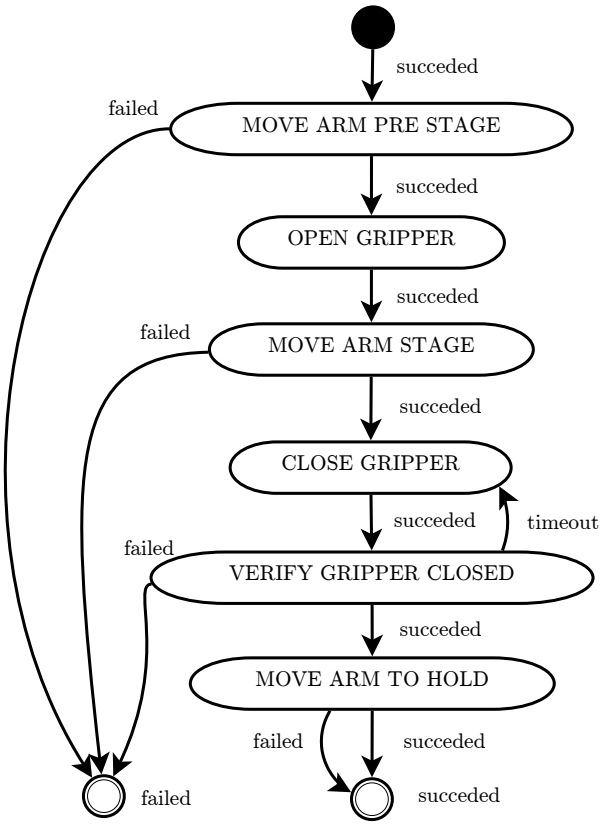


FIGURE 4.8: Unstage action state machine diagram.

4.5.6 Place object action server

The place object action as for now is simple, but is expected to change in near future. It works by moving the arm near the place location (such as a platform), then open gripper and moving the arm back to "HOLD" configuration.

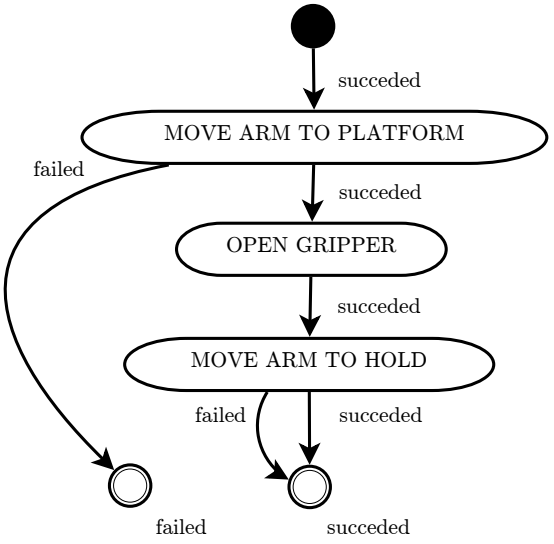


FIGURE 4.9: Place object action state machine diagram.

4.5.7 Insert object action server

Insert object action assumes that the robot is located in front of a service area with an already perceived container on its surface and that the object to be inserted is in the gripper. The action starts with a query to the object selector to generate a pose for the center of the container, this pose gets shifted by a fixed offset to generate a "TOP HOLE" pose, the manipulator then creates a motion arm plan to reach the pose on top of the container and moves the arm towards it. Next step executes a linear motion down and opens gripper, finally a linear motion up gets executed and arm returns to "HOLD" position. State machine for insert object server can be seen in Figure 4.10.

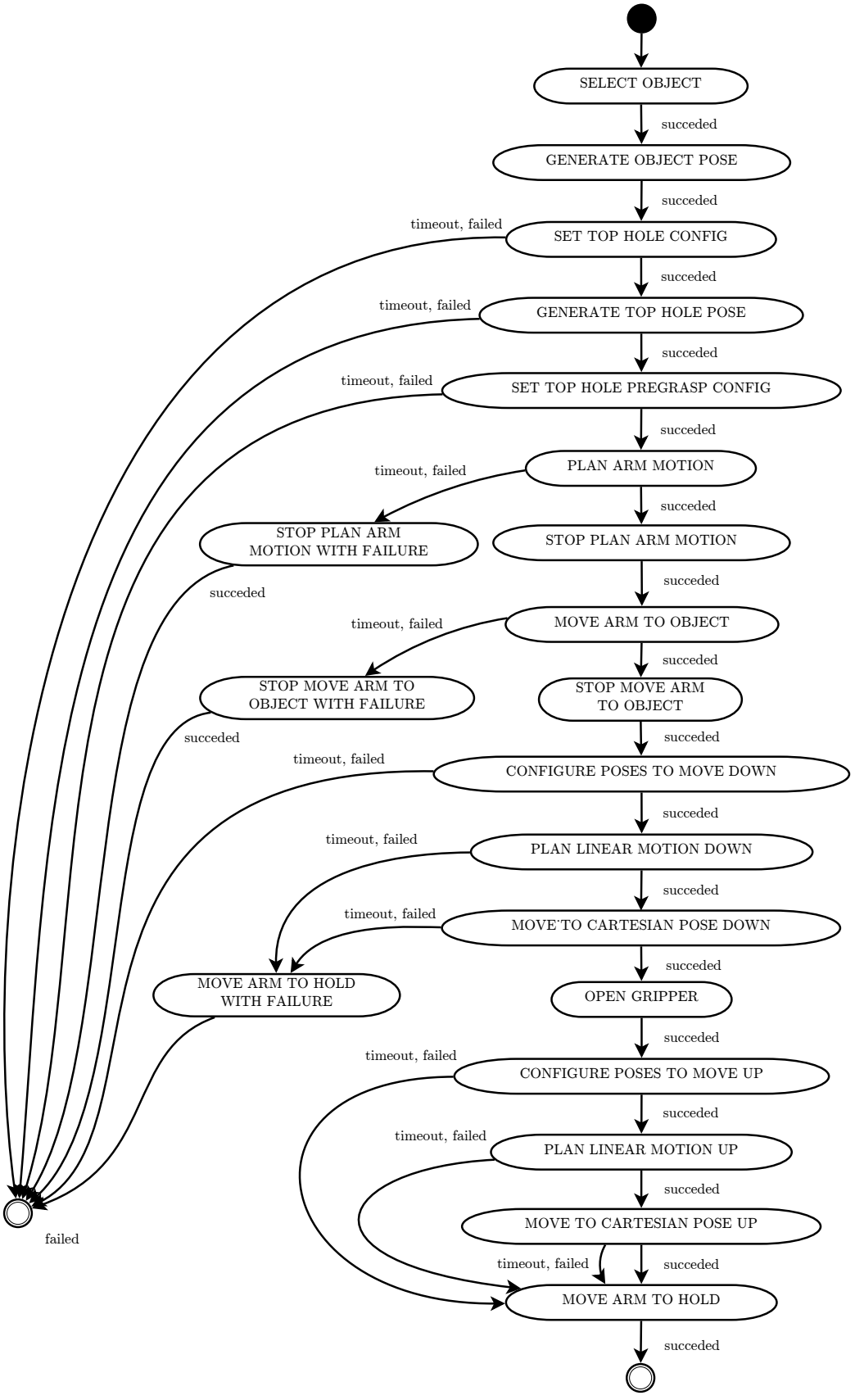


FIGURE 4.10: Insert object action state machine diagram.

4.6 Planning architecture

A component diagram overview (Figure 4.11) of the architecture shows the interaction between previously described components (in Section 4.4). Coordination between all components is controlled by the planning coordinator state machine, which is described below in Section 4.6.1.

4.6.1 Planning coordinator

The planning coordinator controls the behavior of the robot and the planning loop. It starts by sending a request to Knowledge uploader component to upload basic facts to the knowledge base, then, using information from the KB, the KB analyser determines if there are unfinished goals and new knowledge available, otherwise it waits. However if goals are available, for example published by Referebox or in general coming from any other component (speech recognition, etc), it attempts to achieve them. Next step is to generate a PDDL planning problem definition from knowledge base snapshot. This problem is the combination of basic facts, cost information, specified metric and initial state.

The planner tries to generate a plan, based on the domain definition and problem. If it succeeds, it returns a plan, otherwise it does not generate a plan file. The inclusion of the plan validator component (VAL) [32] is to ensure that the generated plan solves the required goals and that no previous plans are taken into account.

Once the plan is confirmed to be valid it gets parsed into a vector of actions which are sent for execution to the scheduler. The planning coordinator waits until the scheduler reports execution success or failure and if action lib reports action failure then re-planning is triggered and a new problem is generated. If execution of the plan is complete with success then the robot waits for more goals. The state machine flow can be matched with component diagram from Figure 4.11.

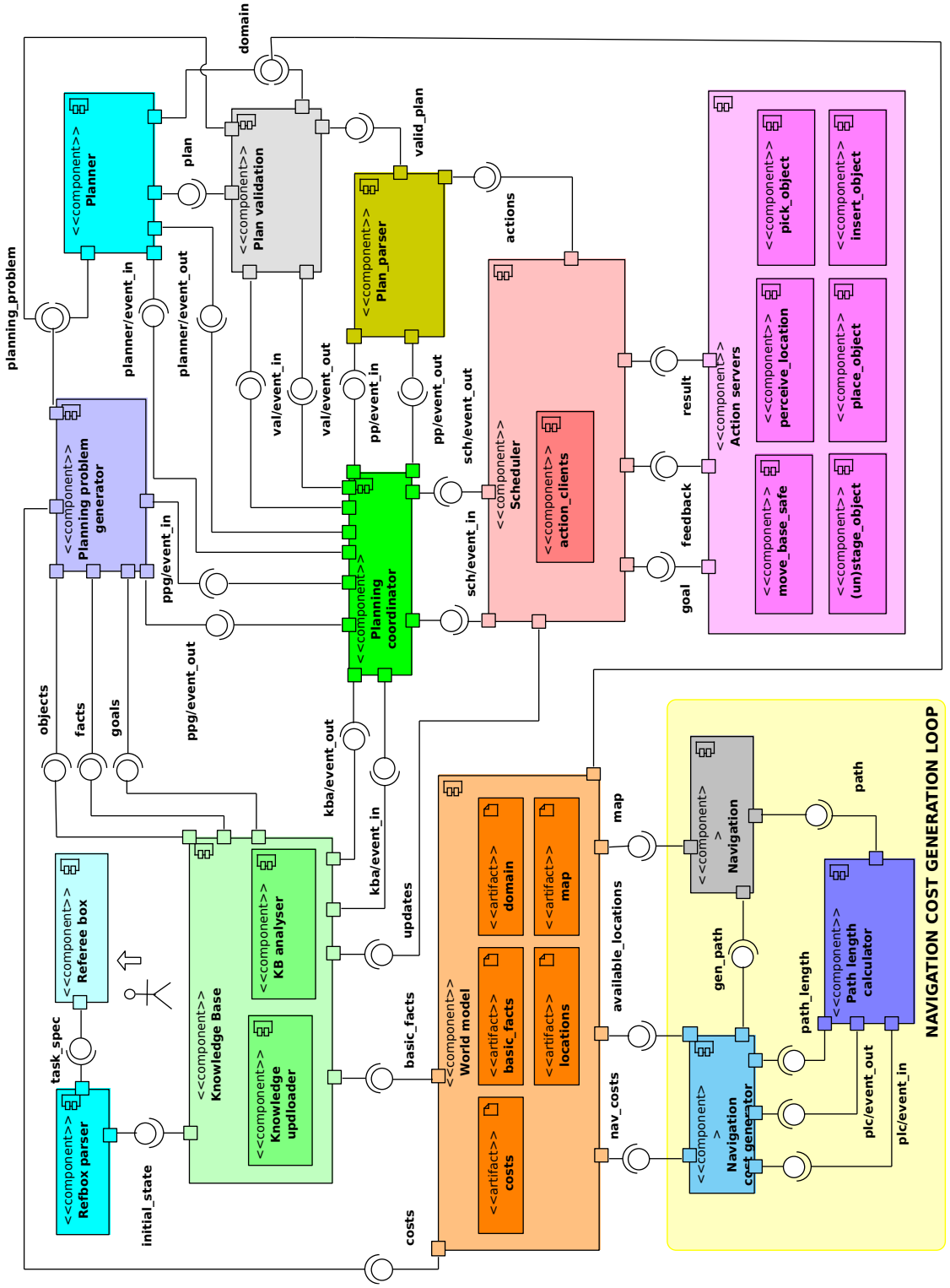


FIGURE 4.11: An overview of the system architecture showing the specifics of the planning framework.

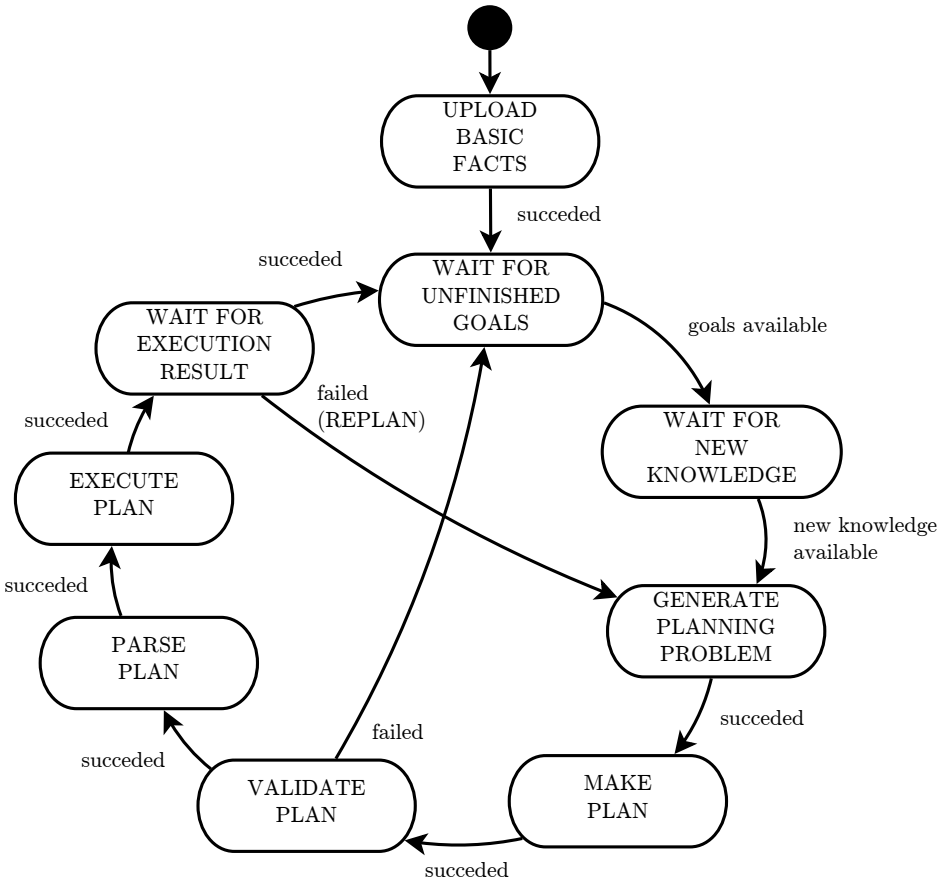


Figure 4.12: Planning coordinator state machine.

Chapter 5

Experimental Evaluation

Chapter 4 presented the planning framework approach and architecture overview, this chapter tests the approach on a real robot and presents different experiments regarding the Mercury planner capabilities.

5.1 Mercury planner experiments

5.1.1 Experiment setup

The environment setup is based on that of the arena at the Robocup@Work last competition 2015[1] (see Figure 5.1).

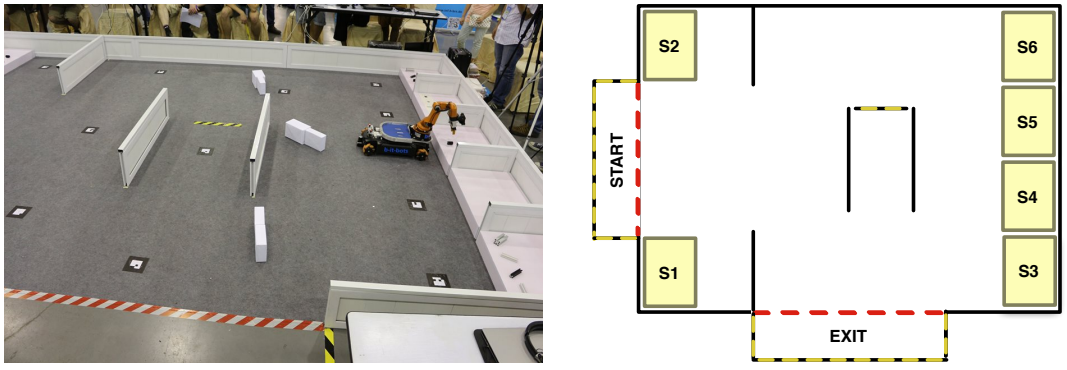


FIGURE 5.1: Real and simplified test environment, based on last RoboCup @Work competition.

[1] http://www.robocup2015.org/

The arena has start and exit locations, delimited by colored tape and six service areas or platforms labeled S1 to S6 which can contain industrial-related objects such as aluminium profiles, bolts, nuts, etc. As a simplification we consider object names to be in the form of: o1, o2, o3, etc.

The computer on which the experiments where run (the robot PC) had at the time of the experiments an i5 intel processor with 2 GB of RAM.

All files required to reproduce the planner experiments are available online[2], and most important parts are presented in the Appendix. Slightly different outcomes might be produced while running the experiments depending on the computer which is used. This is mainly because of the computational power that is available for the planner to search but also because a timeout of 1 minute was given to the planner to produce a solution.

Automatic scripts were used for the robot to create solutions for different problems in a 26 hour experiment run.

A transportation domain was designed to conduct our experiments, it can be used for transporting objects between different service areas, the code can be consulted in Appendix B.1.

The planning framework was tested on the KUKA youBot successfully for at least three months in the lab and has proven to be a working solution.

5.1.2 Mercury planner parameters

The planner can be configured with different search parameters.

For simplicity we define the following parameter names.

	cost type	lm cost type	w
parameter set 1	1	1	1
parameter set 2	1	1	2
parameter set 3	1	1	3
parameter set 4	1	1	4
parameter set 5	1	1	5
parameter set 6	2	2	1
parameter set 7	2	2	2
parameter set 8	2	2	3
parameter set 9	2	2	4
parameter set 10	2	2	5

TABLE 5.1: 10 different search parameters were used for the experiments.

[2] `https://github.com/oscar-lima/mercury_planner_experiments`

- w - heuristic weight
- lm cost type: NORMAL, ONE, PLUSONE - landmark action cost adjustment
- cost type: NORMAL, ONE, PLUSONE - operator cost adjustment type

The 10 different selected parameter combinations for Mercury planner were based on what original authors used in their own planning scripts.

5.1.3 Navigation costs

Navigation cost information was given to the planner related to distances between locations. Assuming that all locations are connected with each other (which is our case), a distance matrix can be formed, such as the one shown in table 5.2. The lower triangle of the distance matrix is the result of the path length calculation between locations and the upper triangle is a integer parsing of the lower part. This is because Mercury planner in particular does not support floating point costs, however other planners such as FF[14] do support floating point costs.

	START	S1	S2	S3	S4	S5	S6
START	X	2	3	6	6	6	6
S1	2.4995	X	3	6	6	5	6
S2	2.5352	3.1637	X	6	6	5	5
S3	6.234	5.6923	5.9866	X	1	2	3
S4	6.2278	5.971	5.701	1.0066	X	1	2
S5	5.7254	5.4699	5.1964	2.028	1.0206	X	1
S6	5.836	5.5775	5.2838	3.1154	2.1078	1.0854	X

TABLE 5.2: Distance matrix for experimental test arena.

An offset version of the calculated distance matrix is then transformed into PDDL format (see Appendix B.5) and is included in the problem definition file (problem.pddl). The offset value is comming from specific domain experiments to determine the best numeric cost range for the planner to perform the search (see section 5.2).

The robot is expected to prefer locations which are closer (or cheaper). If we look at problem 3 (3 objects to transport, see Appendix B.5) results (see Listing 5.1) we notice that the robot indeed first went to pick objects from closer platforms. Initial state for problem 3 is: (on o1 S1)(on o2 S4)(on o3 S2).

```
1 (move_base_safe start s1 youbot-brsu-5 dynamixel)
2 (perceive_location s1 youbot-brsu-5 dynamixel)
3 (pick o1 s1 youbot-brsu-5 dynamixel)
4 (stage o1 platform_left dynamixel)
5 (move_base_safe s1 s2 youbot-brsu-5 dynamixel)
6 (perceive_location s2 youbot-brsu-5 dynamixel)
7 (pick o3 s2 youbot-brsu-5 dynamixel)
8 (stage o3 platform_middle dynamixel)
9 (move_base_safe s2 s4 youbot-brsu-5 dynamixel)
10 (perceive_location s4 youbot-brsu-5 dynamixel)
11 (pick o2 s4 youbot-brsu-5 dynamixel)
12 (stage o2 platform_right dynamixel)
13 (move_base_safe s4 s5 youbot-brsu-5 dynamixel)
14 (unstage o1 platform_left dynamixel)
15 (place o1 s5 youbot-brsu-5 dynamixel)
16 (unstage o2 platform_right dynamixel)
17 (place o2 s5 youbot-brsu-5 dynamixel)
18 (unstage o3 platform_middle dynamixel)
19 (place o3 s5 youbot-brsu-5 dynamixel)
```

Listing 5.1: Generated plan for transporting three objects

According to distance matrix (table 5.2), the closest location from start position is S1 with a cost of 2 and the generated plan (listing 5.1) line 1, commands the robot to first go to S1. In subsequent steps, the robot perceives the location, picks and stages o1 and then goes from S1 to S2. S2 has the closest distance from S1, so we verify that the planner is doing the work properly so far. Finally it goes to pick the last object from platform S4 and delivers all objects to S5. Note that all available robot platforms were used (platform middle, left and right), and the result is optimal (for parameter set 1).

5.1.4 Scalability test

For scalability, this experiment tests the number of objects that can be transported at one time within one minute of planning time.

A total of 25 planning problems were tested, where the complexity of the problem scales with the number of objects, from 1 object in the first problem to 25 objects in the last problem.

Scalability test results are shown in Figure 5.2, as expected, the harder the problem, the more planning time is needed.

The results show that as the number of objects increases, planning time increases exponentially while plan length increases linearly.

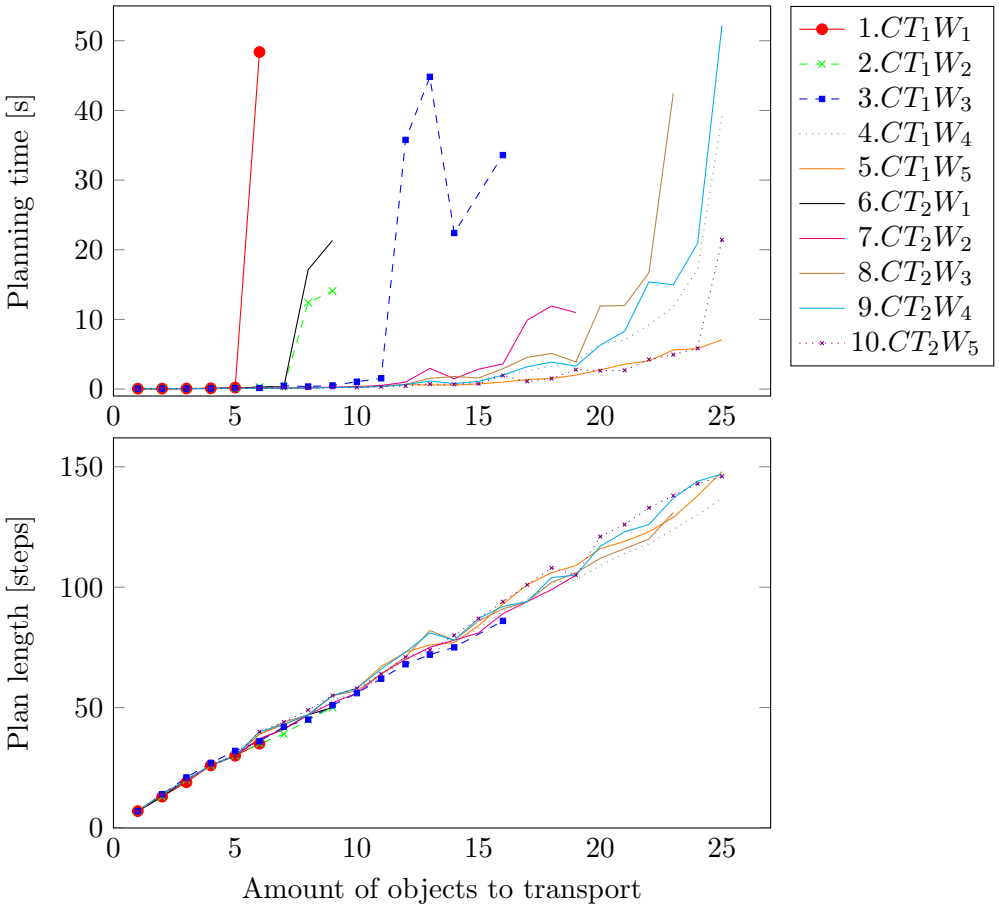


FIGURE 5.2: As problem increases in complexity, so does the planning time, 10 different parameter have been used.

We observe from Figure 5.2 that depending on the search parameters we get different behavior of the planner. There is a trade-off between plan quality and planning time. The best quality was achieved by parameter set 1, however it was only able to solve easy problems (1-5 objects to transport) when restricted to perform in less than one minute.

The best planning time belongs to parameter set 5 (cost type 1, heuristic weight 5), but shows poor performance in terms of plan length. The suggestion would be to look at the problem complexity (for example, amount of objects to be transported) and choose the parameter set accordingly.

The largest generated plan was of 148 steps, non optimal, was generated in 7 seconds using parameter set 5.

5.2 Redundancy test and finding a suitable cost range

The planner was configured with a metric designed to minimize the sum of all the action costs when generating a plan. Due to the heuristics that it uses, the assigned operator costs greatly impact its performance. For simple domains, it is able to generate good quality plans but when uniform cost is assigned or the problem instance grows the planner's behavior is erratic in that it will sometimes produce redundant plans (plans which include actions which can be removed without affecting the outcome).

To test when such redundancies appear, the transportation domain extended with insertion actions was used, but a more complex problem instance was given. The problem specification was to transport nine objects and insert four of them into a container. There were 15 objects in the knowledge base, eight locations, one robot, and free space on its platform which could be used to carry three objects at a time. The cost of the navigation action was then varied from 1 to 1,000 to test if the redundant actions appear when all actions have equal cost.

The results confirmed that uniform action costs produced two redundancies corresponding to four actions and a plan of 63 steps which took 1.5 seconds to generate. When navigation actions were assigned a cost of two (and all remaining actions were assigned a cost of one), the redundancies disappeared and plan length was further reduced to 56 - 57 steps, but returned again for costs ≥205. This coincides with a dramatic increase in planning time of up to 12 minutes (see Figure 5.3). The cost values yielding the shortest plan length (56) range from four to eleven.

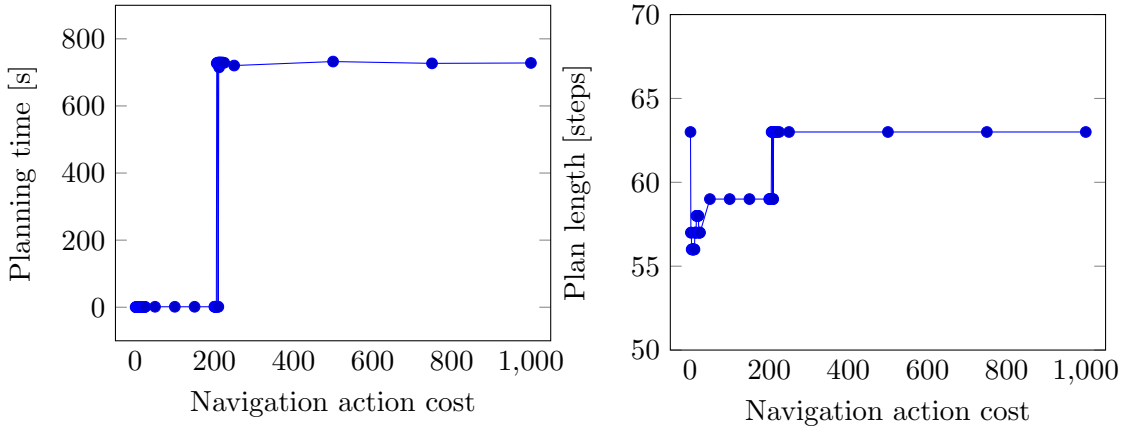


FIGURE 5.3: The impact of increasing navigation cost on planning time and plan length.

Interesting is that further cost experiments with a simpler only transportation domain and problem (5 objects to transport) show that cost information does not influence plan length nor planning time. This is a clear indication that experiments have to be conducted to find the particular cost value which will be suitable for a specific domain.

Chapter 6

Conclusions

Many roboticists today enable intelligent behavior in their robots via finite state machines. This work presents a working example of the integration of a planner into a real robot platform by re-using previous existing state machines refactored into modular actions. We hope this work to be useful and to encourage the use of task planning as standard decision making engine in robotics.

6.1 Lessons learned

Task planning has many advantages compared to finite state machines, for example its structure is based on first order logic, which is something that we humans are very much used to. Modelling preconditions and effects can be quickly learned to make use of existing planners.

One of the challenges of task integration is that the programmer needs to be familiar with all components of the robot, navigation, perception, manipulation, task planning, state machines, etc. In order to perform such task integration the programmer needs to be experienced with the platform and have a good overview knowledge of all components present on the robot. This it is not easy to achieve and would be different depending on the robot, for example task integration on the Care-O-Bot or PR2 robot might be more challenging than with the youBot, as we think that definitely those robots are far more complex.

Another lesson learned in the process is related to the knowledge base. During our tests on the robot we have noticed a lot of failure in the robot actions, specially in the perception component; sometimes it recognizes some of the objects and sometimes misclassifies objects. For our case the knowledge coming from the Referee box can be

considered as ground truth, objects will be located where the referee says and if the robot is not able to identify them then it can be concluded that the robot failed at perceiving the objects, and re-planning is needed. Nevertheless in other domains, exploration processes could be needed, where the robot is unaware of the objects location and generally speaking if the robot failed at perceiving the environment the knowledge base will be updated with wrong information. We have learned that the user, as expert of the system and through a process of continuous tests needs to fine tune the information that gets written into the KB. Maintenance to KB is critical for the system to be robust against failure.

Monitoring systems play a crucial role in planning. For example, when a robot fails to grasp a perceived object, the grasp monitor provides with valuable information about the failure of the executed action. This, in turn, can trigger a recovery mechanism, like retrying the action or report failure, to trigger re-planning and executing the action again.

In our case, a grasp monitor existed but it was insufficient. Originally based on current feedback coming from the gripper motors, this approach worked for big objects only, which for sure will create a bigger effort on the gripper motors but will fail for thin objects, specially if the fingers are made of a flexible material. An optic based grasp monitor would be better for this case but fails if the object has holes on it (imagine a thin plate with a hole in the center). Furthermore our depth camera is mounted on the gripper and works by emitting a pattern of infrared light which affects the optic based gripper, so a low pass filter was designed to solve the problem. Although our robot has a simple two finger gripper, it can get complicated to develop a robust end effector with a reliable grasp monitor which in our opinion must be performed by sensor fusion with data coming from many sensors and conditions.

Sometimes we don't know the result of an action failure, for example fail to grasp could mean the object fell on the floor, but we don't know that. Then it becomes impossible to update the KB. Therefore the importance of monitoring components that need to observe the World and update KB accordingly.

One way to evaluate complexity of a state machine is performed by counting the nodes and edges of a graph. For BTT we have refactored a big state machine of 47 nodes and 81 edges into 7 state machines (move base safe, perceive location, pick object, stage, unstage, place object and planning coordinator) of 45 nodes and 81 edges combined. We do not claim accuracy on this test because the state machines are slightly different, for example visual servoing is used in the original BTT while new actions do not. Nevertheless it is presented to the reader as an interesting analysis.

We have observed that during the development of the state machines that form the actions of the robot, testing needs to be constantly done in the real robot. When having a big state machine there is the need to develop mockups to bypass parts of the state machine which have been previously tested and those where the programmer feels confident about the correct performance of the robot. For example you might want to bypass navigation part of the state machine and try perception + manipulation, etc. Thanks to the refactoring no more mockups have been used and functionality is now tested individually.

Another advantage is the lack data being transferred between the states as it was previously the situation. For example, in BTT state machine a task list data structure was transmitted as input data trough all the states. In planning, information about the world is stored in the knowledge base as predicates.

Action clients have been designed based on a task planning approach and are ready to be integrated with most planners as they follow a logic structure, however the action servers are not of exclusive use of the scheduler and can be called for example from another state machine or by the user, this encourages re-usability of the code and even if new programmers in the team do not feel comfortable using task planning they still can benefit from the work presented in this thesis.

Another disadvantage of having a big state machine, BTT for example is the fact that you could only use the robot to perform that test. In task planning the goals could be of greater diversity, for example you could specify a navigation goal of the type at(robot S2), and the robot will make a plan to execute the given command. This cannot be done by BTT state machine “per se”, a code modification would be needed.

Unfortunately we had a problem in the robot for the last two months of the thesis in which the camera was not able to perform. This did not allow for further documented tests on the real platform.

6.2 Contributions

Contributions of this work are the planning domain model and the domain analysis which enables users to decide which parametrization and cost assignment strategy yields the best results given the size of the planning problem. Demonstration of the benefits of using a task planner in this domain and the necessity to carry out domain analyses. In addition, a number of software components have been developed and improved while being tested on a real robot system.

In order for our robot to carry out its tasks efficiently, we developed a task planning framework, the software components are independent and reusable. The system is fault tolerant and it updates new knowledge every time it performs an action (success or failed action) to re-plan when needed.

By using an automated planner for decision-making, the robot's performance in the @Work setting was improved. Moreover, by refactoring the SMACH state machines now used for execution, re-usability, readability as well as maintainability were enhanced. With the introduction of new SMACH states to monitor the outcome of actions, it is possible to re-plan in order to handle unexpected action outcomes. We investigated the various strategies for assigning costs to actions. The results showed that mercury planner heuristic cost type 1, heuristic weight 1 yields the best plans, but is limited to 5 objects (for 10 seconds of planning time), however different planner parameters allow for tasks of 25 or more objects provided at the same time.

A trade-off between plan quality and plan length needs to be balanced and search parameters have to be chosen based upon the complexity of the problem.

The conclusions presented in this section have been published in a conference paper with title "Task Planning with Costs for Industrial Robotics Domains" which is in the process of being accepted in RoboCup International Symposium 2016 (July 4, 2016).

6.3 Future work

In future, we plan to investigate the adjustment of these costs to reflect the probability of recognizing objects so that those which are easier to detect can be prioritized. Adjusting the costs to reflect any differences in points being awarded for the various objects is also future work. This work has served as proof of concept which we will make use of, together with the lessons learned from modeling the domain for the basic transportation test to model the remaining @Work tasks.

This work has presented the integration of the state of the art Mercury planner from international planning competition (IPC); however more planners can be integrated and tested in future. A comparative evaluation could be performed by taking a planner from sequential optimal IPC branch, in this category the planners are given 30 min of planning time and are expected to perform a deeper search into the problem graph to return better plans, close to optimal. An idea for future would be to include such planners as ground truth information to compare different results.

In this work we have presented a simple re-planning strategy, but in future more strategies could be applied, such as the one suggested by ROSPlan[18] based on mismatch between observed and internal representation of the world (KB).

From this work derive the need to have better monitoring systems that ensure the proper maintenance of the knowledge base as a key to be robust against failure. In future more monitoring components need to be developed for the robot, for example one that performs perception on the rear platform to search for objects that might have rolled during movement of the base.

In satisficing planning we have the disadvantage of sub-optimality, but provided with an initial plan with reasonable quality, other approaches could take as input the sub-optimal plan to repair possible redundancies produced by the planner. Such approaches could be investigated and integrated in future.

The place action needs to be improved. As a proof of concept we have presented a simple three state action of moving arm towards platform open gripper and moving back the arm, but this behavior drops the object at a fixed height from the platform. A better action would first perceive the destination service area and determine a suitable place position before moving the arm to release the object. This improvement although not related with planning presents just one example of an action that currently needs improvement. However in general there are more actions in the robot that surely will be improved in the future by new team members or students from this university.

We have presented in the state of the art section one open source planning implementation (ROSPlan), this work presents an alternative, which re-uses the knowledge base and part of the PDDL problem generator from ROSPlan. In future the idea could be used to develop a Open Task Planning Library (OTPL) that helps to break the bridge between task planning and robotics, since mostly robots currently use state machines as part of their behaviors.

Code cleanup is unfinished but slowly merging into the b-it-bots repository and more domains need to be developed, as an extension of the existing one to cope with the other @Work tasks. For example, for the conveyor belt test or the precision placement test, in which objects need to be grasped from a conveyor belt while moving and insertion tasks, similar to the one presented in this work.

Finally a domain for a multi-agent system with minor modifications to the current domain can be developed, however the scheduler currently does not support more than one robot; in this centralized plan idea, which is left for future work.

Appendix A

Basic Transportation Test (BTT)

A.1 Top level BTT state machine states description

This section describes Figure 3.14 states.

INIT ROBOT - Move arm inside footprint, open gripper.

GET TASK - Wait for referee box task specification comming through the network.

SETUP TASK - Parse single string comming from referee box into task type source locations, destinations and object names.

GO AND PICK - Move robot base to a service area which contains objects, grasp them and store them into rear platform.

CHECK IF PLATFORM HAS STILL OBJECTS - Examine if rear robot platform is empty of not.

SKIP DESTINATION POSE - Robot delivered objects to one destination but remaining ones are to be transported to another location, therefore this component pops out from the list the already visited service area.

SELECT DELIVER WORKSTATION - Examine based on task list the destination of the already picked objects.

MOVE TO DESTINATION LOCATION SAFE - Move the robot base close to the destination service area (where objects are going to be delivered). This is an intermediate state used to help the navigation as a waypoint.

MOVE TO DESTINATION LOCATION - Move the robot base to the location where the objects need to be delivered.

ADJUST POSE WRT WORKSTATION AT DESTINATION - Align robot base towards the workstation or service area.

MOVE TO EXIT SAFE - Move the robot base close to the exit location. This is an intermediate state used to help the navigation as a waypoint.

MOVE TO EXIT - Move the mobile base to exit location.

GRASP OBJECT FROM PLTF - Unload an object from the rear robot platform.

PLACE OBJ IN CONFIGURATION - Places an object which is already on the gripper on the platform which is in front of the robot in a specific pattern (on a line, zig-zag, etc.)

MOVE ARM INSIDE BASE BOUNDARIES - Move the manipulator inside the robot footprint, this is needed because navigation component is 2D based and it has a fixed footprint which does not take the arm into account. Another reason is because it is safer for the arm to navigate while it is folded.

A.2 GO AND PICK sub-state machine description

This section describes Figure 3.15 states.

SELECT SOURCE SUBTASK - Receive task list from previous state and decides the next service area to process (to pick objects from).

MOVE TO SOURCE LOCATION SAFE - Calls navigation component to move close to the source service area. This is an intermediate state used to help the navigation as a waypoint.

MOVE TO SOURCE LOCATION - Move the robot base towards the source service area.

STOP ALL COMPONENTS AT START - Sends event_in: e_stop to all manipulation components.

ADJUST POSE WRT WORKSPACE AT SOURCE - Align the base towards the service area.

SELECT NEXT ARM LOOK POSE - There are currently three robot poses to look for objects, look left, look at center and look right. This accounts for the fact that some 3D cameras (for example Intel RealSense F200) are not able to fully capture the scene, therefore the robot needs to look and recognize many times.

LOOK AROUND - Move the manipulator (which holds the camera) towards left, center or right.

RECOGNIZE OBJECTS - Trigger perception pipeline to detect and recognize previously trained objects.

ACCUMULATE RECOGNIZED LISTS - Append current object list to last object list. This helps to store in memory all previously recognized objects and accumulate them into a big list.

SELECT OBJECT TO BE GRASPED - Get from task list (state machine "global" variable) the next object which needs to be grasped.

COMPUTE ARM BASE SHIFT TO OBJECT - Calculate the distance that the base needs to move to have the target object (object to be picked) in front of the manipulator. This ensures reachability.

STOP COMPUTE ARM BASE SHIFT TO OBJECT - Send event_in e_stop to the component used to calculate the the robot base shift.

MOVE ARM AND BASE TO OBJECT - Moves the manipulator to a fixed predefined pose that ensures that the object is going to be in the field of view of the camera and at the same time moves the robot base to align with the object to be picked.

STOP MOVE ARM BASE TO OBJECT - Send event_in e_stop to the component used to shift the robot base.

ALIGN WITH OBJECT LOOP - Align with the next object to be picked.

VISUAL SERVOING - Assumes that the object to be picked is the only one in the field of view of the camera and triggers a controller that moves the base and arm at the same time to align with the object.

STOP VISUAL SERVOING - Stop the fine alignment to object component.

SET VISUAL SERVOING SUCCESS - Fine alignment to object component has succeded.

OPEN GRIPPER FOR APPROACH OBJECT - Open the gripper.

APPROACH OBJECT - Moves the end effector in the manipulator in a straight line a fixed distance down towards the object.

STOP APPROACH OBJECT - Stop the linear motion of the manipulator previously triggered.

CLOSE GRIPPER - Close the gripper.

PRE PLACE OBJ ON REAR PLATFORM - Move the manipulator to a intermediate joint configuration on the rear platform (see Figure 3.1 - C, G or E).

VERIFY GRASPED - Use the grasp monitor to verify if there is an object in the gripper.

PLACE OBJ ON REAR PLATFORM - Move the manipulator to joint configuration rear robot platform. This is used to store objects (see Figure 3.1 - D, F or H).

DELETE FROM RECOGNIZED OBJECTS - Delete a previously recognized object from the list.

OPEN GRIPPER FOR GRASP FAILURE - Open gripper.

GRASP OBJ - Used to bypass visual servoing component and grasp the object without doing fine alignment on the recognized object.

STOP COMPUTE ARM BASE SHIFT TO OBJECT WITH FAILURE - Send stop signal to shift base component.

STOP MOVE ARM BASE TO OBJECT WITH FAILURE - Send stop signal to linear motion component (stop the arm movement).

STOP VISUAL SERVOING WITH FAILURE - Send stop signal to visual servoing component.

SET VISUAL SERVOING FAILURE - Sets a flag indicating that visual servoing has failed.

SKIP SOURCE POSE - Delete from task list an object but append it to the end, this means ignore the object but if you have time at the end go for it.

Appendix B

World model files

B.1 Basic transportation domain

```
; author Oscar Lima : olima_84@yahoo.com
; Youbot BTT domain, used for transportation tasks
; This domain can be used for robocup@work competition
; http://www.robocupatwork.org/
; license : GPLv3

(define (domain basic_transport_domain)
 (:requirements :typing :action-costs)
 (:types
        location                ; service areas, points of interest, navigation goals
        robot                   ; your amazing yet powerful robot
        object                          ; objects to be manipulated by the robot
        gripper                         ; robot gripper
        robot_platform          ; platform slots for the robot to store objects
 )

 (:predicates

        ; robot ?r is at location ?l
        (at ?r - robot ?l - location)

        ; object ?o is on location ?l
        (on ?o - object ?l - location)

        ; object ?o is stored on robot platform ?rp
        (stored ?o - object ?rp - robot_platform)

        ; robot platform ?rp is occupied, yb has 3 free places to store objects
        (occupied ?rp - robot_platform)

        ; gripper ?g is holding object ?o
        (holding ?g - gripper ?o - object)

        ; gripper ?g is free (does not contain object)
        (gripper_is_free ?g - gripper)

        ; an object ?o is perceived when object recognition was triggered
        ; gets lost if the robot moves the base
        (perceived ?l - location)
 )

 (:functions
          (path-length ?source ?destination - location) - number
        (total-cost) - number
 )

```

```
; move the base of a robot ?r from ?source - location to a ?destination - location
; NOTE : the situation in which the robot arm is in any position before moving
; is not handled at the planning level, therefore we advise to always move the arm
; to a safe position, then navigate (at execution level)
 (:action move_base_safe
     :parameters (?source ?destination - location ?r - robot ?g - gripper)
     :precondition (and (at ?r ?source)
                                        (gripper_is_free ?g)
                        )
     :effect (and (not (at ?r ?source))
                          (at ?r ?destination)
                          (not (perceived ?destination))
                          (not (perceived ?source))
             (increase (total-cost) (path-length ?source ?destination))
                  )
 )

 ; perceive an object ?o which is in a location ?l with a empty gripper ?g
 ; to find the pose of this object before it can be picked
 (:action perceive_location
   :parameters (?l - location ?r - robot ?g - gripper)
   :precondition        (and    (at ?r ?l)
                                                        (gripper_is_free ?g)
                                                        (not (perceived ?l))
                                                )
   :effect      (and    (perceived ?l)
                                        (increase (total-cost) 1)
                        )
 )

 ; pick an object ?o which is inside a location ?l with a free gripper ?g
 ; with robot ?r that is at location ?l
 (:action pick
     :parameters (?o - object ?l - location ?r - robot ?g - gripper)
     :precondition       (and    (on ?o ?l)
                                 (at ?r ?l)
                                 (perceived ?l)
                                 (gripper_is_free ?g)
                                 (not (holding ?g ?o))
                         )
     :effect (and        (holding ?g ?o)
                         (not (on ?o ?l))
                         (not (gripper_is_free ?g))
                         (holding ?g ?o)
                         (increase (total-cost) 1)
             )
 )

; deliver an object ?o which the robot ?r is holding with gripper ?g in location ?l
 (:action place
     :parameters (?o - object ?l - location ?r - robot ?g - gripper)
     :precondition  (and  (at ?r ?l)
                          (holding ?g ?o)
                          (not (on ?o ?l))
                          (not (gripper_is_free ?g))
                    )
     :effect (and    (on ?o ?l)
                     (not (holding ?g ?o))
                     (gripper_is_free ?g)
                     (increase (total-cost) 1)
             )
 )

 ; stage an object ?o in a robot platform ?rp which is not occupied with a gripper ?g
 ; which is holding the object ?o
 (:action stage
     :parameters (?o - object ?rp - robot_platform ?g - gripper)
     :precondition       (and    (holding ?g ?o)
                                 (not (occupied ?rp))
                             (not (gripper_is_free ?g))
                         )
     :effect (and        (not (holding ?g ?o))
                                        (gripper_is_free ?g)
                                 (stored ?o ?rp)
                         (occupied ?rp)
```

```
122                         (increase (total-cost) 1)
123             )
124  )
125
126  ; unstage an object ?o stored on a robot platform ?rp with a free gripper ?g
127  (:action unstage
128     :parameters (?o - object ?rp - robot_platform ?g - gripper)
129     :precondition       (and    (gripper_is_free ?g)
130                                 (stored ?o ?rp)
131                                 (not (holding ?g ?o))
132                         )
133     :effect (and        (not (gripper_is_free ?g))
134                                 (not (stored ?o ?rp))
135                         (not (occupied ?rp))
136                         (holding ?g ?o)
137                         (increase (total-cost) 1)
138             )
139  )
140 )
```

Listing B.1: Basic transportation domain

B.2 Insertion task as an extension to transport domain

An insertion operator needs to be added to the domain.

```
1  ; inserts a object ?o which gripper ?g is holding into another object ?o at location ?l
2  (:action insert
3    :parameters (?peg ?hole - object ?g - gripper ?r - robot ?l - location)
4    :precondition (and
5                         (at ?r ?l)
6                         (on ?hole ?l)
7                         (holding ?g ?peg)
8                         (container ?hole)
9                         (not (gripper_is_free ?g))
10                        (not (container ?peg)) ;a container cannot be inserted into a container
11                        (perceived ?l)
12                )
13   :effect      (and
14                        (not (holding ?g ?peg))
15                        (gripper_is_free ?g)
16                        (in ?peg ?hole)
17                        (on ?peg ?l)
18                        (heavy ?hole)
19                        (heavy ?peg) ;it cannot be picked again, once it is inserted
20                        (increase (total-cost) 1)
21                )
22  )
23 )
```

Listing B.2: Insert object operator

The following predicates add support for insertion actions:

```
1 ; object ?o is able to hold other objects
2 (container ?o - object)
3
4 ; object ?o is inserted inside object ?h
5 (in ?peg - object ?hole - object)
6
7 ; An object ?o already inserted in a object ?o cannot be picked again
8 ; this condition is robot dependant
9 (heavy ?o - object)
```

Listing B.3: Insert object predicates

B.3 Basic facts

```
; basic or minimum complementary facts
(define (problem yb_general_domain_task)
(:domain yb_general_domain)
(:objects

    ; robots
    r--youbot-brsu-3 - robot

    ; robot available platforms, to store objects inside the robot
    rp--platform_middle rp--platform_left rp--platform_right - robot_platform

    ; grippers
    g--dynamixel - gripper

    ; locations
    l--S1 l--S2 l--S3 l--S4 l--S5 l--S6 - location
    l--START l--EXIT - location

    ; objects
    o--ER-02-01
)
(:init
    ; robot initial conditions : location
    (at r--youbot-brsu-3 l--START) ; youbot is at start position

    ; status of the gripper at the beginning
    ; youbot gripper does not have any object and therefore is free
    (gripper_is_free g--dynamixel)

    ; objects which robot can insert into other objects
    (container o--ER-02-01)
)
(:goal)
)
```

Listing B.4: Basic facts example

B.4 PDDL problem with 3 transportation tasks

```
(define (problem three_objects_to_transport)
(:domain basic_transportation_domain)
(:objects

    ; robots
    youbot-brsu-5 - robot

    ; robot available platforms, to store objects on rear platform
    platform_middle platform_left platform_right - robot_platform

    ; grippers
    dynamixel - gripper

    ; locations
    S1 S2 S3 S4 S5 S6 - location
    START - location

    ; objects
    o1 - object
    o2 - object
    o3 - object
)
(:init
    ; distance matrix goes here!
    ; we did not include to save space, but can be consulted
    ; in Appendix B under Costs section

    ; robot initial location
    (at youbot-brsu-5 START)

    ; initial status of the gripper
    (gripper_is_free dynamixel)

    ; initial location of objects
    (on o1 S1)
    (on o2 S4)
    (on o3 S2)
)
(:goal (and
            ; transportation tasks
            (on o1 S5)
                    (on o2 S5)
            (on o3 S5)
        )
)
(:metric minimize (total-cost))
)
```

Listing B.5: Basic facts example

B.5 Costs

```
; initialize total cost
    (= (total-cost) 0)

    ; distance matrix lower triangle
    (= (path-length START S1) 3)
    (= (path-length START S2) 4)
    (= (path-length START S3) 7)
    (= (path-length START S4) 7)
    (= (path-length START S5) 7)
    (= (path-length START S6) 7)

    (= (path-length S1 S2) 4)
    (= (path-length S1 S3) 7)
    (= (path-length S1 S4) 7)
    (= (path-length S1 S5) 7)
    (= (path-length S1 S6) 7)

    (= (path-length S2 S3) 7)
    (= (path-length S2 S4) 7)
    (= (path-length S2 S5) 6)
    (= (path-length S2 S6) 6)

    (= (path-length S3 S4) 2)
    (= (path-length S3 S5) 3)
    (= (path-length S3 S6) 4)

    (= (path-length S4 S5) 2)
    (= (path-length S4 S6) 3)

    (= (path-length S5 S6) 2)

    ;----------------------------

    ; distance matrix upper repeated triangle information
    (= (path-length S1 START) 3)
    (= (path-length S2 START) 4)
    (= (path-length S3 START) 7)
    (= (path-length S4 START) 7)
    (= (path-length S5 START) 7)
    (= (path-length S6 START) 7)

    (= (path-length S2 S1) 4)
    (= (path-length S3 S1) 7)
    (= (path-length S4 S1) 7)
    (= (path-length S5 S1) 6)
    (= (path-length S6 S1) 7)

    (= (path-length S3 S2) 7)
    (= (path-length S4 S2) 7)
    (= (path-length S5 S2) 6)
    (= (path-length S6 S2) 6)

    (= (path-length S4 S3) 2)
    (= (path-length S5 S3) 3)
    (= (path-length S6 S3) 4)

    (= (path-length S5 S4) 2)
    (= (path-length S6 S4) 3)

    (= (path-length S6 S5) 2)

    ;----------------------------

    ; extra information, but required by some planners for completeness
    (= (path-length START START) 100)
    (= (path-length S1 S1) 100)
    (= (path-length S2 S2) 100)
    (= (path-length S3 S3) 100)
    (= (path-length S4 S4) 100)
    (= (path-length S5 S5) 100)
    (= (path-length S6 S6) 100)
```

Listing B.6: Navigation costs example

B.6 Locations

```
START: [3.925776, -11.890519, 3.077088]
S1: [2.510854, -13.377813, -0.005268]
S2: [2.544332, -10.275800, -0.083287]
S3: [-1.684375, -13.444420, 3.071386]
S4: [-1.684657, -12.554193, 3.140368]
S5: [-1.706570, -11.809244, 3.122768]
S6: [-1.712864, -10.988499, 3.120443]
S7: [-1.744885, -10.208173, -3.127836]
EXIT: [0.532509, -9.344218, 1.533976]
```

Listing B.7: Locations example x y and theta

B.7 Map

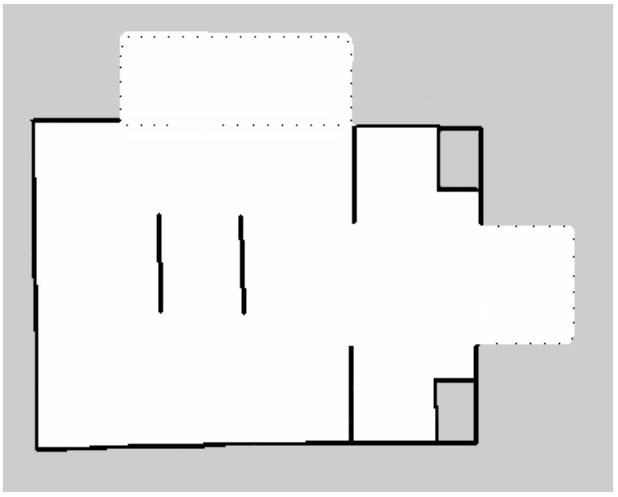


Figure B.1: Example map, recorded at China 2015 RoboCup @Work competition.

resolution: 0.02 meters per pixel

map origin: -5.0, -15.88 meters

Bibliography

[1] Gerhard K Kraetzschmar, Nico Hochgeschwender, Walter Nowak, Frederik Hegger, Sven Schneider, Rhama Dwiputra, Jakob Berghofer, and Rainer Bischoff. Robocup@ work: competing for the factory of the future. In *RoboCup 2014: Robot World Cup XVIII*, pages 171–182. Springer, 2014.

[2] Erico Guizzo. Three engineers, hundreds of robots, one warehouse. *Spectrum, IEEE*, 45(7):26–34, 2008.

[3] Amanda Coles, Andrew Coles, Angel García Olaya, Sergio Jiménez, Carlos Linares López, Scott Sanner, and Sungwook Yoon. A survey of the seventh international planning competition. *AI Magazine*, 33(1):83–88, 2012.

[4] Drew M McDermott. The 1998 ai planning systems competition. *AI magazine*, 21 (2):35, 2000.

[5] Herbert A Simon. Rational choice and the structure of the environment. *Psychological review*, 63(2):129, 1956.

[6] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. Pddl-the planning domain definition language. 1998.

[7] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[8] Edwin P. D. Pednault. Adl: Exploring the middle ground between strips and the situation calculus. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 324–332, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. ISBN 1-55860-032-9. URL `http://dl.acm.org/citation.cfm?id=112922.112954`.

[9] Michael Gelfond and Vladimir Lifschitz. Action languages. 1998.

[10] Tania Bedrax-Weiss, Jeremy Frank, Ari Jónsson, and Conor McGann. Europa2: Plan database services for planning and scheduling applications, 2004.

[11] Malte Helmert. The fast downward planning system. *J. Artif. Intell. Res.(JAIR)*, 26:191–246, 2006.

[12] Michael Katz and Jörg Hoffmann. Mercury planner: Pushing the limits of partial delete relaxation. *IPC 2014 planner abstracts*, pages 43–47, 2014.

[13] Blai Bonnet and Héctor Geffner. Hsp: Heuristic search planner. 1998.

[14] Jörg Hoffmann. Ff: The fast-forward planning system. *AI magazine*, 22(3):57, 2001.

[15] Michael Katz, Jörg Hoffmann, and Carmel Domshlak. Who said we need to relax all variables? In *ICAPS*, 2013.

[16] Malte Helmert and Silvia Richter. Fast downward-making use of causal dependencies in the problem representation. In *Proc. IPC4, ICAPS*, pages 41–43, 2004.

[17] Jörg Hoffmann. Everything you always wanted to know about planning. In *KI 2011: Advances in Artificial Intelligence*, pages 1–13. Springer, 2011.

[18] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos, and Marc Carreras. Rosplan: Planning in the robot operating system. In *Twenty-Fifth International Conference on Automated Planning and Scheduling*, 2015.

[19] Torben Carstensen, Jan Carstensen, Andrej Dick, Jens H/"ubner, Alexander Michailik, Jan Friederichs, , and Daniel Kaczor. Luhbots robocup@work 2016 team description paper. In *RoboCup*, Leipzig, Germany, 2016.

[20] Frederik Zwilling, Tim Niemueller, and Gerhard Lakemeyer. Simulation for the robocup logistics league with real-world environment agency and multi-level abstraction. In *RoboCup 2014: Robot World Cup XVIII*, pages 220–232. Springer, 2014.

[21] Tim Niemueller, Sebastian Reuter, and Alexander Ferrein. Fawkes for the robocup logistics league. In *RoboCup 2015: Robot World Cup XIX*, pages 365–373. Springer, 2015.

[22] Tim Niemüller, Alexander Ferrein, and Gerhard Lakemeyer. A lua-based behavior engine for controlling the humanoid robot nao. In *RoboCup 2009: Robot Soccer World Cup XIII*, pages 240–251. Springer, 2009.

[23] Tim Niemueller, Daniel Ewert, Sebastian Reuter, Alexander Ferrein, Sabina Jeschke, and Gerhard Lakemeyer. The carologistics robocup logistics team 2013. Technical report, Technical report, RWTH Aachen University and Aachen University of Applied Sciences, 2013.

[24] Tim Niemueller, Gerhard Lakemeyer, and Alexander Ferrein. The robocup logistics league as a benchmark for planning in robotics. In *25th International Conference on Automated Planning and Scheduling Jerusalem, Israel. June 7-11*, 2015.

[25] Patrick Henry Winston and Berthold K Horn. Lisp. 1986.

[26] Vandi Verma, Tara Estlin, Ari Jónsson, Corina Pasareanu, Reid Simmons, and Kam Tso. Plan execution interchange language (plexil) for executable plans and command sequences. In *Proceedings of the 9th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2005.

[27] Rainer Bischoff, Ulrich Huggenberger, and Erwin Prassler. Kuka youbot-a mobile manipulator for research and education. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

[28] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[29] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[30] Shehzad Ahmed, Torsten Jandt, Padmaja Kulkarni, Oscar Lima, Arka Mallick, Alexander Moriarty, Deebul Nair, Santosh Thoduka, Iman Awaad, Rhama Dwiputra, Frederik Hegger, Nico Hochgeschwender, Jose Sanchez, Sven Schneider, and Gerhard K. Kraetzschmar. b-it-bots robocup@work team description paper. In *RoboCup*, Leipzig, Germany, 2016. URL `https://mas-group.inf.h-brs.de/wp-content/uploads/2016/01/tdp_b-it-bots_atwork_2016.pdf`.

[31] Tim Field. SMACH documentation. Online at http://www.ros.org/wiki/smach/Documentation, March 2016.

[32] Richard Howey and Derek Long. Val's progress: The automatic validation tool for pddl2. 1 used in the international planning competition. In *Proceedings of the ICAPS Workshop on The Competition: Impact, Organization, Evaluation, Benchmarks*, pages 28–37, 2003.