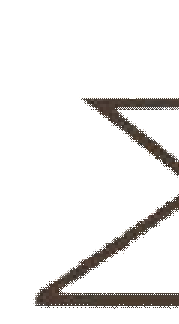
# GOVERNMENT POLYTECHNIC COLLEGE MATTANNUR-670702

### (Department of Technical Education, Kerala)



**SEMINAR REPORT ON**

**AN TU TU BENCHMARK**

**SUBMITTED BY ABHINAV C V**

**(Reg.No:19042501)**

**DEPARTMENT OF ELECTRONICS ENGINEERING**

**2021-22**

**GOVERNMENT POLYTECHNIC COLLEGE MATTANNUR-670702**

**(Department of Technical Education, Kerala)**

****

**CERTIFICATE**

*Certified that seminar work entitled* “***ROS OPERATING SYSTEM*”***is a bonafide work carried out by* **“*ABHINAV C V* ”** *in partial fulfilment for the award of Diploma in Electronics Engineering from Government Polytechnic College Mattannur during the academic year* 2021-2022.

### Seminar Co-ordinator Head of Section

**Internal Examiner External Examiner**

**DECLARATION**

I hereby declare that the report of *the* ***ROS OPRATING SYSTEM*** work entitled which is being submitted to the Govt. Polytechnic College Mattannur, in partial fulfilment of the requirement for the award **of *Diploma in Electronics Engineering*** *i*s a confide report of the work carried out by me. The material in this report has not been submitted to any institute for the award of any degree.

Place:Mattannur **ABHINAV CV**

Date:

### ACKNOWLEDGMENT

I would like to take this opportunity to extend my sincere thanks to people who helped me to make this seminar possible. This seminar will be incomplete without mentioning all the people who helped me to make itreal.

Firstly, I would like to thank GOD, almighty, our supreme guide, for bestowing his blessings upon me in my entire endeavor.

I would like to express my deepest gratitude **Mr. M C PRAKASHAN** (Principal GPTC, Mattannur), **Mr. GEORGE KUTTY (**Head of Department of Electronics Engg.), for the help rendered by him to prepare and present this Seminar in proper way. Moreover I am very much indebted to **Mr. SREEJITH A** (Lecturer, Electronics Engg, seminar co-ordinator), for their advice.

I am also indebted to all my friends and classmates who have given valuable suggestion and encouragement.

**ABHINAV CV**

**ABSTRACT**

Nowadays the benchmarking tool is commonly used to benchmark phones and other devices to check the performance of the device itself. Here I concentrated on the area of mobile phones .In market there are large amount of mobile phones available in flagship rate , which is provided is very small budget for the normal people . Hence it is important to check the performance and the durability of the mobile phones. The process is simple. You can use An Tu Tu Benchmark to analyze the performance of your device. Each test gets a score that gives you an idea of the capability of your device. You can then compare your scores with those of other phones on the market to see how it ranks among them.

The AnTuTu benchmark is so common that some hardware manufacturers have cheated on the benchmark which makes the benchmark unreliable.In response to this, AnTuTu has created a new benchmark, called AnTuTu X, which makes it more difficult for manufacturers to cheat on the benchmark.

### TABLE OF CONTENTS

**CHAPTERNO TOPICS PAGENO**

Chapter1 INTRODUCTION……………………………………...9

Chapter2 DESIGN AND GOALS………………………………..10

Chapter2.1 PEER TO PEER………………………………………...10

Chapter2.2 MULTI LINGUAL……………………..………………11

Chapter2.3 TOOLS BASED…………….....………..………………12

Chapter2.4 THIN……………………………………………………12

Chapter2.5 FREE AND OPEN SOFTWWARE……………………13

Chapter3 NOMENCLATURE………………………..………….14

Chapter3.1 USE CASES………………………………….………..15

Chapter3.2 DEBUGGING A SINGLE NODE…………………….15

Chapter3.3 LOGGING AND PLAYBACK………………………..16

Chapter3.4 PACKAGED SYSTEM……………………………….17

Chapter4 COLLABORARTIVE METHOD..……………………18

Chapter5 VISUALISATION AND MONITORING…………….20

Chapter6 COMPOSITION AND FUNCTIONALITY…..………..21

Chapter7 TRANSFORMATION 22

Chapter8 CONCLUSION 23

Chapter9 ACKNOWLEDGEMENT 24

Chapter10 REFERENCE. 25

|  |  |  |
| --- | --- | --- |
| **FIGURE NO** | **LIST OF FIGURES**  **NAME** | **PAGE NO** |
| Fig 2.1 | DESIGN AND GOALS......................................  .. | 10  11 |
| Fig 3.1  Fig 3.2  Fig 3.3 | NOMENCLATURE……………….......…….…  LOGGING AND PLAYBACK…………………  PACKAGED SUBSYSTEM……………..……. | 14  16  17 |
| Fig 6.1 | HEIRARCHICAL MULTI ROBOT CONTROL SYSTEM………………………………..……… | 21 |

**CHAPTER 1**

**INTRODUCTION**

Writing software for robots is difficult, particularly as the scale and scope of robotics continues to grow. Differenttypes of robots can have wildly varying hardware, making code reuse nontrivial. On top of this, the sheer size of the required code can be daunting, as it must contain a deepstack starting from driver-level software and continuing up through perception, abstract reasoning, and beyond. Since therequired breadth of expertise is well beyond the capabilities of any single researcher, robotics software architectures must also support large-scale software integration efforts. To meet these challenges, many robotics researchers, in- clouding ourselves, have previously created a wide varietyof frameworks to manage complexity and facilitate rapid prototyping of software for experiments, resulting in the many robotic software systems currently used in academia and industry [1]. Each of these frameworks was designed fora particular purpose, perhaps in response to perceived weak- nesses of other available frameworks, or to place emphasis on aspects which were seen as most important in the design process. ROS, the framework described in this paper, is also the product of tradeoffs and prioritizations made during its de- sign cycle. We believe its emphasis on large-scale integrativerobotics research will be useful in a wide variety of situationsas robotic systems grow ever more complex. In this paper, we discuss the design goals of ROS, how our implementationworks towards them, and demonstrate how ROS handlesseveral common use cases of robotics software development.

**CHAPTER 2**

**DESIGN GOALS**

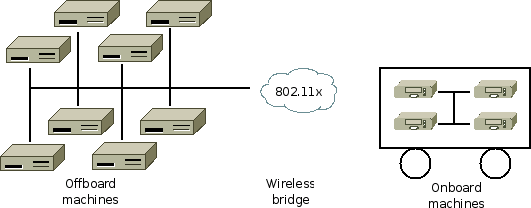
 We do not claim that ROS is the best framework forall robotics software. In fact, we do not believe that sucha framework exists; the field of robotics is far too broadfor a single solution. ROS was designed to meet a specificset of challenges encountered when developing large-scale

Fig 2.1 Design goals

service robots as part of the STAIR project [2] at Stanford University1 and the Personal Robots Program [3] at Willow Garage,2 but the resulting architecture is far more general than the service-robot and mobile-manipulation domains.

The philosophical goals of ROS can be summarized as:

• Peer-to-peer

• Tools-based

• Multi-lingual

• Thin

• Free and Open-Source

To our knowledge, no existing framework has this same set of design criteria. In this section, we will elaborate these philosophies and shows how they influenced the design and implementation of ROS.

**2.1 Peer-to-Peer**

A system built using ROS consists of a number of pro- cases, potentially on a number of different hosts, connected at runtime in a peer-to-peer topology. Although frameworks based on a central server (e.g., CARMEN [4]) can also re- Alize the benefits of the multiprocess and multi-host design, a central data server is problematic if the computers are connected in a heterogenous network. For example, on the large service robots for which ROS was designed, there are typically several onboard computers connected via ethernet. This network segment is bridgedvia wireless LAN to high-power offboard machines that are running computation-intensive tasks such as computer vision or speech recognition (Figure 1). Running the central server either onboard or offboard would result in unnecessary.

1http://stair.stanford.edu 2http://pr.willowgarage.com

Traffic flowing across the (slow) wireless link, because manymessage routes are fully contained in the subnets either onboard or offboard the robot. In contrast, peer-to-peer connectivity, combined with buffering or “fanout” software modules where necessary, avoids the issue entirely. The peer-to-peer topology requires some sort of lookup mechanism to allow processes to find each other at runtime. We call this the name service, or master, and will describeit in more detail shortly.

**2.2 Multi-lingual**

When writing code, many individuals have preferences forsome programming languages above others. These prefer- fences are the result of personal tradeoffs between programMing time, ease of debugging, syntax, runtime efficiency, and a host of other reasons, both technical and cultural. For these reasons, we have designed ROS to be language-neutral. ROS currently supports four very different languages: C++, Python, Octave, and LISP, with other language ports in various stages of completion. The ROS specification is at the messaging layer, not any deeper. Peer-to-peer connection negotiation and configure- ton occurs in XML-RPC, for which reasonable implementstations exist in most major languages. Rather than providea C-based implementation with stub interfaces generated for all major languages, we prefer instead to implement ROS natively in each target language, to better follow the conventions of each language. However, in some cases it is expedient to add support for a new language by wrappingan existing library: the Octave client is implemented by wrapping the ROS C++ library. To support cross-language development, ROS uses a sim- plea, language-neutral interface definition language (IDL) to describe the messages sent between modules. The IDL uses (very) short text files to describe fields of each message,and allows composition of messages, as illustrated by the complete IDL file for a point cloud message:

Header header

Point32[] pts

ChannelFloat32[]

Code generators for each supported language then generatenative implementations which “feel” like native objects, and are automatically serialized and deserialized by ROS as messages are sent and received. This saves considerable programmer time and errors: the previous 3-line IDL file automatically expands to 137 lines of C++, 96 lines of Python, 81 lines of Lisp, and 99 lines of Octave. Because themessages are generated automatically from such simple text files, it becomes easy to enumerate new types of messages. At time of writing, the known ROS-based codebases containover four hundred types of messages, which transport data ranging from sensor feeds to object detections to maps. The end result is a language-neutral message processing scheme where different languages can be mixed and matchedas desired.

**2.3 Tools-based**

In an effort to manage the complexity of ROS, we have opted for a microkernel design, where a large number of small tools are used to build and run the various ROS components, rather than constructing a monolithic development and runtime environment. These tools perform various tasks, e.g., navigate the sourcecode tree, get and set configuration parameters, visualizethe peer-to-peer connection topology, measure bandwidthutilization, graphically plot message data, auto-generate doc-mentation, and so on. Although we could have implementedcore services such as a global clock and a logger insidethe master module, we have attempted to push everything into separate modules. We believe the loss in efficiency is more than offset by the gains in stability and complexity management.

**2.4 Thin**

As eloquently described in [5], most existing roboticssoftware projects contain drivers or algorithms which couldbe reusable outside of the project. Unfortunately, due toa variety of reasons, much of this code has become soentangled with the middleware that it is difficult to “extract”its functionality and re-use it outside of its original context.To combat this tendency, we encourage all driver and algorithm development to occur in standalone libraries thathave no dependencies on ROS. The ROS build systemperforms modular builds inside the source code tree, andits use of Cake makes it comparatively easy to follow this“thin” ideology. Placing virtually all complexity in libraries,and only creating small executables which expose libraryfunctionality to ROS, allows for easier code extraction andreuse beyond its original intent. As an added benefit, unittesting is often far easier when code is factored into libraries,as standalone test programs can be Header header Point32 pts ChannelFloat32[] Chan written to exercise various features of the library. ROS re-uses code from numerous other open-source projects, such as the drivers, navigation system, and sim- orators from the Player project [6], vision algorithms from OpenCV [7], and planning algorithms from Open RAVE [8], among many others. In each case, ROS is used only to exposevarious configuration options and to route data into and out ofthe respective software, with as little wrapping or patchingas possible. To benefit from the continual community im- provident, the ROS build system can automatically update source code from external repositories, apply patches, andso on.

**2.5 Free and Open-Source**

The full source code of ROS is publicly available. We believe this to be critical to facilitate debugging at all levels of the software stack. While proprietary environments such as Microsoft Robotics Studio [9] and WebOS [10] have many commendable attributes, we feel there is no substitute for a fully open platform. This is particularly true whenhardware and many levels of software are being designedand debugged in parallel. ROS is distributed under the terms of the BSD license, which allows the development of both non-commercial and commercial projects. ROS passes data between modules using inter-process communications, and does not require that modules link together in the same executable. As such, systems built around ROS can use fine-grain licensing of their various components: individual modules can Incorp- rate software protected by various licenses ranging from GPLto BSD to proprietary, but license “contamination” ends at the module boundary.

**CHAPTER 3**

**NOMENCLATURE**

The fundamental concepts of the ROS implementation arenodes, messages, topics, and services, Nodes are processes that perform computation. ROS is designed to be modular at a fine-grained scale: a systemis typically comprised of many nodes. In this context, the term “node” is interchangeable with “software module.” Our use of the term “node” arises from visualizations of ROS- based systems at runtime: when many nodes are running, itis convenient to render the peer-to-peer communications asa graph, with processes as graph nodes and the peer-to-peer links as arcs. Nodes communicate with each other by passing messages.A message is a a strictly typed data structure. Standard primitive types (integer, floating point, Boolean, etc.) are supported, as are arrays of primitive types and constants. Messages can be composed of other messages, and arrays of other messages, nested arbitrarily deep. A node sends a message by publishing it to a given topic, which is simply a string such as “odometry” or “map.” A node that is interested in a certain kind of data will subscribeto the appropriate topic. There may be multiple concurrent publishers and subscribers for a single topic, and a single node may publish and/or subscribe to multiple topics. In general, publishers and subscribers are not aware of each other’s existence. The simplest communications are along pipelines:

microphone

speech recognition

dialog manager

speech synthesis

speaker

Fig 3.1 Nomenclature

However, graphs are usually far more complex, often con- training cycles and one-to-many or many-to-many connect- tons.

Although the topic-based publish-subscribe model is aflexible communications paradigm, its “broadcast” routing scheme is not appropriate for synchronous transactions, which can simplify the design of some nodes. In ROS, we call this a service, defined by a string name and a pairof strictly typed messages: one for the request and one for the response. This is analogous to web services, which are defined by URIs and have request and response documentsof well-defined types. Note that, unlike topics, only one nodecan advertise a service of any particular name: there can onlybe one service called” classify image”, for example, just as there can only be one web service at any given URI.

**3.1 Use cases**

In this section, we will describe a number of common scenarios encountered when using robotic software frame- works. The open architecture of ROS allows for the creation of a wide variety of tools; in describing the ROS approachto these use cases, we will also be introducing a number of the tools designed to be used with ROS.

**3.2 Debugging a single node**

When performing robotics research, often the scope ofthe investigation is limited to a well-defined area of thesystem, such as a node which performs some type of planning, reasoning, perception, or control. However, to geta robotic system up and running for experiments, a much larger software ecosystem must exist. For example, to do visionbased grasping experiments, drivers must be running for the camera(s) and manipulator(s), and any number of intermediate processing nodes (e.g., object recognizers, posedetectors, trajectory planners) must also be up and running. This adds a significant amount of difficulty to integrative robotics research.

ROS is designed to minimize the difficulty of debuggingin such settings, as its modular structure allows nodes undergoing active development to run alongside preexisting,well-debugged nodes. Because nodes connect to each otherat runtime, the graph can be dynamically modified. In the previous example of vision-based grasping, a graph withperhaps a dozen nodes is required to provide the infares-structure. This “infrastructure” graph can be started and left running during an entire experimental session. Only thenode(s) undergoing source code modification need to beperiodically restarted, at which time ROS silently handlesthe graph modifications. This can result in a massive increasein productivity, particularly as the robotic system becomes more complex and interconnected.

To emphasize, altering the graph in ROS simply amounts to starting or stopping a process. In debugging settings, thisis typically done at the command line or in a debugger. The ease of inserting and removing nodes from a running ROS- based system is one of its most power full and fundamental features.

**3.3 Logging and playback**

Research in robotic perception is often done most con- leniently with logged sensor data, to permit controlled comparisons of various algorithms and to simplify the ex- peri mental procedure. ROS supports this approach by pro- viding generic logging and playback functionality. Any ROS message stream can be dumped to disk and later replayed. Importantly, this can all be done at the command line; it requires no modification of the source code of any pieces of software in the graph. For example, the following network graph could be quickly set up to collect a dataset for visual-odometry research:

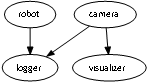


Fig 3.2 Logging and playback

As before, node instantiation can be performed simply by launching a process; it can be done at the command line, ina debugger, from a script, etc. To facilitate logging and monitoring of systems distributedacross many hosts, the Ros console library builds upon the Apache project’s log4cxx system to provide a convenientand elegant logging interface, allowing print-style dig- nastic messages to be routed through the network to a single stream called roost.

**3.4 Packaged subsystems**

Some areas of robotics research, such as indoor robotnavigation, have matured to the point where “out of thebox” algorithms can work reasonably well. ROS leverages the algorithms implemented in the Player project to providea navigation system, producing this graph:

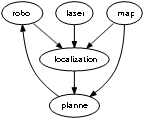


Fig 3.3 packaged subsystem

Although each node can be run from the command line, re- heatedly typing the commands to launch the processes could get tedious. To allow for “packaged” functionality such as a navigation system, ROS provides a tool called relaunch,which reads an XML description of a graph and instantiates the graph on the cluster, optionally on specific hosts. The end-user experience of launching the navigation system then boils down to

*relaunch navstack.xml*

and a single Ctrl-C will gracefully close all five processes. This functionality can also significantly aid sharing and reuseof large demonstrations of integrative robotics research, as the set-up and tear-down of large distributed systems can be easily replicated.

**CHAPTER 4**

**COLLABORARTIVE METHOD**

Due to the vast scope of robotics and artificial intelligence,collaboration between researchers is necessary in order to build large systems. To support collaborative development, the ROS software system is organized into packages. Our definition of “package” is deliberately open-ended: a ROS package is simply a directory which contains an XML file describing the package and stating any dependencies.

A collection of ROS packages is a directory tree with ROSpackages at the leaves: a ROS package repository may thus contain an arbitrarily complex scheme of subdirectories. For example, one ROS repository has root directories including “nav,” “vision,” and “motion planning,” each of which con- trains many packages as subdirectories.

ROS provides a utility called repack to query andinspect the code tree, search dependencies, find packagesby name, etc. A set of shell expansions called rosebush is provided for convenience, accelerating command-line navy- ration of the system.

The repack utility is designed to support simulate- oust development across multiple ROS package repositories. Environment variables are used to define the roots of local copies of ROS package repositories, and repack crawls the package trees as necessary. Recursive builds, supported by the remake utility, allow for cross-package library dependencies.

The open-ended nature of ROS packages allows for great variation in their structure and purpose: some ROS packageswrap existing software, such as Player or OpenCV, au- to mating their builds and exporting their functionality. Somepackages build nodes for use in ROS graphs, other packages provide libraries and standalone executables, and still others provide scripts to automate demonstrations and tests. The packaging system is meant to partition the building of ROS- based software into small, manageable chunks, each of whichcan be maintained and developed on its own schedule by its own team of developers.

At time of writing, several hundred ROS packages exist across several publicly-viewable repositories, and hundreds more likely exist in private repositories at various institutions and companies. The ROS core is distributed as its own package repository in Source forge:

<http://ros.sourceforge.net>

However, the Ros repository includes only the base ROS communications infrastructure and graph-management tools.Software which actually builds robotic systems using ROS is provided in a second repository, also on Source forge:

<http://personalrobots.sourceforge.net>

This repository contains many useful tools and libraries, suchas those discussed in this paper.

**CHAPTER 5**

**VISUALISATION AND MONITORING**

While designing and debugging robotics software, it oftenbecomes necessary to observe some state while the systemis running. Although print is a familiar technique for debugging programs on a single machine, this technique can be difficult to extend to large-scale distributed systems, and can become unwieldly for general-purpose monitoring.

Instead, ROS can exploit the dynamic nature of theconnectivity graph to “tap into” any message stream onthe system. Furthermore, the decoupling between publishersand subscribers allows for the creation of general-purposevisualizers. Simple programs can be written which subscribeto a particular topic name and plot a particular type of data, such as laser scans or images. However, a more powerfulconcept is a visualization program which uses a pluginarchitecture: this is done in the viz program, which is distributed with ROS. Visualization panels can be dynamic-call instantiated to view a large variety of datatypes, suchas images, point clouds, geometric primitives (such as objectrecognition results), render robot poses and trajectories, etc.Plugins can be easily written to display more types of data.A native ROS port is provided for Python, a dynamically- typed language supporting introspection. Using Python, apowerful utility called Ros’s topic was written to filtermessages using expressions supplied on the command line,resulting in an instantly customizable “message tap” whichcan convert any portion of any data stream into a text stream.

These text streams can be piped to other UNIX command-line tools such as grep, sed, and awk, to create complex monitoring tools without writing any code. Similarly, a tool called replot provides the functionalityof a virtual oscilloscope, plotting any variable in real-time asa time series, again through the use of Python introspection and expression evaluation.

**CHAPTER 6**

COMPOSITION AND FUNCTIONALITY

In ROS, a “stack” of software is a cluster of nodes that does something useful, as was illustrated in the navigation example. As previously described, ROS is able to instantiate a cluster of nodes with a single command, once the clusteris described in an XML file. However, sometimes multiple instantiations of a cluster are desired. For example, in multirobot experiments, a navigation stack will be needed foreach robot in the system, and robots with humanoid torsos will likely need to instantiate two identical arm controllers. ROS supports this by allowing nodes and entire relaunchcluster-description files to be pushed into a child namespace,thus ensuring that there can be no name collisions. Essentally, this prepends a string (the namespace) to all node,

topic, and service names, without requiring any modificationto the code of the node or cluster. The following graph shows a hierarchical multi-robot control system constructed by simply instantiating multiple navigation stacks, each in their own namespace:

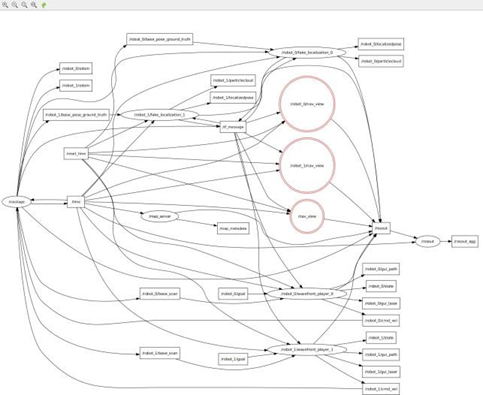
****

Fig 6.1 Hierarchical multi-robot control system

The previous graph was automatically generated by the graph tool, which can inspect and monitor any ROS graph at runtime. Its output renders nodes as ovals, topics as squares, and connectivity as arcs

**CHAPTER 7**

**TRANSFORMATION**

Robotic systems often need to track spatial relationships for a variety of reasons: between a mobile robot and some fixed frame of reference for localization, between the varioussensor frames and manipulator frames, or to place frames on target objects for control purposes.

To simplify and unify the treatment of spatial frames,a transformation system has been written for ROS, calledft. The ft system constructs a dynamic transformation tree which relates all frames of reference in the system.As information streams in from the various subsystems ofthe robot (joint encoders, localization algorithms, etc.), the ft system can produce streams of transformations betweennodes on the tree by constructing a path between the desired nodes and performing the necessary calculations.

For example, the ft system can be used to easily generate point clouds in a stationary “map” frame from laser scans received by a tilting laser scanner on a moving robot. As another example, consider a two-armed robot: the ft system can stream the transformation from a wrist camera on one robotic arm to the moving tool tip of the second arm of the robot. These types of computations can be tedious, error- prone, and difficult to debug when coded by hand, but theft implementation, combined with the dynamic messaging infrastructure of ROS, allows for an automated, systematic approach.

**CHAPTER 8**

**CONCLUSION**

We have designed ROS to support our philosophy of modular, toolsbased software development. We anticipate that its open-ended design can be extended and built upon by others to build robot software systems which can be usefulto a variety of hardware platforms, research settings, and runtime requirements.

**CHAPTER 9**

**ACKNOWLEDGEMENTS**

We thank the fledgling ROS user community for their feedback and contributions, especially Rosen Dikano (au- thorn of the ROS Octave library) and Bhaskar Martha (authorof the ROS LISP library)

**CHAPTER 10**

**REFERNCE**

[1] J. Kramer and M. Schuetz, “Development environments for au- tenuous mobile robots: A survey,” Autonomous Robots, vol. 22, no. 2, pp. 101–132, 2007.

[2] M. Quigley, E. Berger, and A. Y. Ng, “STAIR: Hardware and Software Architecture,” in AAAI 2007 Robotics Workshop, Vancouver, B.C, August, 2007.

[3] K. WyoTech, E. Berger, H. V. der Loos, and K. Salisbury, “Towards a personal robotics development platform: Rationale and design of an intrinsically safe personal robot,” in Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA), 2008.

[4] M. Montebello, N. Roy, and S. Thrum, “Perspectives on standardize- ton in mobile robot programming: The Carnegie Mellon Navigation (CARMEN) Toolkit,” in Proc. of the IEEE/RSJ Intl. Conf. on IntelliJ- gent Robots and Systems (IROS), Las Vegas, Nevada, Oct. 2003, pp. 2436–2441.

[5] A. Makarenko, A. Brooks, and T. Kauppi, in Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS), Nov. 2007.

[6] R. T. Vaughan and B. P. Geekery, “Reusable robot code and the Player/Stage Project,” in Software Engineering for Experimental Robotics, ser. Springer Tracts on Advanced Robotics, D. Brugal, Ed. Springer, 2007, pp. 267–289.

[7] G. Brad ski and A. Koehler, Learning OpenCV, Sep. 2008.

[8] R. Dikano and J. Kuffner, “The robotic busboy: Steps towards developing a mobile robotic home assistant,” in Intelligent Autonomous Systems, vol. 10, 2008.

[9] J. Jackson, “Microsoft robotics studio: A technical introduction,” in IEEE Robotics and Automation Magazine, Dec. 2007, http://msdn. microsoft.com/end-us/robotics. [10] O. Michel, “WebOS: a powerful realistic mobile robots’ simulator,” in Proc. of the Second Intl. Workshop on Robocopy. SpringerVerlag, 1998