Proceedings, 44th IEEE Southeastern Symposium on System Theory (SSST)
Jascksonville FL, USA May 11-13, 2012

Robot Operating Systems: Bridging the Gap between Human and Robot

John Kerr, Kevin Nickels
Trinity University Engineering
jkerr@trinity.edu, kevin.nickels@trinity.edu

Abstract - **A robot operating system (ROS) is a collection of programs which allow a user to easily control the mobile operations of a robot. This paper describes research conducted on sixteen different ROSs to determine which one will most accommodate future Trinity undergraduates for use in further robotics research. The goal of this research is to reduce this list of 16 ROSs to a single ROS that can be used by Trinity undergraduates with limited programming experience to perform simple robotic motion tasks. First, a detailed list of criteria describing the ideal ROS was created. The list of ROSs was narrowed down to a single ROS that best fit these criteria. This ROS is called Player/Stage. Next, Player/Stage was tested to ensure the validity of the research performed. In these tests, a robot's mobility and sensors were controlled by a user via Player/Stage. This ROS excelled in both the mobility tests and the sensor tests, and also proved simple to navigate and manage.**

I. INTRODUCTION

In order to understand what a robot operating system is and what function it serves, it is important to first understand the function of an ordinary operating system on a personal computer. An operating system is a collection of programs which control the raw computing power of the hardware of the computer. The operating system retains control of the hardware by choosing when application programs will receive computer resources, and when these programs will not. Computer resources can be either hardware or software; examples include the CPU, main memory, input-output devices, communication devices, and data.

The operating system also provides a user-friendly environment for the execution of the application programs, with the ultimate goal of producing useful work. A user-friendly environment is one in which the low-level details of the bare hardware machine are separated and hidden so as to provide the user with an interface that is clean, uncluttered, and easy to navigate.

A robot operating system (ROS) is similar to that of an operating system on a personal computer, in that it comprises a collection of programs which offer control to a user. In the case of an ROS however, these programs allow a user to control the mobile operations of a robot rather than applications on a computer. A good ROS will also make this control user-friendly.

This research consisted of analyzing 16 ROSs in order to determine which one would best serve Trinity engineering undergraduate students in future research and in the classroom. These students are assumed to have a basic knowledge of a common programming language, such as C or java, along with a limited knowledge of embedded systems, as this is typical for the target audience.

II. GOALS AND PURPOSE

The goals of this research are all intended to benefit the programs and classes at Trinity University in some way, and the purpose is to set the foundation for these goals to be implemented. There are three goals of this research, and each of these goals will improve programs at Trinity. The first goal of this research is to have a working robot operating system to be used as a teaching tool in the Trinity classroom. This will allow professors to use this technology at Trinity to help reduce the barriers to entry in robotics. For example, some students may be interested in getting a robot to move around a room and avoid obstacles, but might find the smaller, lower level tasks daunting; such as calculating the rotational velocity of a wheel and the rotary time required to move a robot 10 feet forward.

The second goal of this research is to help prevent 'reinventing the wheel'. Often in the robotics field, the microprocessor that controls a robot has very specific code that is required to program it. This means that if a year or two down the line, the microprocessor being used becomes obsolete and must be replaced, all of the code that was written thus far to control that robot is no longer compatible and will not work. Robot operating systems allow for controlling code to be written in a common, more abstract language, and thus to be easily transferrable to a different controlling microprocessor. Therefore, having a working robot operating system for students to use will be beneficial to the engineering program at Trinity because there will be less wasted time due to incompatibility issues.

The third and final goal of this research is to allow a better, more relevant engineering education to be provided to Trinity engineering students. If an acceptable robot operating system is found that can be used as a teaching tool by Trinity engineering professors, Trinity students will be able to work with tools that are pertinent to current engineering practices.

III. METHODOLOGY

To narrow down the list of ROSs, it was first considered what was needed from an ideal ROS. Therefore, the first step was to create a list of criteria which would designate attributes required from the ROS. The ROS that most closely fit these criteria would, in theory, best fulfill the research goals.

After gaining a better grasp on what type of ROS was required, the first pass at literary research began. Key points of each of the 16 ROSs such as the developer of the ROS, the date of the latest update released, the number and quality of tutorials, etc., were noted and compared. From this information, it was possible to eliminate six of the initial 16 ROSs.

Also, after looking at these ROSs closer and gaining more knowledge of how an ideal ROS should function, it was then possible to update the criteria by both adding criteria that were not considered beforehand and detailing specifics of the criteria already present. Taking this one step further, points were assigned out of a total of 100 to each criterion according to how important each one was.

With these new weighted criteria, the next step was to perform a second research pass at the 10 remaining ROSs on the list. With each ROS, each criterion was given a score out of its total weighted score. In this way, once the ROS was given a score in each of the criterion categories, the summation of these scores would yield a final overall score out of 100 for each ROS remaining on the list. These scores allowed for seven more ROSs to be eliminated. However, three ROSs each scored high enough on this absolute scale, as well as close enough to each other that a final research pass was required to narrow down the list to only one ROS.

In the final pass of research, it was considered how each ROS was being used in current research, either in academia or in industry. This research allowed the most relevant and useful ROS out of the three remaining to be chosen.

Once this ROS was chosen, a copy of the software was downloaded and tested on a Linux based workstation controlling an iRobot Create mobile robot. Tests included simple physical mobile and sensor operations, as well as simulations of both.

IV. Criteria

The characteristics that affect the utility of an ROS, based on the research goals, can be divided into five criteria; **easy to use, capable, adaptable, easy to install and maintain, and developmental stage**.

Easy to use encompasses not only a user-friendly interface that provides basic functions which keep the low-level, unnecessary details of the hardware separated and hidden, but it also includes documentation provided for the ROS in question. Documentation consists of tutorials, downloadable code, and a dictionary of functions; all generally found on the official website of the ROS in question. Tutorials, if provided, must be well written and detailed, and also must be simple enough to be understood by the target undergraduate student who has a basic level of programming skill. Functions are code which is used to command a robot. A dictionary of these functions is useful because it provides a quick and easy way to look up specific commands needed by the user. Finally, downloadable code can be useful because other users could have already written programs that perform the actions the user needs, thus saving valuable time. The easy to use criterion was assigned a weight of 23 which is the highest of all the criteria.

The ROS also has to be *capable*. This criterion can be split into the capability of the ROS simulator, and the capability of the ROS to physically control the mobile operations and sensors of the robot. The properties that make a simulator useful are its power and simplicity. Power refers to the ability of the simulator to provide useful data to the user. Simplicity, not to be confused with a lack of power, refers to how easy the simulator is to use. A simulator can be very powerful, but if the target user cannot understand how to get useful data from it, it will not be as helpful. The other aspect to the criterion of capability is the ability of the ROS to control the physical mobile operations as well as the sensors of a robot. ROSs control these aspects of a robot by means of modules or toolkits, which are parts of code which include functions that tell the robot specifically what to do. The capable criterion was given a weight of 22, which means that it is the second most influential criterion on the overall score of an ROS.

Adaptability includes the supported operating systems that can run the ROS, as well as the supported robot hardware that the ROS can run. If the ROS can run on both the Windows operating system as well as multiple Linux based operating systems, then more choice is provided to the user, which can be beneficial. More importantly however, the number of supported robot hardware that the ROS can run affects the number of robots that the ROS can control. The adaptability criterion was given a weight of 20.

The ability of the ROS to be *easily installed and maintained* is also important because this saves time, as well as ensures that the ROS will not fall out of use. As far as installation is concerned, typically, ROSs that run on Linux based operating systems are much more difficult to install that just double clicking a setup file on a Windows based operating system. With respect to the maintenance of an ROS, this can be measured by how much active development and support is given to the ROS by its creator; more specifically, how often it is updated or upgraded. The easy to install and maintain criterion was given a weight of 20 as well because it is of similar importance to adaptability.

The final criterion that determines the overall score of an ROS is its *developmental stage*. If the ROS has been around a while, then it will be less likely to contain bugs or errors in the code. This criterion was weighted 15 which is slightly lower than the others only because an ROS that is new, but still has all of the other qualities proposed by the criteria should not be eliminated simply because it is newer.

V. Second Research Pass Results

Table 1 shows the scored results from the second pass of research of each of the 10 ROSs judged. The final ROS on this list, Webots, was deemed too expensive during this second pass of research and therefore did not receive any scores. This is not an issue for any of the other nine ROSs however because they are all free of charge.

Table 1: Robot Operating System Criteria Scores

ROS	Ref	Ease of Use	Capable	Adaptable	Ease of Install/Maintain	Development	Total (of100)
CARMEN	[4]	14	15	11	16	15	71
RDS	[10]	20	19	18	18	15	92
MOOS	[12]	7	5	8	5	10	35
Player/Stage	[15]	18	20	18	17	15	90
ROS	[20]	10	14	19	13	15	71
Orocos	[14]	8	4	0	20	15	47
YARP	[27]	12	0	5	19	15	51
MRPT	[13]	15	7	15	18	14	69
Urbi	[25]	12	16	20	14	15	77
Webots	[26]	-	-	-	-	-	-

As seen in Table 1, Microsoft Robotics Developer Studio (RDS) and Player/Stage scored high above all the others on the list, but are very close to each other. RDS has a simple and easy to understand interface. It includes extensive, detailed tutorials that a novice programmer would easily be able to grasp. It has eight supported robots, which is fewer than some of the other ROSs that were looked at. However, the robots supported are all at the complexity level and price range of the type of robots that would be used in a Trinity classroom. RDS is well developed and has extensive support from Microsoft. It also comes equipped with a beautiful three-dimensional simulator called Visual Simulation Environment (VSE) that seems to provide useful data to the user [10]. Figure 1 shows an example of a simulated robot in a simulated environment using VSE.

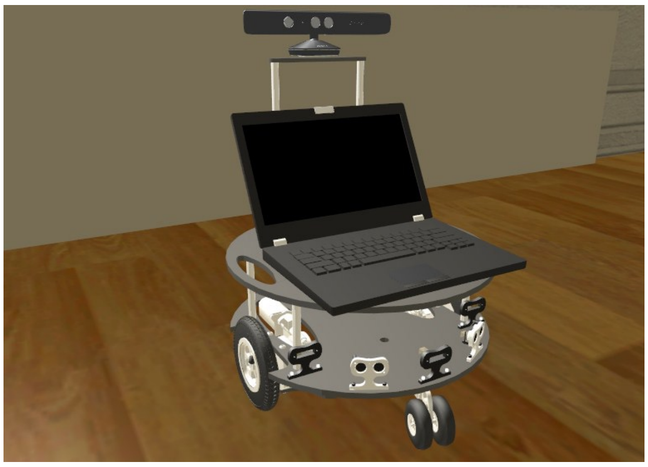
Fig. 1. Simulated robot using VSE in Microsoft Robotics Developer Studio [1].

Player/Stage does not have as simple an interface as RDS. However, the documentation provided for it on the official Player/Stage website is more extensive. It has slightly fewer tutorials, but it makes up for it with its dictionary of functions and commands. This dictionary provides a function for nearly every possible command that a robot could perform. Player/Stage also provides a clean and useful two-dimensional simulator as well as another simulator (Gazebo) that simulates robots and environments in three dimensions. It supports 13 different robots and as with RDS, these robots are all at the correct complexity level and price range required. Player/Stage is a well rounded and well developed robot operating system.

With such a close outcome, a different type of research was required to decide between the two. Also, while researching the robot operating system named ROS, it was found that it is actually the most widely used robot operating system available. Therefore, despite the low scores it received, this robot operating system was included in the final research process.

ROS contains relevant documentation and tutorials. It uses the same two-dimensional simulator (stage) and three-dimensional simulator (gazebo) as Player, contains approximately 60 supported robots, and is well developed, powerful, and complex. The issue with ROS is that it is more complex than is needed to meet the research goals. It is designed to be used by programmers with more experience than the target audience; programmers who aim to produce solutions to problems that are far more complex than simple mobility or sensor operations. ROS would be far too difficult for a student with limited programming experience to manage.

VI. Use in Robotics Research

The final pass at research dealt with determining which robot operating systems were used frequently in current research, be it in academia or industry.

An example of how powerful and complex ROS can be is demonstrated in an experiment at the Worcester Polytech. Institution in Worcester, Massachusetts. A computational model was developed that could recognize engagement such as gestures or speech between a human and a humanoid robot [23]. In order to test the model over a broad range of robot architectures, the ROS framework was used [23].

RDS is not well known at all in the academic research area. Little could be found of any experiments or scientific studies which used RDS as a framework. Every source found that utilized RDS seemed to portray it as a tool for robot hobbyists rather than researchers. For instance, according to Max Reichardt, Lisa Wilhelm, Martin Proetzsch, and Karsten

Berns, “Explicitly targeting non-programmers, … [the language used by Microsoft Robotics Developer Studio] is hardly used in professional robotic projects to our knowledge” [9]. Another source that provides evidence that RDS is not widely used as a full robot operating system in research presents a research scenario in which the simulator of RDS is used to acquire helpful data [24]. However, no other aspects of the robot operating system are used.

Player/Stage however had plenty of appearances in academic research endeavors. Sklar, Elizabeth, Simon Parsons, and Susan Epstein describe a demonstration which shows a framework developed for experimentation in human-robot-team-based interaction and coordination [23]. The experiment consists of sending out multiple robots in different directions to scope out unknown areas. These areas represent areas that could be harmful to human subjects, such as war zones or structurally unsound buildings. The robots communicate with each other with the goal of finding different objects which represent wounded or stranded people in these hazardous areas. Player/Stage is used as the robot operating system which controls these robots.

VII. Final Results, Testing, and Validation

Due to its lack of presence in the academic and research community, RDS was eliminated from the acceptable robot operating system candidates. ROS was eliminated as well due to its complex and confusing nature. With no restrictions on the programming skill of the user, ROS could have been the best choice for a robot operating system. However, since the system chosen has to be worked by students with limited programming experience, ROS was not the best candidate. Therefore, Player/Stage was chosen as the best possible robot operating system for the purpose of this research.

In order to verify the usefulness of Player/Stage, the system was installed on a Linux based computer, and many aspects of it were tested. At first, the tutorials [17] were difficult to comprehend, but after a day or so of performing simple functions using Player/Stage, it became simple for the first author, an undergraduate in the target user group, to work with.

The first aspect of Player/Stage that was tested was the two-dimensional simulator called Stage. Figure 2 shows an example image of a Stage simulation.

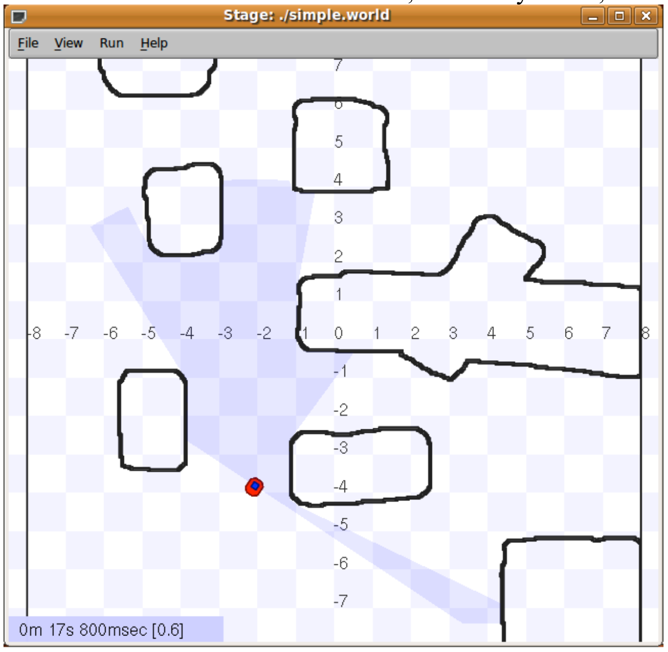

Fig. 2. Two-dimensional simulation using Stage [6].

This simulator provides useful data back to the user while remaining simple to utilize. Examples of useful data include robot sensor readings as well as the Cartesian coordinate position of the robot in real time as the robot makes its way around the environment. The robot sensor readings can be seen in Fig. 2 as the light blue shaded region surrounding the red dot in the middle. This red dot represents the robot, and the shaded region represents what the robot can see at the current position. As the robot makes its way around the environment, this shaded region will change as different objects obstruct the line-of-sight of the robot sensor.

Once the simulator was assessed, tests were conducted in order to determine how Player/Stage fared when controlling a physical robot. The robot used in this test was an iRobot Create, which is very similar to the iRobot Roomba vacuum cleaning robot, only without the vacuum cleaner attached to the bottom of it. This robot contains an infrared sensor on the front top of it as well as a bumper sensor which sends a signal to the robot each time it runs into an obstacle. Figure 3 shows the iRobot Create.

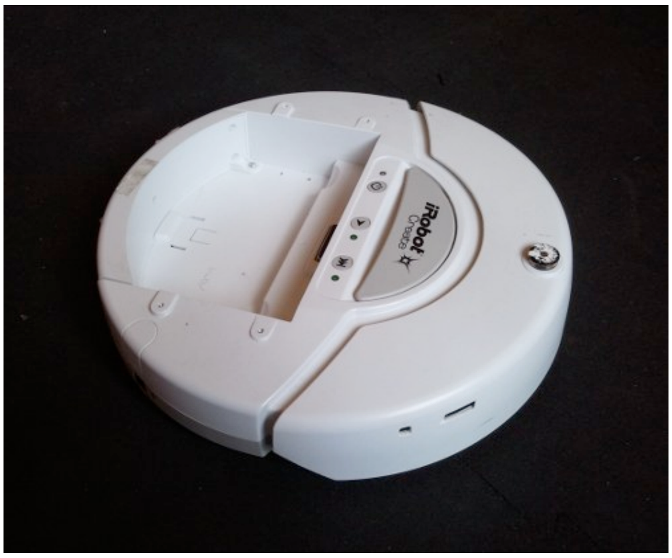
Fig. 3. iRobot Create [7].

Simple maneuverability tests were conducted, such as commanding the robot to continuously travel in a square pattern. Next, mobile operations and sensor commands were combined in slightly more complex tests including commanding the robot to turn clockwise slightly every time it sensed the front bumper sensor being pressed. In this way, the robot would make its way around the outer perimeter of an environment. Figure 4 shows an example of an environment traversed using this method.

---Figure 4---

Under control of Player/Stage, the robot was able to successfully maneuver around the perimeter of this four-walled structure. Each time the robot would come into contact with the outside wall, it would correct its direction slightly clockwise and then continue moving forward. In a few tests, the robot would disconnect from the outer wall briefly because it had rotated too far. This problem was easily solved by reducing the allowed amount of time the wheels would rotate and spin the robot after each bumper sensor reading. With this implemented, the robot followed the perimeter more closely.

Finally, mobile operations, the bumper sensor, as well as the infrared sensor were all tested simultaneously. In this test, the robot was commanded to move forward until it registered either a contact from the bumper sensor or an infrared signal. Once this signal was received, the robot would turn 180° and then continue moving forward. A physical object was placed at one end of the robot's path and an infrared emitter was placed at the other end. In this way, the robot would simply move back and forth along this line from the object to the infrared emitter and then back to the object, etc. Figure 5 shows this environment with the infrared emitter on the right and the physical object on the left.

---Figure 5---

All three of these tests performed as expected, and with satisfactory results.

VIII. CONCLUSIONS

This research endeavor spanned two and a half months. In this time, a viable ROS was found, tested, and validated that could be used by Trinity undergraduates with limited programming experience. These tests were conducted by the first author, an undergraduate in the target user group. With no prior experience working with robot operating systems and with a limited knowledge of programming, this student was able to control a robot with multiple scenarios both in simulation and also physically within a time period of about 25 hours. Due to the success of the tests performed, Player/Stage has been chosen as the best ROS candidate. It is easy to use, capable, adaptable, easy to install and maintain, and well developed.

Obtaining this ideal robot operating system has not directly fulfilled the goals set forth at the beginning of the research in order to benefit the engineering programs at Trinity University. However, this research has set the foundation for these goals to eventually be implemented. In this way, this research has helped to ensure a better engineering experience for future Trinity engineering students.

REFERENCES

[1] "A Platform for Developing Robotic Applications." *Microsoft Robotics Developer Studio 4 Beta*. Microsoft, Sept. 2011. [Online]. Available: http://www.microsoft.com/robotics/Content.aspx?pg=Product [Accessed: 18 Nov. 2011].

[2] "AnyKode Marilou - Modeling and Simulation Environment for Robotics." ANYKODE, 2011. [Online]. Available: http://www.anykode.com/index.php [Accessed: 18 Nov. 2011].

[3] Brooks, Alex, Tobias Kaupp, Alex Makarenko, and Michael Moser. "Orca: Components for Robotics." *Orca Robotics*. [Online]. Available: http://orca-robotics.sourceforge.net/ [Accessed: 18 Nov. 2011].

[4] *CARMEN*. CARMEN-Team. [Online]. Available: http://carmen.sourceforge.net/home.html [Accessed: 18 Nov. 2011].

[5] "DROS." *Dave's Robotic Operating System*. [Online]. Available: http://dros.org/ [Accessed: 18 Nov. 2011].

[6] "INSTALLING PLAYER-STAGE : WHAT WORKED FOR ME." *Mobotica*. 29 Mar. 2010. [Online]. Available: http://mobotica.blogspot.com/2010/03/installing-player-stage-what-worked-for.html [Accessed: 18 Nov. 2011].

[7] Kirbis, David S. "Arduino Controlled IRobot Create Computer Vision Cinema." *Computer Vision Cinema*. 14 June 2011. [Online]. Available: http://cvcinema.blogs.upv.es/2011/06/14/arduino-controlled-irobot-create [Accessed: 18 Nov. 2011].

[8] "MARIE: Mobile and Autonomous Robotics Integration." *MARIE*. 2009. [Online]. Available: http://marie.sourceforge.net/wiki/index.php/Description [Accessed: 18 Nov. 2011].

[9] M. Reichardt, L. Wilhelm, M. Proetzsch, and K. Berns. "Applications of Visualization Technology in Robotics Software Development," Germany, University of Kaiserslautern, Department of Computer Sciences.

[10] "Microsoft Robotics Developer Studio." Microsoft, 2011. [Online]. Available: http://www.microsoft.com/robotics/ [Accessed: 18 Nov. 2011].

[11] Milenković, Milan. *Operating Systems: Concepts and Design*. New York: McGraw-Hill, 1987.

[12] "MOOS: Cross Platform Software for Robotics Research." *Robotics Research Group Home Page*. Oxford Mobile Robotics Group. [Online]. Available: http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php [Accessed: 18 Nov. 2011].

[13] "MRPT: Mobile Robot Programming Toolkit." *The Mobile Robot Programming Toolkit*. 2011. [Online]. Available: http://www.mrpt.org/ [Accessed: 18 Nov. 2011].

[14] "Orocos." *The Orocos Project | Smarter Control in Robotics & Automation*. [Online]. Available: http://www.orocos.org [Accessed: 18 Nov. 2011].

[15] "Player/Stage." *The Player Project: Free Software Tools for Robot and Sensor Applications*. 2010. [Online]. Available: http://playerstage.sourceforge.net/ [Accessed: 18 Nov. 2011].

[16] "Player/Stage." *Tutorials*. 2010. [Online]. Available: http://playerstage.sourceforge.net/doc/Player-3.0.2/player/group__tutorials.html [Accessed: 18 Nov. 2011].

[17] Rich, C., B. Ponsler, A. Holroyd, and C. L. Sidner. "Recognizing Engagement in Human-robot Interaction." *Human-Robot Interaction (HRI), 5th ACM/IEEE International Conference* (2010): 375-82. *IEEE Xplore*.

[18] "RIK: Robot Intelligence Kernel." Idaho National Laboratory, 2011. [Online]. Available: https://inlportal.inl.gov/portal/server.pt/community/robot_intelligence_kernel/457 [Accessed: 18 Nov. 2011].

[19] "Robotic Machine Vision Software." *RoboRealm: Vision for Machines*. 2005. [Online]. Available: http://www.roborealm.com/index.php [Accessed: 18 Nov. 2011].

[20] "ROS: Robot Operating System." *ROS.org*. [Online]. Available: http://www.ros.org/wiki/ [Accessed: 18 Nov. 2011].

[21] Silberschatz, Abraham, and Peter B. Galvin. *Operating System Concepts*. Reading, MA: Addison Wesley Longman, 1998.

[22] Singhal, Mukesh, and Niranjan G. Shivaratri. *Advanced Concepts in Operating Systems: Distributed, Database, and Multiprocessor Operating Systems*. New York: McGraw-Hill, 1994.

[23] Sklar, Elizabeth, Simon Parsons, and Susan Epstein. "Developing a Framework for Team-based Robotics Research." [Online]. Available: www.cs.hunter.cuny.edu/~epstein/papers/sklar-parsons-et-al3.pdf [Accessed: 18 Nov. 2011].

[24] S. Cook, T. Mansell, Q. Do, P. Campbell, P. Relf, and S. Shoval, Stephen, "Infrastructure to Support Teaching and Research in the Systems Engineering of Evolving Systems," *7th Annual Conference on Systems Engineering Research, 2009.*

[25] "Urbi." *UrbiForge Main/Home Page*. UrbiForge, 2011. [Online]. Available: http://www.urbiforge.com/ [Accessed: 18 Nov. 2011].

[26] "Webots 6." *Webots: Robot Simulator*. Cyberbotics: Professional Mobile Robot Simulation. [Online]. Available: http://www.cyberbotics.com/ [Accessed: 18 Nov. 2011].

[27] "YARP: Yet Another Robot Platform." 2011. [Online]. Available: http://eris.liralab.it/yarpdoc/index.html [Accessed: 18 Nov. 2011].