AN ABSTRACTION LAYER FOR CONTROLLING HETEROGENEOUS MOBILE CYBER-PHYSICAL SYSTEMS

THESIS

Presented to the Graduate Council
of Texas State University-San Marcos
in Partial Fulfillment
of the Requirements

for the Degree

Master of SCIENCE

by

Trevor R. Hanz, B.B.A.

San Marcos, Texas
May 2013

AN ABSTRACTION LAYER FOR CONTROLLING HETEROGENEOUS MOBILE CYBER-PHYSICAL SYSTEMS

Committee Members Approved:

Mina S. Guirguis, Chair

Anne H.H. Ngu

Yijuan Lu

Approved:

J. Michael Willoughby
Dean of the Graduate College

ACKNOWLEDGEMENTS

I would like to thank Dr. Mina S. Guirguis for his guidance and support without which I would not be involved in research. I would like to thank Dr. Anne H.H. Ngu for her encuragement that lead me to do research. I would also like to thank Dr. Yijuan Lu for her invaluable technical expertise. Finally, I wish to thank my familty for their support throughout my graduate studies.

This manuscript was submitted on April 29, 2013.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Cyber-Physical Systems (CPSs) are systems that closely integrate computation with their physical environments through various communication mediums. Mobile CPSs is a subset of CPSs in which a subset of their components are mobile. Due to recent advances in wireless networking and embedded systems, mobile CPSs are emerging as powerful systems in various areas such as autonomous vehicles and swarm robotics. For example, many mobile CPS applications have been developed for exploration [2; 7], border control [3; 8; 9], and search and rescue operations in land [12; 18], water [13], and air [6; 19].

Many mobile CPS applications rely on devices being aware, to some extent, of their position in the physical world. While in some cases, the position can be obtained through external localization services (e.g., GPS or infrared), such systems may not be present and may not work in specific environments. Thus, devices must rely on sensors that are typically attached to their wheels and drive train (or accelerometers and gyroscopic sensors in the case of aerial devices). Moreover, these devices are typically equipped with various infrared, ultrasonic and visual sensors that are used to detect distances to physical objects, helping the devices to navigate and generate their relative positions in their environments. As one would expect various mobile devices are equipped with different types of sensors, capabilities and mobility models.

Currently the development of mobile CPS applications face a daunting portability challenge: applications designed (and implemented) using a particular hardware platform will not be able to run on another. This is due to the high dependency on the configuration of the hardware that the system is implemented on. For example, developing a robotics application that can have a robot explore an area will depend on the type, dimensions, capabilities, and the physical model of the robot. If one were to switch to a new robot, the application must be redesigned and re-implemented. Although the new hardware must be taken into account, we argue that the high-level application may not need to change much to accommodate the new hardware. Since many tasks can be broken down into simple movement, rotation and sensing commands, the high level application can remain largely intact. Underneath, however, we must provide a mean to accommodate a heterogeneous set of mobile devices.

Devising control mechanisms for Mobile CPSs that are portable from a particular platform to another is a complex problem. It requires providing proper levels of abstractions that allow high level design and implementation to be reused, while supporting mapping common functionalities to different hardware platforms.

In this thesis, we present a framework for managing mobile CPSs with different hardware configurations that incorporates novel approaches for controlling the devices. The method relies on abstracting the representation of the mobile CPS in a way that can allow the system to fully utilize the capabilities of the hardware while providing a simplified interface to the end-user. We have built a prototype of this framework as a proof-of-concept with three main components. Support for components that are not described in this thesis may be easily built on top of this framework. The first component is a motion control system that demonstrates the abstract nature of the control system. It uses the abstract representation of a mobile CPS to devise the best method to move devices to achieve the desired result. It also incorporates a physics engine for different hardware platforms. The second component is remote control application that utilizes the motion control systems that allow the mobile CPSs to be controlled remotely. This component is implemented as an Android application and is intended to demonstrate the capabilities of the motion control system as well as explore new and unique methods of controlling a MCPS. The third component is a visualizer that creates visual representation of what the control systems perceives of the current state of the system.

Thesis organization: This thesis is organized as follows: Section 2 describes related work. In Sections 3, 4, and 5, we discuss the three components of our control system, Core Framework, Physics Layer and Motion Control Layer respectfully. In Section 6, we discuss a novel input system. In Section 7, we discuss our method for testing and verifying our Physics Layer. The performance results of our control system are available in Section 8 followed by our conclusion and future work in Section 9.

CHAPTER II

RELATED WORK

To understand the challenges we had to face while developing my system, it is necessary to understand the enormous variance in technologies used in CPSs. A lot of research has been spent on finding the position of a CPS and its relation to other objects. J. Borenstein, H.R Everett, L.Feng, and D.Wehe [1] define seven types of positioning systems.

1. Odometry - This is the most widely used method for determining a mobile robot's position. It calculates the new position by adding the estimated change in its position from its movements to its last known position. This method is easy to calculate, but can be inaccurate. Since this method is based on accumulated estimates, inaccuracy increases over time.

2. Inertial Navigation - Similar to odometry except that it utilizes accelerometers and gyroscopes to calculate change in position, this method benefits from not being based on relationships with objects in the physical world, thus it will provide accurate data even if the device was moved by an external force. Gyroscopes can be used to correct rotational inaccuracy in odometry measurements.

3. Magnetic Compasses - This method utilizes a magnetic compass to

calculate the mobile agent's heading. Compasses can also be used to correct inaccuracies in odometry rotational measurements. Unfortunately, compasses readings can be distorted by power lines or large steal structures.

4. Active Beacons - This is a highly reliable and simple method to implement that requires the use of stationary transmitting or receiving beacons at a known location to calculate the mobile agent's absolute position. These systems usually require line of sight to the beacon to function properly, thus it is a popular method for navigation of planes and boats.

5. Global Positioning Systems - Similar to the active beacon methods, Global Positioning Systems, or GPS, uses signals from orbital satellites to calculate its latitude and longitude. A standard commercial GPS has an accuracy within 20 meters, thus it isn't practical for small mobile agents. The accuracy can be improved using Differential GPS (DGPS) to an accuracy of about 10 cm by using multiple land based stations that monitor and transmit the difference between its known location and that reported by the satellites. As with active beacons, GPS needs a line of sight to take readings, so it won't work indoors.

6. Landmark Navigation - This method utilizes sensor data to find unique static objects in the physical wold at a known position to assess its own position. Cameras are often used for the purpose of detecting the landmarks and may require additional processing power compared to other methods.

7. Model Matching - Similar to landmark navigation, model matching attempts to calculate its position based on its perception of the physical world. This method compares a map of the area that it already knows about to what it currently perceives the map of the area to be in order to evaluate its position on the known map. Sometimes the map has to be generated by the mobile robot before the map data can be matched.

While these are the more common methods for calculating position, there are others. One in particular [11] incorporates the use of multiple mobile robots equipped with laser scanners to detect movement. This system works by having only one robot moving at a time and using odometry data to calculate the position of each stationary robot. With the collected data from each of the stationary robots the position of the moving robot can be found. Unfortunately, this method only calculates position and not orientation.

Other methods [16] however do cover orientation. Shraga Shoval and Johann Borenstein demonstrate that relative position and orientation can be calculated using binaural sonar. By placing an ultrasonic transmitter and two ultrasonic receivers 350mm apart on one edge of an agent, it is possible to calculate the relative position and orientation of both agents using the phase difference of the sound waves received at the opposite ends. Unfortunately, this method requires that the receivers are facing the transmitter on the other robot, so it is an impractical method unless you place enough sensors so that at least two receivers are always facing the transmitter. However, this method does yield itself to be useful for aiding in alignment in simpler contexts such as coordinating two agents in close proximity where their rough position and orientation is already known.

In an experiment [15], another method for determining relative positioning using ultrasonic waves was introduced. For this experiment, three ultrasonic receivers were spread equally apart, angled directly upwards and had mounted above it a cone to direct sound into the receiver from any direction. By measuring the relative times that it took sound to be received by each receiver, they where able to calculate the relative position of the transmitter. They found the system to be accurate to about 8mm and 3 degrees.

In order to manage this wide variety of hardware, a Robotic Operating System(ROS) is needed. A study presented by John Kerr and Keven Nickels [4] gives an overview of the available ROSs including Microsoft's Robotics Developer Studio (RDS), Player/Stage, and the open source alternative that is named after what it is, ROS. This study started out surveying 16 different ROSs based on ease of use, capability, adaptability, ease of maintenance, and software maturity. The three mentioned above scored the highest in their study.

ROS is a very robust system with many modules and drivers already developed including a driver for the roomba. This driver provides many convenience functions such as functions to gather odometry information and to send movement commands using linear and angular velocity.

Using an android smart phone to remotely control a robot has also been attempted. In this experiment [10], a robotic agent was build specifically to accept the Android OS as its control software. Further more, the agent received commands from a remote Android smart phone or tablet and the agent uses the hand held device to display image data from its camera. This uses the built in Android functions and features, such as camera drivers, to demonstrate the flexibility of the Android OS.

After the OS of the CPS has been installed, it is necessary to get the CPS to perform some action. Often multiple robotic agents will work together to perform a single action. A Web-of-Things framework [17] was proposed to solve common problems with coordinating real-time multi-agent CPSs. This framework allows different agents to process events and possibly relay those events over the Internet to other agents for additional processing. In addition, users can monitor and send commands to these CPSs through uses of the web.

Often when a multi-agent CPS becomes too large or impractical to manage each agent, a robotic swarm is needed. A defining characteristic of robotic swarms is the presence of emergent behavior, the appearance of a globally organized behavior arising from local interaction between individual agents. Movements of individual agents in robotic swarms are usually modeled after physics simulations (physicomimetics) of natural biological systems(biomimetic) [14]. One experiment [5] shows a practical use of robotic swarms to sort randomly distributed boxes. The boxes are colored either blue or green. Two beacons are placed, one as the drop zone for each color. Each agent is able to operate autonomously to complete the task.

CHAPTER III

CORE FRAMEWORK

Since it is unreasonable for us to develop a system that can support all known capabilities, we have picked a subset of the more common abilities as a demonstration of the system. We have chosen to support an MCPS's ability to understand its location and move to a new location. To support this we have implemented a system for odometry and an interface to the hardware to enable movement. It is provided by two components that are called the Physics Layer and Motion Control Layer. More details about these components can be found in the sections 4 and 5 respectively.

While it is desirable to create a system with a simplified high level interface, we did not want to remove any possible functionality from the MCPS. To accomplish this, we decided to use a layered architecture as depicted in figure 3.1. Each layer provides an additional level of abstraction from the layer below. There is no forced hierarchy so if a higher layer doesn't provide all the necessary functionality, a component may probe a lower layer to gather more detailed information. If possible, this should be avoided as lower levels tend to contain more system specific information which will generally make the accessing component less portable.

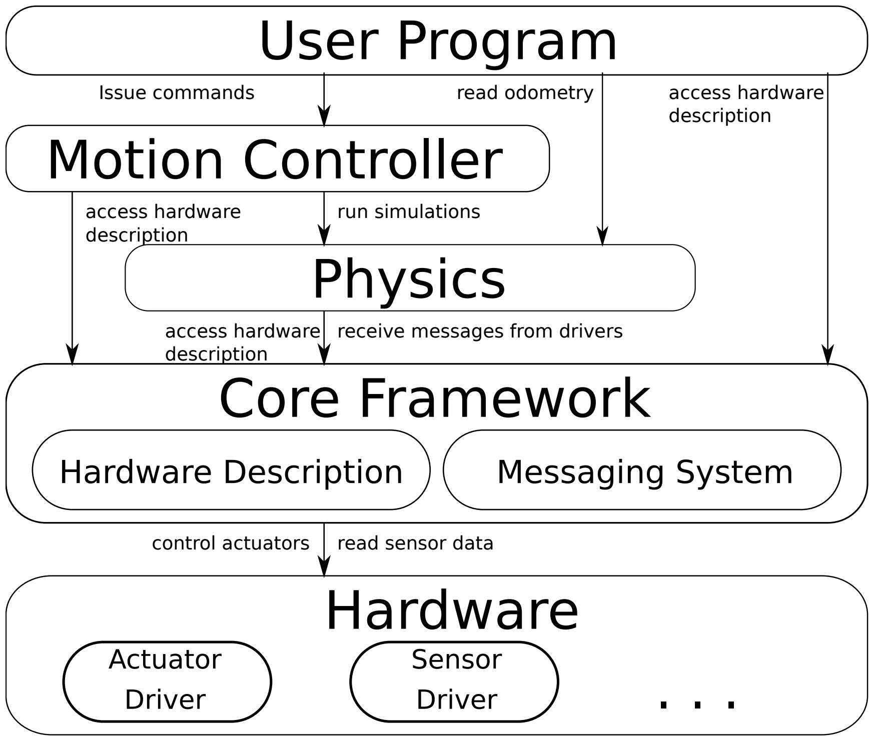

Figure 3.1: Layered system architecture

This is also how the control system achieves easy expandability. New components can be added to the system without affecting the existing components. A new component may access any of the other already existing components and in turn be accessed by other components to become a part of the system.

The lowest layer to which all components are, at least indirectly, connected is the core framework. The core framework contains the system specific information. It can be thought of like an operating system for the MCPS. Its job is to manage all known data about the hardware of the MCPS as well as facilitate access to the drivers for the actuators and sensors. Additionally, the core framework provides a system for message passing between components.

The core framework manages data in a structured object-oriented manner. All data about the MCPS is stored in objects that symbolically represent the physical components of the MCPS. The components are divided into three categories: structural component, sensor or actuator. Structural components define a physical shape such as a wheel or the chassis. They take parameters such as size, shape and weight. Sensors and actuators can be thought of as the input and output devices. Both of them provide access to the specific drivers necessary to utilize the hardware of the MCPS. Sensors are periodically polled by the core framework at which point the sensor may broadcast a message about its current state to any listening process. Actuators can take a single floating point value normalized between -1.0 and 1.0. This value represents the desired state of the actuator. In the case of an electric motor, -1.0 to 1.0 would relate to full reverse to full forward respectfully. It is done this way so that users of the actuators don't need to know the potential of the actuator to use its greatest potential. Whenever the actuator's state has changed, it should broadcast a message informing all listening processes about the change.

The components are then structured logically. The top level component is a structural component that represents the chassis and is referred to as the Agent. Some components, such as the Agent, may contain any number of other components allowing some components to be nested to any level. In addition to the properties mentioned earlier, all components contain a three dimensional position relative to their parent component with the Agent having an implicit position at the origin. It is structured this way to help other systems to better understand the relationship between components and the structure of the MCPS.

CHAPTER IV

PHYSICS LAYER

So far, a low level abstraction for the hardware has been established with the core framework. The next layer of our control system is the Physics layer. This layer provides a system for handling position and odometry measurements.

Odometry is often simplified for the specific hardware it is implemented on. For instance, the odometry calculations for a roomba is depicted in equations 4.1, 4.2 and 4.3 where θ is the roomba's rotation and W_{Right} and W_{Left} is the rotated distance of the roomba's right and left wheels respectfully.

$$\theta_k = \theta_{k-1} + \frac{\Delta(W_{Right}) - \Delta(W_{Left})}{(AxleLength)} \tag{4.1}$$

$$X_k = X_{k-1} + \frac{\Delta(W_{Right}) + \Delta(W_{Left})}{2} \times \cos\theta_k \tag{4.2}$$

$$Y_k = Y_{k-1} + \frac{\Delta(W_{Right}) + \Delta(W_{Left})}{2} \times \sin\theta_k \tag{4.3}$$

Since these equations won't work on other hardware, we used a more general system for calculating odometry. We use the information provided by the core framework to create a physics model. From there we sum up all the forces and torques on each physical component and apply the constraints enacted on the physics model by the joints and the collision with the ground or other objects resulting in a new force and torque vectors.

$$F_{total} = \sum F \tag{4.4}$$

$$\tau_{total} = \sum \tau \tag{4.5}$$

The new force and torque vectors can then be used in the calculation of the new linear and angular velocity.

$$v_k = v_{k-1} + \frac{F_{total}}{m} \times t \tag{4.6}$$

$$\omega_k = \omega_{k-1} + \frac{\tau_{total}}{m} \times t \tag{4.7}$$

CHAPTER V

MOTION CONTROL LAYER

So far as described, the control system is useful for providing information but it has no facilities to control the movements of the MCPS. For this reason, we developed the motion control layer. This system provides an additional level of abstraction on top of the physics layer and the core framework. It creates a high level interface to the actuators of the MCPS so that users do not have to understand anything about the hardware of the MCPS to be able to control its movements.

This layer takes two three dimensional vectors representing linear and angular velocities as input and makes a decision on how best to move the available actuators in order to achieve the desired linear and angular velocities. The result of its calculations is an animation file describing the actuators to be moved and in what direction.

To create the animation file the system performs a linear regression using a simple batch gradient decent algorithm on each actuator. The batch gradient decent results in a model depicted in equation 5.1 where X represents a six dimensional feature vector containing the desired linear and angular velocities and θ represents the correlation between the state of an actuator and the corresponding feature. This model must be repeated for each actuator after calculating a different θ vector for each actuator resulting in a model depicted by equation 5.2 where n is the number of actuators.

$$h(x) = \sum_{i=0}^{6-1} \theta_i \times X_i \tag{5.1}$$

$$\begin{array}{c} 0 \\ \vdots \\ n-1 \end{array} \begin{pmatrix} \sum_{i=0}^{6-1} \theta_{i,0} \times X_i \\ \vdots \\ \sum_{i=0}^{6-1} \theta_{i,n-1} \times X_i \end{pmatrix} \tag{5.2}$$

To calculate the θ vector for each actuator, the gradient decent algorithm must first gather a data set. To uphold the current level of abstraction, the physics engine's simulator is used to produce the data set by retrieving the linear and angular velocities for every possible combination of actuator states with each actuator using a state space of $\{-1, 0, 1\}$.

$$\left.\begin{array}{ccc} a_{0,0}, & \ldots, & a_{n-1,0} \\ \vdots & \ddots & \vdots \\ a_{0,3^n-1}, & \ldots, & a_{n-1,3^n-1} \end{array}\right| \begin{array}{ccc} X_{0,0}, & \ldots, & X_{5,0} \\ \vdots & \ddots & \vdots \\ X_{0,3^n-1}, & \ldots, & X_{5,3^n-1} \end{array} \tag{5.3}$$

The resulting data set is depicted in 5.3 where a represents actuator states, Xrepresents the feature vectors and n represents the number of actuators in the system.

CHAPTER VI

MOBILE INPUT SYSTEM

In order to demonstrate the capabilities of our software, we developed a mobile input system. This system allowed us to give commands to the control system remotely. This is accomplished with two components: an Android app and a receiving daemon.

The Android app uses the large touch screen real estate of an Android device to display a customizable user interface. Preloaded on the device is an XML file that is used to define the layout of the user interface allowing the interface to be easily changed by reloading a new XML file. The user interface of this app consists of virtual buttons and joysticks referred to as controls. Each control is related to a key value that is synced with the receiving daemon.

The receiving daemon is the small companion program to the Android app. It is run as a daemon process on a Linux system, in this case the MCPS. Its purpose is to receive changes in key value pairs from the app and relay the pertinent information to requesting processes on the host machine. Additionally the processes on the host machine can set key value pairs through the daemon process that are then synced with the app thus allowing two way communication. This process helps make the mobile input system a versatile tool when working with MCPSs.

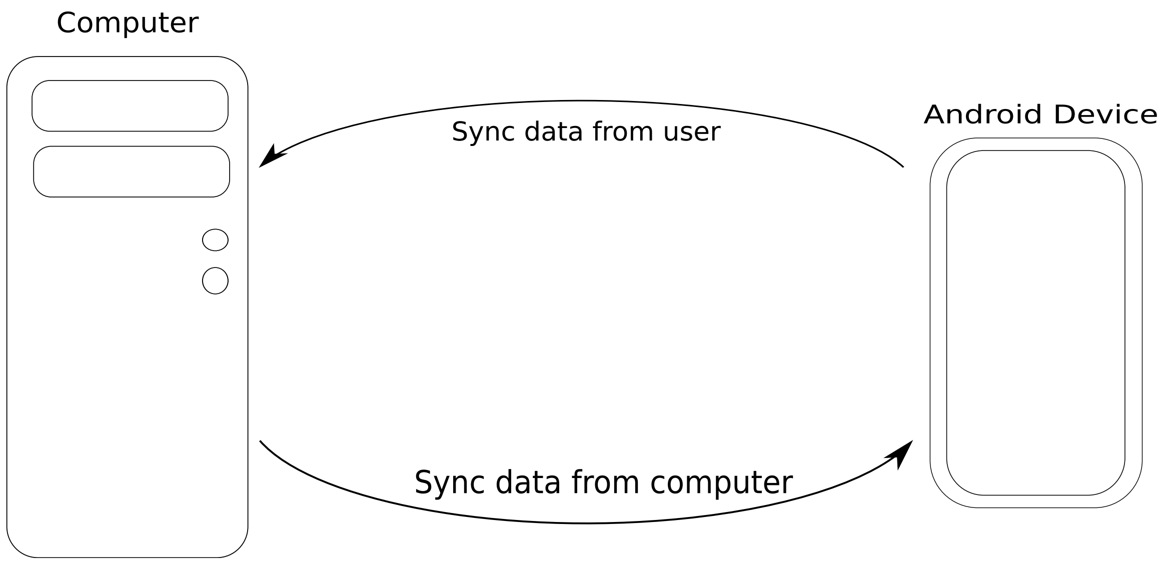

Figure 6.1: Mobile app relationship with computer

The receiving daemon also includes a plug-in system to facilitate the communication between itself and other processes. In this case, the control system has a companion plug-in that is loaded by the receiving daemon. The plug then relays this information to running thread in the control system that uses the information to pass commands to the Motion Controller.

CHAPTER VII

VISUALIZER

With a system as complex as ours it is necessary to have a proper method for verification. To accomplish this, we developed the Visualizer. The visualizer displays what the control system perceives its environment to be.

In figure 7.1, you can see a demonstration of the visualizer. The gray cylinder is the MCPS, in this case a roomba, with a red arrow painted on top to indicate its facing direction.

This system starts by accessing the core framework to gather agent IDs. These IDs are then used when accessing the Physics layer to get the odometry information. With the odometry information, it can then place a representation of the MCPs at that location in the visualizer display. That process is then repeated for each frame of the display to ensure accurate correlation to the odometry data.

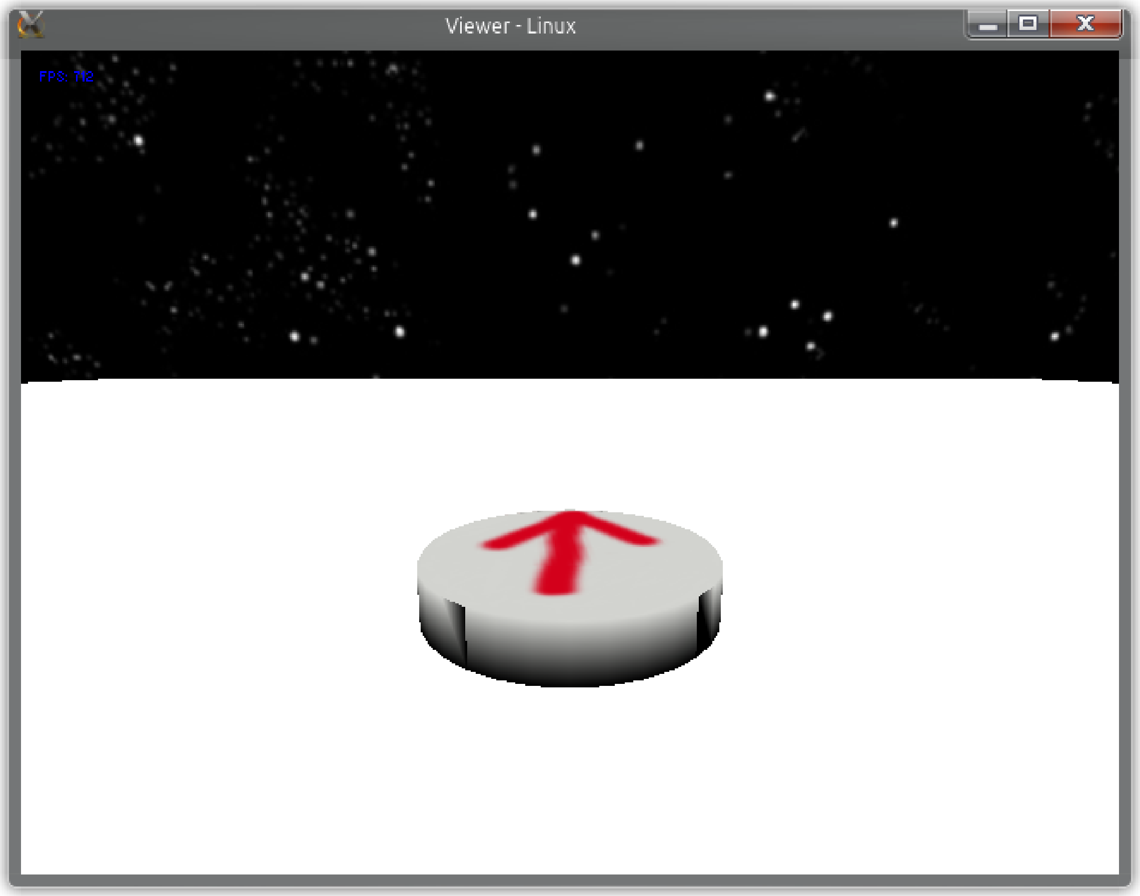

Figure 7.1: Visualizer demonstration

CHAPTER VIII

PERFORMANCE EVALUATION

In order to evaluate the performance of our system, we implemented our tests on a roomba and compared the results with that of an equivalent program we wrote using the roomba's own Open Interface api. For each test, the roomba was told to preform some action at half of its maximum speed and the systems odometry was used to determine when the desired action was complete.

The first test measures the accuracy of the odometry on the respective system when moving in a straight line. We placed the roomba at a marked location and requested the roomba to travel for certain number of cm. After the program determined it had traveled that distance, it stops and the real traveled distance was measured which is then subtracted from the desired distance to get the offset. Tests were divided into increments of 10cm where each test was run five times and the average was taken. Figure 8.1 shows the results of this test.

As you can see, the Open Interface maintains a very steady offset where the roomba traveled about 1.5 cm further than it was told to travel where as our system achieved more accurate results at shorter distances but the performance gradually declines. After about 60cm of traveled distance our accuracy is equivalent to the

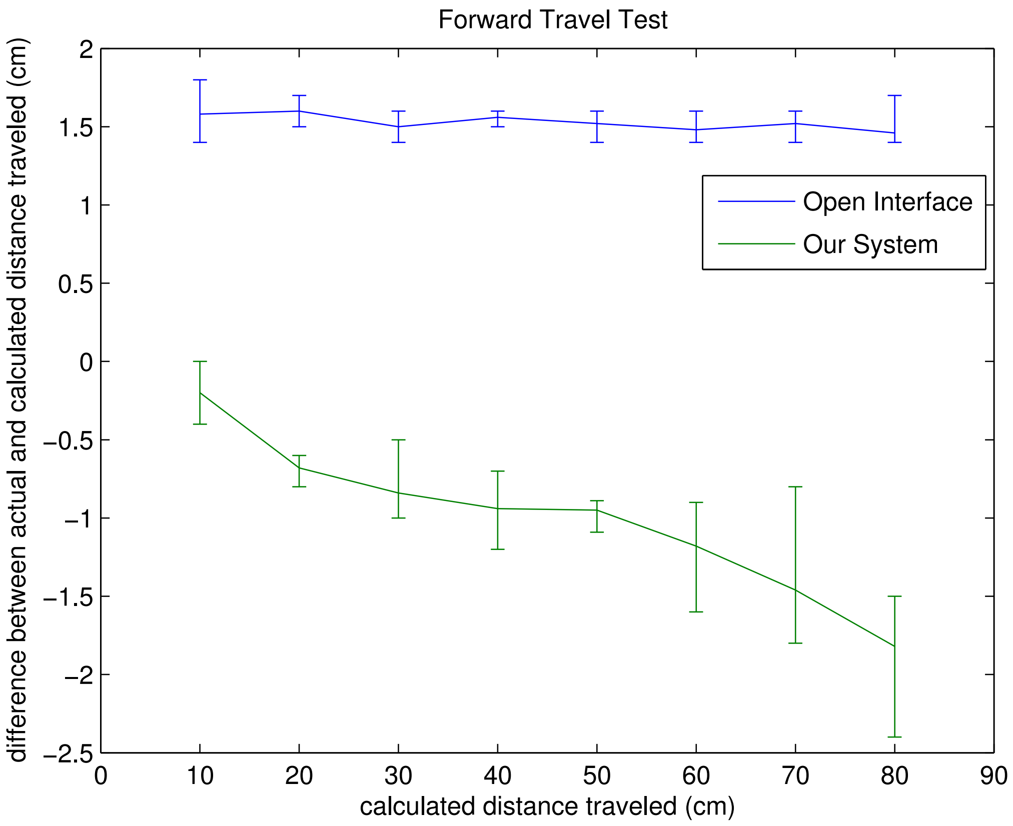

Figure 8.1: Difference between odometry and actual traveled distance

Open Interface. This relationship is emphasized in figure 8.2 where we took the results and divided them by the total distance traveled to get the relative offset.

In the next test, we compare the ability of the systems to accurately measure the speed of rotation. For each system, we tell the roomba to rotate in place for a specified number of complete rotations and measure the offset from the desired angle. For each number of rotations, we run the test five times and take the average. Figure 8.3 shows the results of this test.

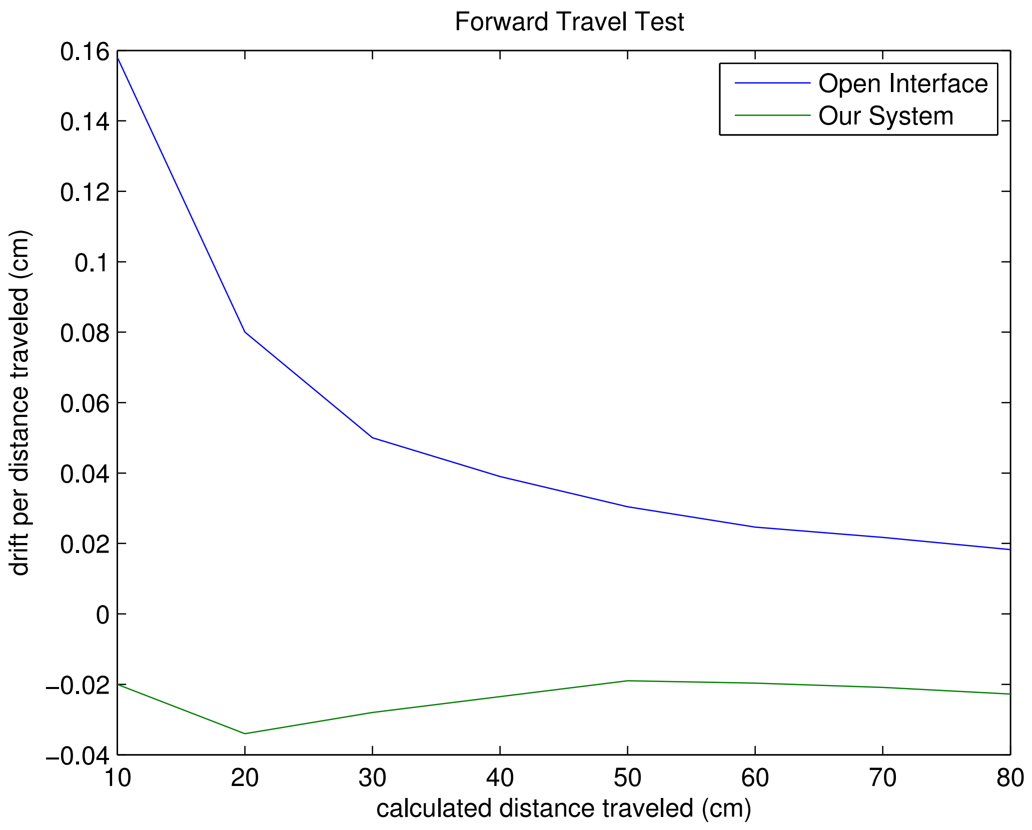

Figure 8.2: Relative distance offset

As you can see from this test, both systems become less accurate the longer they are requested to rotate though our system maintains a higher accuracy. To analyze this result further, we divided the offset by the distance it has rotated. These results are displayed in figure 8.4.

The two systems approach similar accuracies over time. Our system experiences a decrease in accuracy as it approaches an accuracy of .025 degrees off per degree while the Open Interface solution an improved accuracy as it approaches

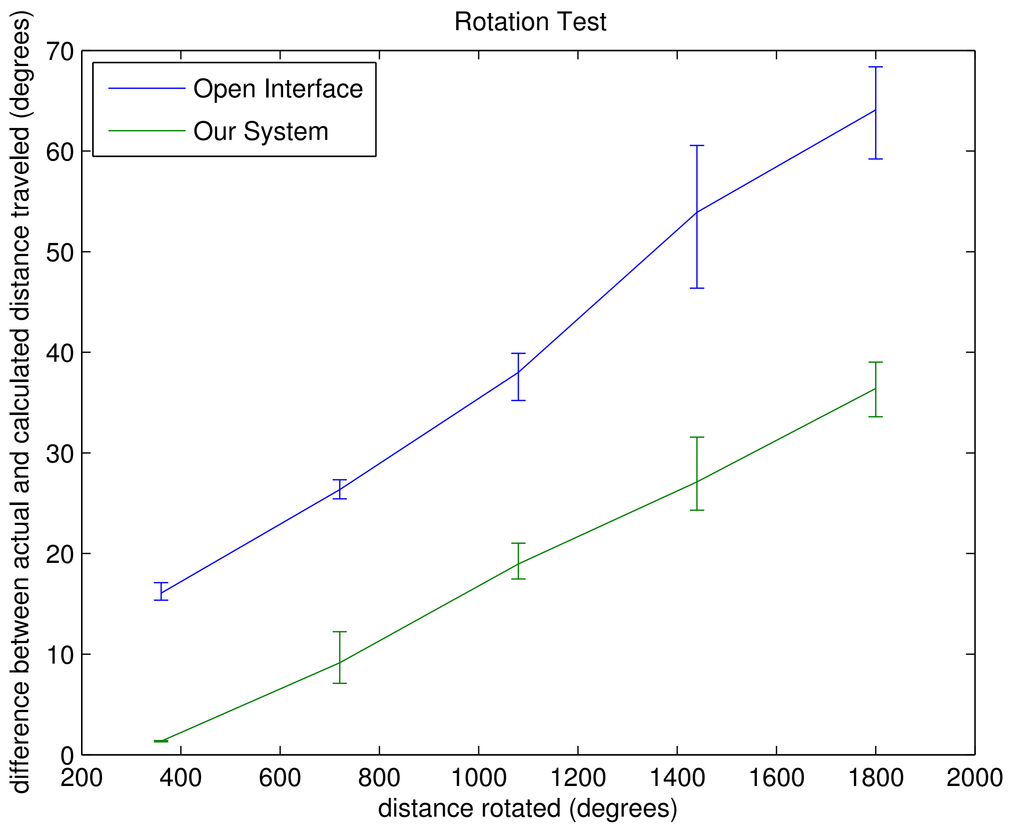

Figure 8.3: Difference between odometry and actual rotated distance

an accuracy of .034 degrees off per degree.

We also tested the ability of this control system to be implemented on different MCPSs. Unfortunately, due to resource restrictions we could not acquire separate hardware for this test. To conduct this test, we altered portions of the roomba driver in order to emulate a different MCPS. We limited the motor drivers by rejecting commands to move backwards thus changing the movement pattern of the roomba. To test the proper handling of different hardware, we ran an identical

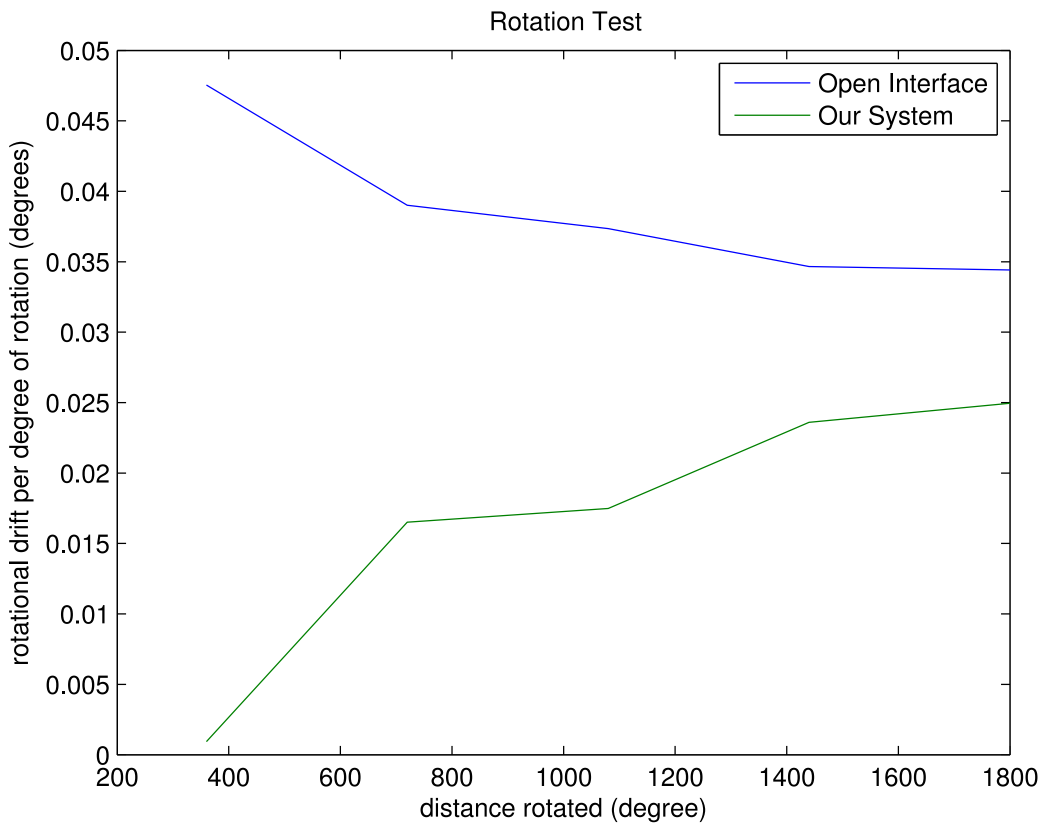

Figure 8.4: Relative rotation offset

program on both systems. The program commanded the roomba to travel straight for 50cm, stop, turn right 90 degrees, and repeat until the roomba has made a full circle. The notable change between these two system is that the roomba with full movement can turn by pivoting around its center while the roomba with limited drivers must take a wider turn by pivoting around one of its wheels.

We ran this program on both configurations five times and measured the distance that the final position differed from the initial starting position. The results of the two configurations were very similar. The unaltered driver produced an average inaccuracy of .98cm with a max and min inaccuracy of 1.82cm and .32cm respectively. The limited driver produced an average inaccuracy of .69cm with a maximum of 1.06cm and a minimum of .57cm. The limited driver produced a more accurate average while the unaltered driver produced a smaller minimum inaccuracy.

CHAPTER IX

CONCLUSION

We demonstrated a method for controlling a MCPS at a high level abstracted from the hardware as well as a few novel approaches for controlling our system. The control system can help speed up the development of new robotics systems and it has been proven to be more accurate than current methods in certain scenarios.

While our current work is a great first step, there is more that could be done in future work. The physics engine requires a high degree of fine tuning to reach a high level of accuracy. This process could be partially automated given a set of test data and a proper set of machine learning algorithms. A proper automated tuning feature could potentially yield a greater accuracy than was demonstrated in this thesis. Coupling the visualizer with the simulator component of the control system could yield a handy prototyping tool. This could allow developers to design and test a potential hardware design before they build it thus saving developers lots of time and money. For the Mobile input system, we would like to add the ability for the Android app to receive images. This ability would allow us to send a visual representation of the environment as perceived by the control system thus giving us new ways to interact with the control system.

BIBLIOGRAPHY

[1] J. Borenstein, H. Everett, L. Feng, and D. Wehe. Mobile robot positioning âĂŞ sensors and techniques. *Journal of Robotic Systems*, 14, April 1997.

[2] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. Collaborative Multi-Robot Exploration. In *Proceedings of IEEE International Conference on Robotics and Automation*, 2000.

[3] T. Fong, C. Thorpe, and C. Baur. Multi-robot Remote Driving with Collaborative Control. *IEEE Transactions on Industrial Electronics*, 50(4):699–704, 2003.

[4] J. Kerr and K. Nickels. Robot operating systems: Bridging the gap between human and robot. In *44th Southeastern Symposium on System Theory*. IEEE, 2012.

[5] A. Kettler and H. Worn. Sorting boxes with a robotic swarm: An analysis by means of experiment, simulation and model. In *International Conference on Robotics and Biomimetics*, December 2011.

[6] J. Kim and Y. Kim. Moving Ground Target Tracking in Dense Obstacle Areas Using UAVs. In *Proceedings of the 17th IFAC World Congress*, Seoul, South Korea, 2008.

[7] W. H. M. Zapata, N. Kannen, M. Sullivan, and J. Conrad. An Autonomous Vehicle for Space Exploration. In *Proceedings of IEEE Southeastcon*, Huntsville, AL, 2008.

[8] A. Marino, F. Caccavale, L. Parker, and G. Antonelli. Fuzzy Behavioral Control for Multi-Robot Border Patrol. In *Proceedings of the 17th Mediterranean Conference on Control and Automation*, Thessaloniki, Greece, June 2009.

[9] A. Marino, L. Parker, G. Antonelli, F. Caccavale, and S. Chiaverini. A Modular and Fault-Tolerant Approach to Multi-Robot Perimeter Patrol. In *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Guilin, China, December 2009.

[10] S. W. Moon, Y. J. Kim, H. J. Myeong, C. S. Kim, N. J. Cha, and B. H. Kim. Implementation of smartphone environment remote control and monitoring system for android operating system-based robot platform. In *8th International Conference on Ubiquitous Robots and Ambient Intelligence*, Incheon, Korea, November 2011.

[11] M. Moors, F. Schneider, and D. Wildermuth. Relative position estimation in a group of robots. In *Proceedings of the 6th International Conference on Climbing and Walking Robots (CLAWAR)*, pages 983–990, 2003.

[12] R. Murphy. Rescue Robotics for Homeland Security. *Communications of the ACM*, 47(3), 2004.

[13] H. Okada, T. Iwamoto, and K. Shibuya. Water-Rescue Robot Vehicle with Variably Configured Segmented Wheels. *Journal of Robotics and mechatronics*, 18(3):278, 2006.

[14] W. Othman, B. Amavasai, S. McKibbin, and F. Caparrelli. An analysis of collective movement models for robotic swarms. In *International Conference on "Computer as a Tool"*. IEEE, 2007.

[15] F. Rivard, J. Bisson, F. Michaud, and D. Le'tourneau. Ultrasonic relative positioning for multi-robot systems. In *IEEE International Conference on Robotics and Automation*. IEEE, 2008.

[16] S. Shoval and J. Borenstein. Measuring the relative position and orentation between tow mobile robots with binaural sonar. In *International Topical Meeting on Robotics and Remote Systems*, Seattle, Washington, March 2001.

[17] J. Singh, O. Hussain, E. Chang, and T. Dillon. Event handling for distributed real-time cyber-physical systems. In *15th IEEE International Symposium*, 2012.

[18] D. Stormont. Autonomous Rescue Robot Swarms for First Responders. In *Proceedings of the Computational Intelligence for Homeland Security and Personal Safety*, Orlando, FL, April 2005.

[19] U. Zengin and A. Dogan. Real-Time Target Tracking for Autonomous UAVs in Adversarial Environments: A Gradient Search Algorithm. In *Proceedings of the 45th IEEE Conference on Decision and Control*, San Diego, CA, December 2004.

VITA

Trevor Hanz was born in Austin, Texas, on June 17, 1987, the son of Karl and Jan Hanz. After completing a B.B.A. in Entrepreneurship from the University of North Texas, in 2010, he entered Texas State University-San Marcos. There he pursued a Masters Degree in Computer Science while being employed as a graduate research assistant for the Applied Research Laboratories, The University of Texas at Austin.

Permanent Address: 5908 Travis Woods Cove.
Austin, Texas 78734

This thesis was typed by Trevor R. Hanz.