Virtual Sensor System: Merging the Real World with a Simulation Environment

THESIS

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Michael A. Vernier, B.S.

Graduate Program in Electrical and Computer Engineering

The Ohio State University

2010

Master's Examination Committee:

Professor Ümit Özgüner, Advisor

Professor Yuan Zheng

Abstract

This thesis presents an implementation of a Virtual Sensor System which extended the testbed capabilities of the Control and Intelligent Transportation Research Lab at The Ohio State University. Through this system physical objects including robots and obstacles were modeled in a simulation environment and oriented based on their state in the physical world through the use of an existing Virtual Positioning System. Virtual sensors were able to be utilized by robots without the added cost of purchasing the sensor and creating an interface between it and the robot controller. Other components like a traffic light were added in order to improve the visualization abilities of the simulation environment.

Three sensors were implemented within this system. A laser range finder was created based on a sensor model already included in the underlying simulator. A new model was implemented to provide measurements from a magnetometer. A fictitious lane edge sensor was also designed to show situations where sensor data can be generated independent of constraints in the physical environment.

In order to test the usability of the Virtual Sensor System, two test scenarios were devised. Proportional and proportional-integral controllers were designed independently using the lane edge sensor and a magnetometer array to control a robot as it followed along a simple path. The laser range finder was tested in a small test area containing randomly placed obstacles. Using only the information obtained through this sensor, a

control algorithm was written so that the robot maneuvered through the environment without coming into contact with the obstacles. At the same time, the robot produced a map representation of the explored portions of the test area.

Dedicated to my family and friends

Acknowledgments

First, I would like to thank my advisor, Dr. Ümit Özgüner, for providing the opportunity for me to work with him the past few years. I would not have been able to succeed without his guidance, support, and patience. I am grateful to both Dr. Özgüner and Dr. Yuan Zheng for being on my committee and reviewing this thesis.

I am extremely thankful for everyone I have worked with in the Control and Intelligent Transportation Research Lab, especially Dr. Keith Redmill, Arda Kurt, and Scott Biddlestone. Without their help and patience throughout this project, I would not have been able to accomplish it.

Also, I am very thankful to Dr. Rick Freuler and the Fundamentals of Engineering for Honors program that I have been a part of for so many years. The knowledge that I have gained and the friends that I have made from both going through the program as a student and as a teaching assistant are invaluable to me.

I have taught a lot of students over the years in the Fundamentals of Engineering for Honors program and through both FIRST Robotics and FIRST Lego League. I would like to thank all of them for motivating me with their never ending intelligence and curiosity.

Finally, I would like to thank my parents, Anthony and Betty, for their unending love and patience over the years. Without the supportive environment created by them,

my sister, Amy, my brother, Timothy, and all of my friends, I know that I would not have been able to achieve as much as I have.

Vita

December 29, 1984.....Born — La Junta, Colorado, USA

2007 .....B.S. Electrical and Computer Engineering, The Ohio State University

2007 to present .....Graduate Teaching Associate, The Ohio State University

Publications

[1] M Vernier, C Morin, P Wensing, R Hartlage, B E Carruthers, R Freuler, "Use of a Low-Cost Camera-Based Positioning System in a First-Year Engineering Cornerstone Design Project," in Proceedings of the ASEE 2009 Annual Conference and Exposition, Austin, Texas, 2009.

[2] S. Biddlestone, A. Kurt, M. Vernier, K. Redmill, and Ü. Özgüner, "An Indoor Intelligent Transportation Testbed for Urban Traffic Scenarios," in *Proceedings of the International IEEE Conference on Intelligent Transportation Systems*, St. Louis, Missouri, 2009.

[3] M Vernier, C Morin, P Wensing, R Hartlage, B E Carruthers, R Freuler, "Use of a Low-Cost Camera-Based Positioning System in a First-Year Engineering Cornerstone Design Project," *ASEE Computers in Education*, vol. 1, no. 2, pp. 6-14. April 2010.

Fields of Study

Major Field: Electrical and Computer Engineering

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1. Motivation

When designing a controller for a system, the control engineer may not be concerned with where the system output measurements are generated. This is especially true during the initial design phases when only a proof of concept controller is required before additional funding is provided. Through the use of virtual sensors, system outputs can be generated through simulation of a system model and provided to a control application alone or augmented with output from real sensors.

In an Intelligent Transportation application, a large vehicle is equipped with numerous sensors to detect various portions of the vehicles surroundings. Many of these sensors are redundant in order to increase the robustness of the system. This sensor suite can cost hundreds of thousands of dollars. Like any other control system, several simulation steps are performed before an algorithm is tested on the full-scale expensive vehicle. Simulations do allow for repeated testing under similar conditions as well as testing of scenarios impossible to plan in a real environment, but lack the sometimes necessary nondeterministic components required to test the robustness of the algorithms. In cases like this, a scale testbed is created where real robotic vehicles are used as an intermediate testing step before an algorithm moves on to the full-scale vehicle. In order to provide a sufficient test scenario, the small-scale robots must be equipped with similar

sensing capabilities as their larger counterparts. Sometimes this is still not cost effective as similar sensors may still cost several thousands of dollars. It may also not be possible or time effective to create realistic enough environments for the small robotic vehicles to navigate.

The main focus of this thesis is to describe the implementation of a Virtual Sensor System that combines the nondeterministic nature of a physical testbed with the simple environment creation and cost effectiveness of a simulation environment in order to expand the sensing capabilities of the robotic agents within the testbed. Within this system, robots can be equipped with virtual sensors that interact with the objects simulated in a virtual environment. These virtual objects can be linked to real objects so that they are positioned based on the state of the object in the physical testbed. Measurements for the virtual sensors are calculated and transmitted to the robots as if they were equipped with a physical version of the sensor. In this chapter, relevant literature describing uses and applications of virtual sensors and similar virtual environments is reviewed. Finally, the outline and organization of this thesis is given.

1.2. Literature Review

Virtual sensors, sometimes called soft sensors, are software based sensors that use an internal system model and other data including a systems control inputs and outputs from physical sensors to create a desired output signal. This provides sensor data to a system in place of a real sensor.

There are several reasons for choosing to use a virtual sensor over a real sensor [1], the most common being cases where a specific quantity cannot be measured directly, e.g.

the position of a robotic vehicle in Cartesian coordinates. In these cases an observer paradigm is used. Sometimes real sensors may not respond fast enough or there is significant lag due to communication to be ideal for control applications. Virtual sensors can be used to resolve these by predicting the sensor outputs using a Kalman Filter to provide continuous output data from the periodic, time-delayed input signals.

Virtual sensors can also significantly reduce the cost of a sensor suite in order to mean budget constraints as physical sensors can be either too expensive to install or maintain throughout the lifespan of the sensor. Some sensors have issues maintaining their calibration due to drift. In these cases, a virtual sensor model can be trained using data from a freshly calibrated sensor and used in the system in place of it. Virtual Sensors are also useful when the installation of the physical sensor is not possible due to either physical size constraints or environmental constraints (the environment either inside or outside the system is too harsh for the sensor to function appropriately).

A popular use for virtual sensor is in the area of fault tolerance. If a real sensor fails, a virtual sensor can be inserted into a system until the real sensor can be brought back online. [2] used a virtual sensor to provide data redundancy in a Helicopter Adaptive Aircraft which was able to take-off as a helicopter then unfold wings and transfer motion from the rotor to a propeller to proceed in forward flight as an airplane. The output of the virtual sensor was compared to the output of real sensors in order to detect a fault. A nonlinear virtual sensor used Nonlinear AutoRegressive with exogenous excitation (NARX) system identification to estimate the sweeping angle of the wings. With the use of the virtual sensor, the control system was able to detect a fault, localize

the fault within the system, and limit the propagation of the fault until the system was able to recover.

No matter the application, a virtual sensor requires some sort of system model. Depending on this model, the control inputs may be needed as well as data from a real system or sensor. This sensor data can be different than the actual information the model is estimating. If the system model is accurate enough, the state of the system can be predicted allowing the optimization of control inputs. A virtual sensor is only as good as its model.

In most cases, the system model may be unknown or rather complex therefore, most virtual sensors are based around system identification technologies. [1] described using a neural network to estimate the model of complex nonlinear systems. Real sensor data is used to generate these models so accurate sensors are required in order to ensure an accurate virtual sensor.

Virtual sensors were used to estimate the sideslip angle and lateral forces of a vehicle in [3]. Four different observers (a linear Luenberger observer, an extended Luenberger observer, an extended Kalman filter, and a sliding-mode observer) were tested using three different sensor configurations (yaw rate, vehicle speed, and yaw rate and vehicle speed together) in order to explore the stability of the observers and models as the car reaches linear dynamic limits. Using the Callas vehicle simulator and a nonlinear bicycle model, Stéphant, et al. were able to show that the nonlinear observers provided the best estimation of the sideslip angle and that the vehicle speed was not

needed for sideslip estimation and all observers were sufficient if the lateral acceleration was low (typical for normal driving situations).

The University of Washington in conjunction with the Washington State Department of Transportation [4] used virtual speed sensors to provide travel time and speed measurements of arterials and freeways based on probe sensor locations. Looking only at these locations may not provide an accurate description of current traffic conditions (e.g. high occupancy vehicle lanes tend to be faster and lanes containing buses are typically slower). The system that was implemented used Kalman filters to track vehicles to provide continuous estimations of the vehicle speed and location. This estimated data was used instead of the probe measurements by a Probe Estimator to provide reports about traffic conditions. This system was extended in [5] to interface with the existing traffic management system in Seattle, Washington. The information obtained from the existing infrastructure and the probe vehicles were fused together to provide the traffic condition estimations. This merging was initially problematic because the probe density was variable in both time and space.

Virtual sensors have been used in several instances for encapsulating sensor information to provide simpler interfaces in an effort to increase usability of the sensor data and decrease the overhead of learning the different interfaces. [6] designed a Mobile Virtual Sensors which was a processing middleware for tracking object through several cameras in a smart surveillance application. Each agent was configured to start and stop based on a series of events defined by the system programmer, handled simple tracking between a programmer-defined set of resources, swapped in and out other resources as

the object of interest moved between camera views. Previously, systems used a multi-threaded structure where each thread was responsible for gathering data from a small set of cameras. This structure required that each tracking agent be able to obtain information from each resource thread thus decreasing the overall speed of the system. Through the mobile virtual sensors, Kumar et al. were able to reduce CPU load by 60% because the sensors allowed for more selective processing of the image data.

This abstraction was not only useful for users but also for the other software systems. NASA's Jet Propulsion Laboratory discussed resource management in [7] where no distinction was made between real and virtual sensors when scheduling or monitoring the use of resources in a planetary rover.

In [8], Kabadayi et al. also utilize virtual sensors for purposes of abstraction. They describe a customizable interface to create a sensor fusion of heterogeneous data. In their system, sensors were abstracted in such a way that the programmer could obtain information from sensors that provide "location" or "temperature" instead of specifically requiring information from GPS or a thermocouple. The programmer would then implement an aggregator that would combine the different measurements into a custom output state. The update frequency of the aggregation was also customizable independent of the update frequencies of individual sensors.

Xiang and Özgüner describe a tree-shaped hierarchical virtual sensor structure in [9] to abstract sensor data from the low physical level to a high symbol level. A feature level of virtual sensors was used to extract features from real sensors. At this level, there was a single virtual sensor for each physical sensor. The next level used general virtual

sensors in a low level data fusion. Each general virtual sensor combined and integrated multiple, possibly heterogeneous virtual sensors from the feature level to create more complete and accurate representations of the features. A high level data fusion layer was added above this in order to perform decision making based on both the general virtual sensors and the feature level virtual sensors if necessary. A navigation example was presented describing a vehicle equipped with multiple laser range finders, radar, ultrasonic, and vision sensors. First, objects of interest were detected from each sensor individually. Then, the data from groups of sensors were fused together to find similar objects of interest. The decision making level was used to fill an occupancy grid used for planning the route the vehicle travelled through the world.

All of the examples described thus far have required that either the system in which the virtual sensor was used or inputs to the virtual sensor be in the real world. Several groups have done work with virtual sensors either completely in a virtual environment or augmenting the real world information with virtual information. Redmill, Martin, and Özgüner explain the implementation of a simulation environment designed to test the usability of sensor data and sensor fusion algorithms in intelligent transportation scenarios in [10]. The modular simulator created through this research, VESim, provided three dimensional environment simulations that could be used as input for vision algorithms, supported the simulation of vehicle models with six degrees of freedom and complex sensor output generation. This system allowed a user to create repeatable test scenarios for testing complex, dynamic control algorithms and multi-level sensor fusion algorithms.

The Player/Stage Project [11] has been widely used as a robotic development tool which encapsulates various hardware components of a robot so that control algorithms can be written independent of the robotic platform. A control application can be initially designed using the two dimensional, Stage, or three dimensional, Gazebo, simulators. This same application can then be used on a physical robot without modification. The Player/Stage Project also provides several standalone tools that can be used to visualize data obtained by a robot's sensors. Though these tools are very useful when debugging robotic systems, they can sometimes be difficult to interpret because they only display a small subset of the data utilized by the control systems. Collett and MacDonald [12] created an augmented reality system which overlaid the robot's world view on top of an image of the real world so that developer's can have a better understanding of the robot's view of the environment.

Dixon et al. created a system called RAVE [13] which allowed for the collaboration of both real and virtual robots which utilize both real and virtual sensor data. The primary goal of this project was to establish a common way for heterogeneous robots to interact with each other. This was similar to the Player/Stage project. Virtual sensors were used by this system to provide sensing systems for virtual robots as well as to augment existing onboard sensors on a real robot with virtual data. In the latter case, data from real sensors sensing only the real world were fused with that from virtual sensors sensing only the virtual environment. Many tools were also created to allow multiple users access to visualize robot information and control their systems.

1.3. Thesis Organization

This thesis presents a Virtual Sensor System implemented in the Control and Intelligent Transportations Research Lab at The Ohio State University. The thesis is organized as follows:

Chapter 1 begins by describing the motivation behind this work followed by a literature review in the area of virtual sensor networks.

Chapter 2 describes the initial multi-agent intelligent transportation testbed in use at the Control and Intelligent Transportation Research Lab. Each agent is described including software systems and the different robotic vehicles available for testing. Several unique testbeds are also described.

Chapter 3 presents the implementation of the Virtual Sensor System and how it interacts with the other agents in the testbeds. Details are given for both the various components implemented in the system and the three virtual sensors that were implemented.

Chapter 4 explains three test scenarios that were implemented in order to test the implementation and usability of the Virtual Sensor System. First, a general control setup is given that was used within the test scenarios. Then, two path following implementations are shown one using the lane edge sensor and the second using a magnetometer array. Lastly, an area mapping and obstacle avoidance scenario is described. This scenario utilized the laser range finder.

Chapter 5 summarizes the work, concludes this thesis, and points out areas where future work may be required.

Chapter 2: Simulation Environment

2.1. Introduction

In the Control and Intelligent Transportation Research Laboratory [14], numerous systems have been implemented both in hardware and in software in order to explore various intelligent vehicle application scenarios. All of the various components were designed so that each system could be used independently of the others in cases where a particular component was not needed or if it crashed due to an implementation flaw. This also facilitated the creation of additional components when that need arose. Systems were designed to use the Player interface whenever possible to achieve a smooth transition between the simulation and physical environments. Stage [11] and Gazebo have been used in various projects conducted at the CITR Lab to simulate various test scenarios before a full physical implementation was attempted.

The following sections describe the interconnectivity of the various systems in the lab including the Virtual Positioning System and the three different robotic vehicles utilized. This architecture allows for the rapid implementation of different testbed environments customized to the specific scenarios to be tested. The details of two such environments are outlined, an urban area environment called SimVille [15] and an open area environment called MiniMAGIC. Finally, the Stage simulator is described in detail.

2.2. System Architecture

A diagram of the various component interconnections is shown in Figure 1. The primary components of the system were the Virtual Positioning System and the mobile robots. The positioning system received images of the testbed over Firewire from two cameras and detected unique glyphs attached to objects in the environment. The dotted lines in the figure represent the cameras viewing the tags mounted to the mobile robots. The state of the tags was transmitted to each of the mobile robots in the testbed. This component is described in more detail in the following section.

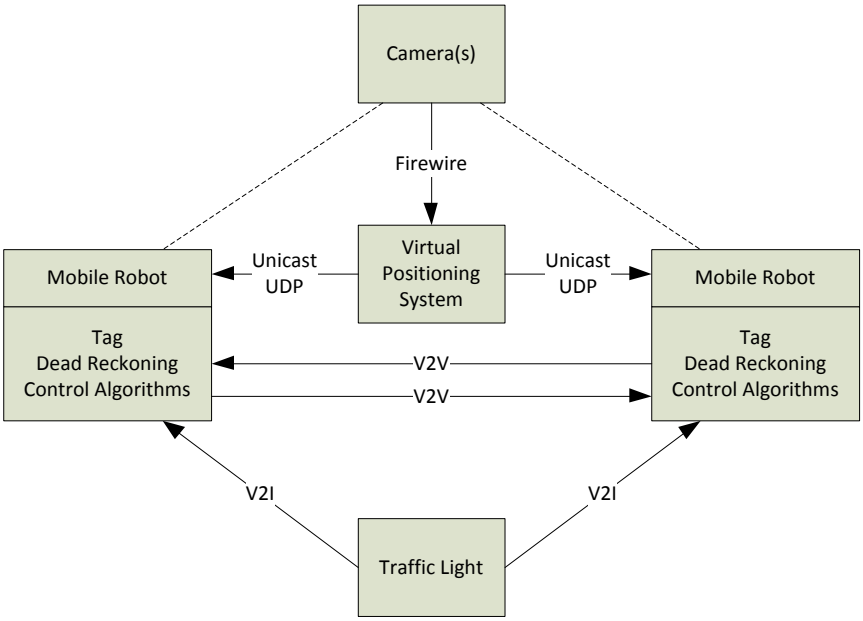

Figure 1: System Architecture

Each of the mobile robots had a robotic vehicle as its basic component and was equipped with several hardware and software components independent of which vehicle was utilized. In addition to its situation specific control logic, the robot included a sensor

fusion module which estimated position and orientation states from the measurements received from the Virtual Positioning System and the robot's onboard wheel encoders and inertial sensors.

In order to mimic the modularity of the whole system, the software systems used on the robots were created as separate applications that could be run when needed. Communication between these various software components was done through the use of shared memory structures.

Player was used on all of the mobile robots in the implementation of the logic controllers in order to create a smooth transition between the simulation environment and the physical robot as well as the transition between different robotic platforms. Player provided a way of encapsulating all of the platform specific information of each robot through a standard software interface. A configuration file was used to setup this specific information. This file was created once for each robot and could be utilized by any new control algorithm without modification.

The mobile robots communicated between each other through a vehicle to vehicle interface. The V2V packets contained information about the robot's position, orientation, and linear and angular velocities as well as other important information including whether the current robot was following another robot or if it was navigating toward a specific checkpoint. Through this interface, the robot indicated whether it was in the process of driving into a docking station for charging.

For urban test environments, a traffic light was implemented and transmitted its position and light status to other agents in the system through a vehicle to infrastructure

interface. The V2I packet also contained information about the position of the stop lines for the individual lanes of the intersection and timer information to determine the length of time until the next light transition. This packet was based off of the Signal Phase and Timing (SPaT) message defined in [16]. Just like the mobile robots, any number of traffic lights could be added into the environment without interrupting the other functionality of the system. The physical traffic light used an ATmega128 based RoboStix from Gumstix [17] to control several LEDs. The V2I information was transmitted over 802.11B using a serial to wireless module provided by Qualcomm.

Typically, all of the information between subsystems was transmitted over the 2.4 GHz wireless band using 802.11B wireless cards. A 900 MHz wireless network was also implemented to support data transmissions to one of the older robotic platforms used.

2.2.1. Virtual Positioning System

A Virtual Positioning System was implemented in order to simulate the functionality of the Global Positioning System that would be used by intelligent vehicles when in outdoor environments. Two Scorpion cameras from Point Grey Research [18] were used to acquire top down images of the lab area. The intrinsic parameters of each camera were calculated before they were mounted to the ceiling of the lab at a height of 6.5m. These cameras transmit grayscale images to a processing computer over Firewire at a rate of 15 Hz.

For detection by the Virtual Positioning System, a unique two dimensional barcode was mounted to the top of each mobile robot and static object of interest such as buildings. The Augmented Reality Toolkit Plus (ARTKPlus) [19] was used to detect

these barcodes in the grayscale images from the two Scorpion cameras. Each image was processed independently to give the position and orientation of each tag found in the image relative to the center of the camera's field of view. This information was then translated into a global coordinate system and was tracked in order to estimate linear and angular velocities of the tags. This data was collected for each tag and transmitted over Unicast UDP packets to the IP address associated with each tag ID. Figure 2 shows a screenshot of the processing application displaying the images received from each camera and an overlay of the tag IDs and positions.

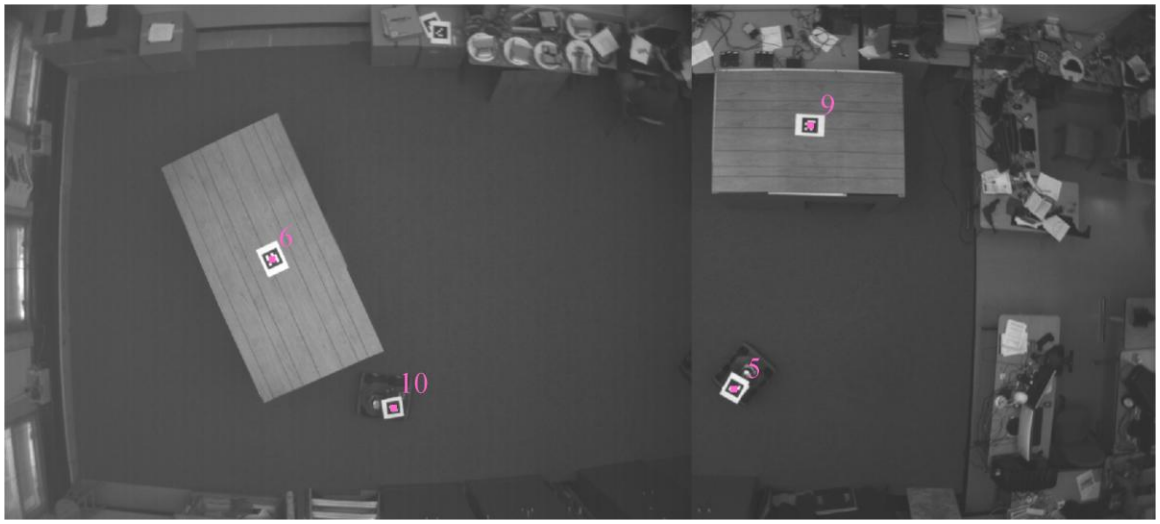

Figure 2: Screenshot of the Virtual Position System Processing Application

2.3. Robotic Vehicles

Several robotic vehicles have been interfaced with the testbeds created in the Control and Intelligent Transportations Research Lab. All of the systems created in the lab have been designed to facilitate the addition of different agents, robotic or otherwise,

without the rebuilding of the previously added systems. This section describes three of the primary robotic platforms that have been used for various intelligent vehicle designs.

2.3.1. iRobot Create

One of the primary robotic vehicles used in the CITR Lab was the Create Programmable Robot from iRobot [20]. The Create, shown in Figure 3, was designed as an affordable robot platform tailored toward academic and hobby applications. The small 33 centimeter diameter disk shaped robot looked similar to iRobot's Roomba vacuum cleaning robot but provided a small cargo bay and greater interface capabilities in place of the vacuuming components.

The iRobot Roomba Open Interface protocol [21] was used by an external microcontroller connected to the Create's 25 pin cargo bay connector to collect information from the Create's onboard sensors and transmit motor commands to the robot. This connector also included several digital inputs, one analog input, and a few digital outputs in case the attached controller does not have these capabilities or if addition ones are needed. The Create was equipped with individual encoders for each of the two wheels with millimeter accuracy. The attached motors would allow the Create to travel up to 0.5 m/s. Four cliff sensors were provided in order to detect if the Create had reached the edge of the platform on which it was driving. To detect whether the robot had encountered any physical obstacles, two micro-switches were supplied on the front of the Create and were connected by a large bumper. iRobot's virtual walls and docking stations were detected by utilizing the omnidirectional infrared receiver that was attached to the top of the bumper.

Figure 3: iRobot Create

A Gumstix Connex [17] microcontroller board was used as the controller interface to the Create equipped with a PXA255 Xscale processor from Marvel. A CFStix was connected to the Connex in order to provide a CompactFlash socket for an 802.11B CompactFlash wireless card. An Element Direct Sticky Interface board was used to connect the Connex to the Create through its 25-pin cargo bay connector. This interface board provides a switching power supply to maximize the Create's battery life, a USB host connector, general purpose IO breakouts, and I^2C expansion ports.

2.3.2. MobileRobots Pioneer

MobileRobots [22], formerly ActivMedia Robotics, manufactures the Pioneer 3-AT four wheel drive, all-terrain robotic platform primarily for research and prototyping applications. Figure 4 shows the basic Pioneer 3-AT platform. The Pioneer's drivetrain consisted of two high torque, high speed motors each to drive two of the nine inch

diameter pneumatic tires. Three lead acid batteries were used to provide ample power for all of the Pioneer's subsystems. An external battery charger was provided by MobileRobots to charge these batteries.

The Pioneer was equipped with two eight sensor sonar arrays, one in the front of the robot and one in the rear, with each sensor positioned in twenty degree intervals to provide a full 360° of coverage. Each sensor measurement was acquired at 25 Hz and provided a resolution between 10 cm and 4 m. Ten bumpers were used for obstacle detection when other sensing methods were inconclusive.

Figure 4: MobileRobots Pioneer 3-AT Robotic Platform
[http://www.mobilerobots.com/]

The basic Pioneer platform allowed numerous other sensors to be attached to the robot in order to expand its capabilities for both indoor and outdoor environments. For outdoor test scenarios, a NovAtel GPS module was attached to the Pioneer, but for the indoor environment described here, this sensor was not utilized. In addition to the GPS

module, a Sick LMS-200 Laser Range Finder was mounted to the robotic platform providing a 180° field of view at up to 0.25° angular resolution with distance measurements up to 80 m. The Pioneer was also equipped with a Canon VC-C4 camera with motorized pan, tilt, and zoom mechanisms which were utilized for vision based applications. Figure 5 shows the Pioneer 3-AT robot with all of the optional accessories described.

A PC/104 form factor computer was equipped with an Intel Pentium 3 processor and was used to implement the hybrid control logic necessary for vehicle autonomy. This computer had access to all of the onboard sensors and was programmed to utilize all of these sensors in order to navigate its environment.

Figure 5: Pioneer 3-AT Robot with Optional Sensors

2.3.3. K-Team Khepera II

The Khepera II robot was designed and distributed by the K-Team Corporation [23] based in Switzerland. These 70 mm diameter hockey puck shaped robots were equipped with a 25 MHz Motorola 68331 processor programmable through a serial connection. Eight infrared proximity and ambient light sensors surrounded the outside of the robot giving it the ability to sense obstacles around it. Three external analog inputs were provided for use with other sensors, but were not used. An AeroComm [24] serial to 900 MHz wireless adapter was attached to the Khepera to provide wireless communication to the other agents in the environment. Figure 6 shows a picture one of the Khepera II robots equipped with the AeroComm wireless adapter.

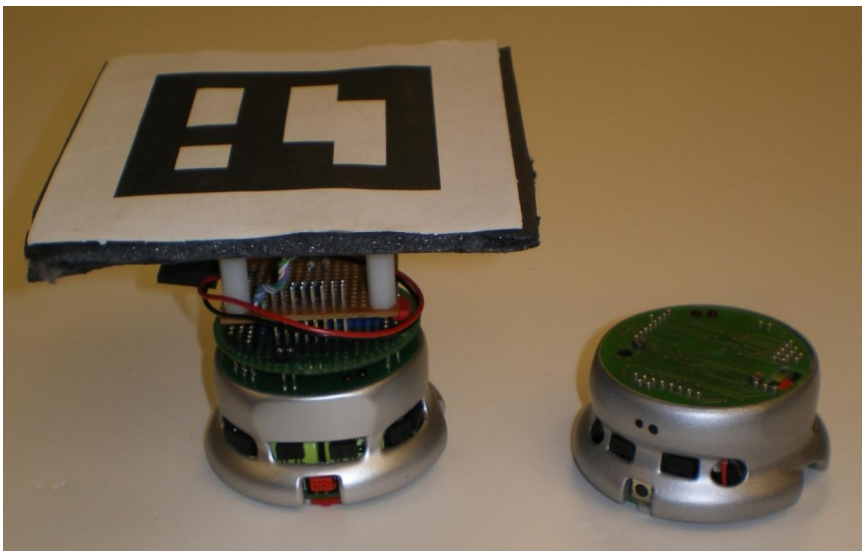

Figure 6: Khepera II Robots

The Khepera II was equipped with high accuracy wheel encoders providing submillimeter distance resolution. The servo motors powering the wheels were able to

drive the robot between 0.02 m/s and 0.5 m/s. Continuous driving would allow for approximately one hour of battery life.

2.4. Environments

2.4.1. SimVille: An Urban Area Testbed

SimVille, shown in Figure 7, was created after the 2007 DARPA Urban Challenge [25] in order to continue research efforts in urban environment scenarios. A road network was created to be 1/7 scale with 0.5 m wide lanes in order to use the Create robots as urban vehicles. This scale also allowed for the Khepera II robots to be used to mimic pedestrian behavior if the test scenario required. The road network was designed to provide areas for testing many diverse scenarios without the need to change the test environment. Several intersections were included both with and without traffic lights. An overpass and several simulated buildings were added in order to test GPS dropout situations. A zone area was used to test robotic maneuvers in a more loosely constrained area. Parking spaces were also added into the zone area. Many of the road lanes were laid out in a fashion such that the direction of travel for the lane could be changed or a specific purpose be given to it (e.g. a turn only lane).

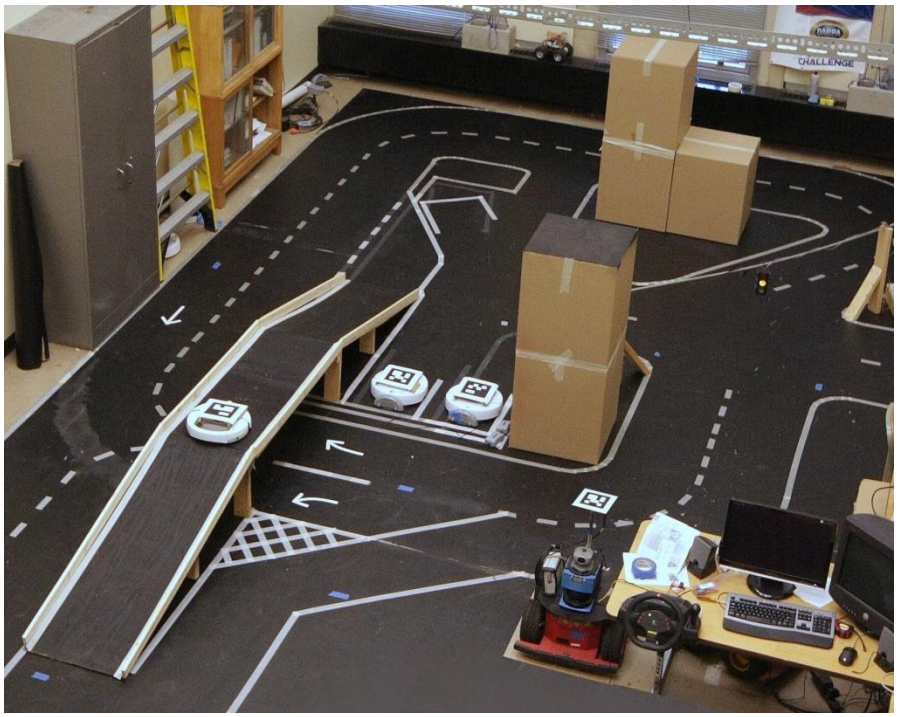

Figure 7: Simville Urban Area Environment

A large straight portion of road was created to simulate a highway environment. This two lane section was used to test both high and low speed passing and merging maneuvers into and out of this section. Convoy maneuvers were also explored in this section, shown in Figure 8. The exit of this area was used to test methods of having one vehicle merge into the middle of an already formed convoy.

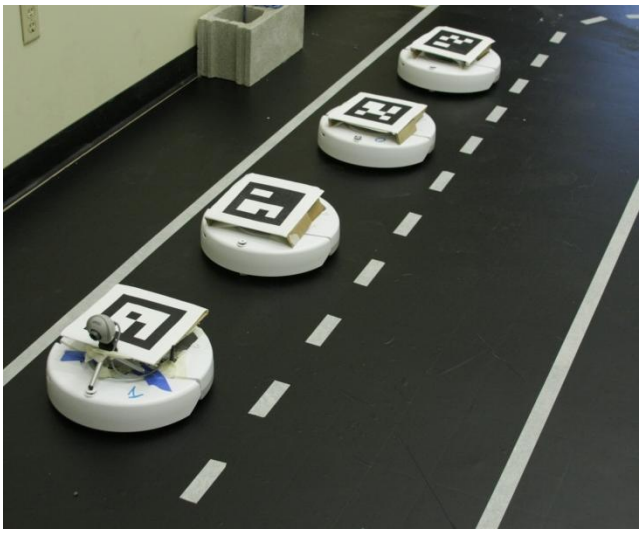

Figure 8: Convoy Maneuvers

A Road Network Definition File (RNDF) [26] was a road layout specification created by DARPA for use in both Grand Challenges and the Urban Challenge. This file format contained information about the start and end locations of a lane in the network as well as waypoints on the lane in between the two. Lane width and the type of lane boundary, such as solid yellow or broken white lines, were also included. Special points of interest on the lane could be designated as checkpoints. A Mission Definition File (MDF) was used to list the checkpoints that were required for the vehicle to navigate to. Both files were used by the vehicles' control logic to plan routes through the network.

2.4.2. MiniMAGIC: An Open Area / Building Testbed

MiniMAGIC was created as a testbed environment for developing control logic and testing various scenarios related to the Multi Autonomous Ground-robotic International Challenge (MAGIC) 2010 [27]. MAGIC was sponsored jointly by the Australian and United States' Departments of Defense in an effort to start creating the

next generation of robotic vehicles to be effective in both civilian rescue scenarios and military operations. Multi-robot teams navigated through and mapped an open area environment trying to locate simulated threats. Robots also had to be able to navigate into and through buildings in order to find some of the targets.

MiniMAGIC, shown in Figure 9, was designed to maximize open area in order for dynamic obstacles to be placed. Building structures were created to simulate the buildings that would be seen at the actual competition field. Carpeting was used on the testbed surface to create a much more rugged terrain similar to the outdoor environment. The Pioneer 3-AT robots were used as they were designed as an all-terrain vehicle and were more similar in size and sensing capabilities as the robots actually used for the competition.

Figure 9: MiniMAGIC Environment

2.5. Stage Simulator

The Stage simulator was originally designed as a two dimensional simulation for populations of mobile robots. Robots and sensors used computationally inexpensive models rather than high accuracy, computationally expensive models so that real-time tests could be performed using entire fleets of robots. Recent updates expanded the two dimensional simulator into two and a half dimensions by utilizing a stack of two dimensional planes to simulate the third dimension. Figure 10 shows a screenshot of Stage using the SimVille environment, and Figure 11 shows the MiniMAGIC environment implemented in Stage.

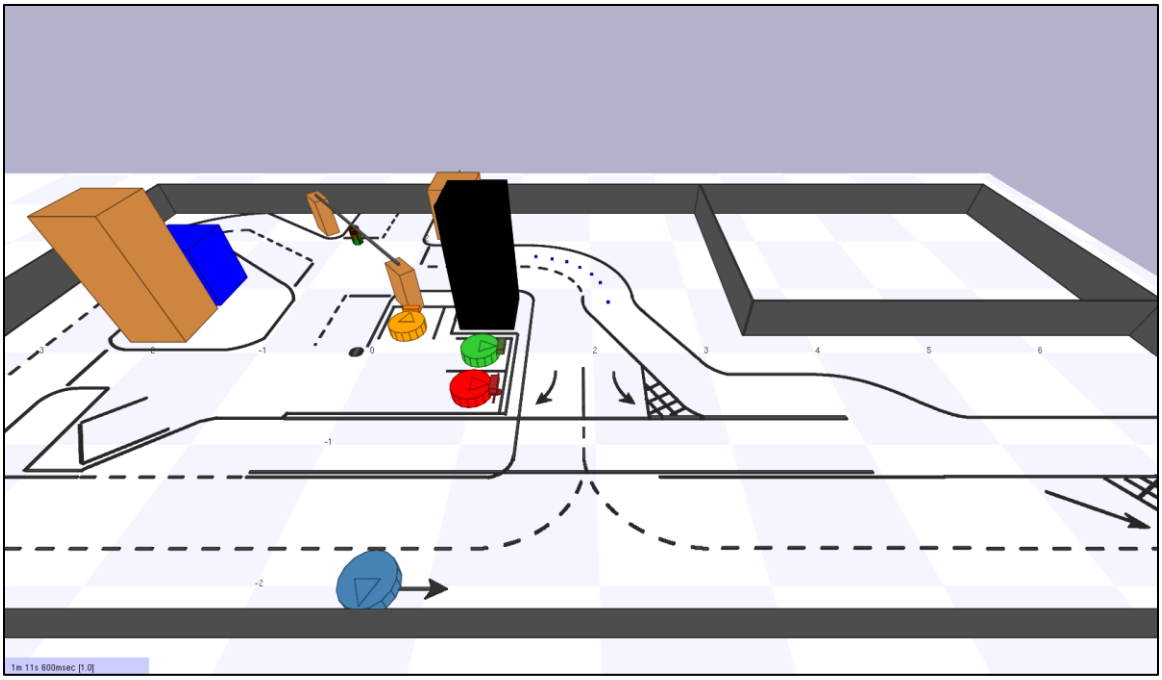

Figure 10: Stage Simulator with SimVille Environment

Simulated controllers were compiled ahead of time and dynamically loaded through the use of a world file. This world file allowed the setup of the ground layout and the various static and dynamic objects required in the simulation scenario. Different sensor and robot models could be created in separate files and used in the world file. Objects were defined using either a set of points to create a polygon, a simple block structure, or a separate image file. This basic outline was then extruded to a specified height to give the illusion of three dimensions.

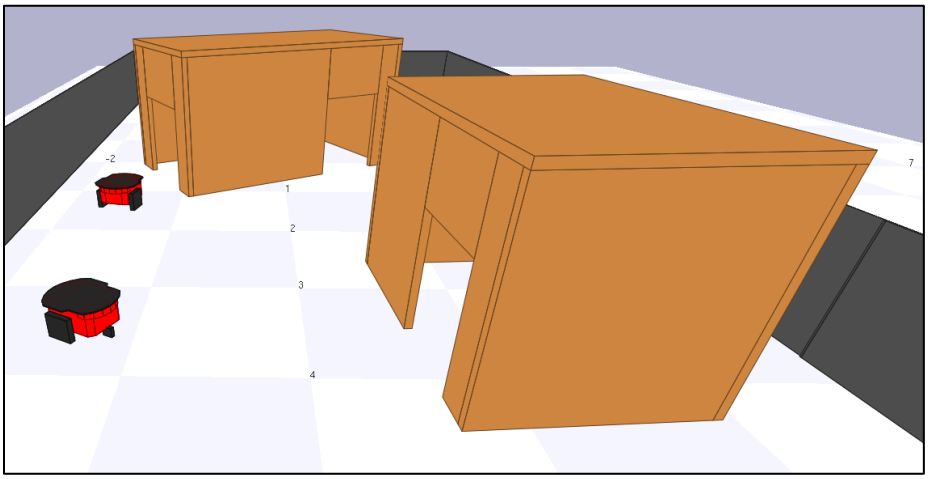

Figure 11: Stage Simulator with MiniMAGIC Environment

Player was used in conjunction with Stage to create logic to control robots as they navigated through the simulation environment. Stage also allowed more flexibility for different simulation needs through the libstage modules. Using this setup, the simulator setup and execution could be more specifically customized. This was the method used to implement the Virtual Sensor Framework described in the following chapter.

Chapter 3: Virtual Sensors

3.1. Introduction

In the Control and Intelligent Transportation Research Lab, many autonomous vehicle simulations have been conducted using its existing infrastructure. In order further extend the capabilities of the lab, a Virtual Sensor System was created and interfaced with the other agents. In this system, sensors were modeled in the Stage simulation environment and were simulated in order to provide data to actual robotic vehicles in the testbed as if a real sensor was physically connected to the robot. This allowed for additional sensors to be used by the robot's control algorithms without the budget overhead of purchasing the sensors or the time overhead of creating the interface between a new sensor and the robot itself. Through the implementation of the Virtual Sensor System, the randomness of the real world was integrated with the deterministic structure and infinite possibilities of a simulation environment.

The following sections describe how the Virtual Sensor System interacted with other agents in the CITR lab and how real world objects were linked to their virtual representations. The implementation of a traffic light is detailed to show how the visualization capabilities of the system could be utilized. Three sensors were implemented in the system: a laser range finder, a magnetometer, and a fictitious lane edge sensor. Details on their creation and use are given below.

3.2. System Extensions

3.2.1. System Architecture

The Virtual Sensor System runs as an extension to the Stage simulation environment. Through the *libstage* interface provided by Stage, the system was able to control the various aspects of the simulation rather than relying on the default Stage executable. This allowed for more flexibility and control over the execution of the system. The Virtual Sensor System was able to process the Stage world definition file, make any necessary modifications to the objects in the world, and execute callback functions when individual models were updated. The world definition file contained information about the virtual representation of each robot and its virtual sensor setup as well as models for other static objects that were required for a testbed scenario (e.g. building, trees, and traffic lights in an urban environment).

Figure 12 shows the communication interconnects between the agents in the CITR lab with the addition of the Virtual Sensor System. This system was added to the existing lab setup so that it could be started and stopped independently of any other running system. This ensured that the Virtual Sensor System did not have to be running if it was not needed for a specific autonomous vehicle scenario run in the testbed.

The Virtual Sensor System was able to receive V2V data packets transmitted from each of the robotic vehicles in the testbed as well as the V2I data packets sent by the traffic light. Information on the position and orientation of static objects was obtained from the Unicast UDP packets generated by the Virtual Positioning System. Once the data was calculated for each sensor in the simulation, Unicast UDP packets containing

this information were transmitted to the corresponding robot for use in its unique control software.

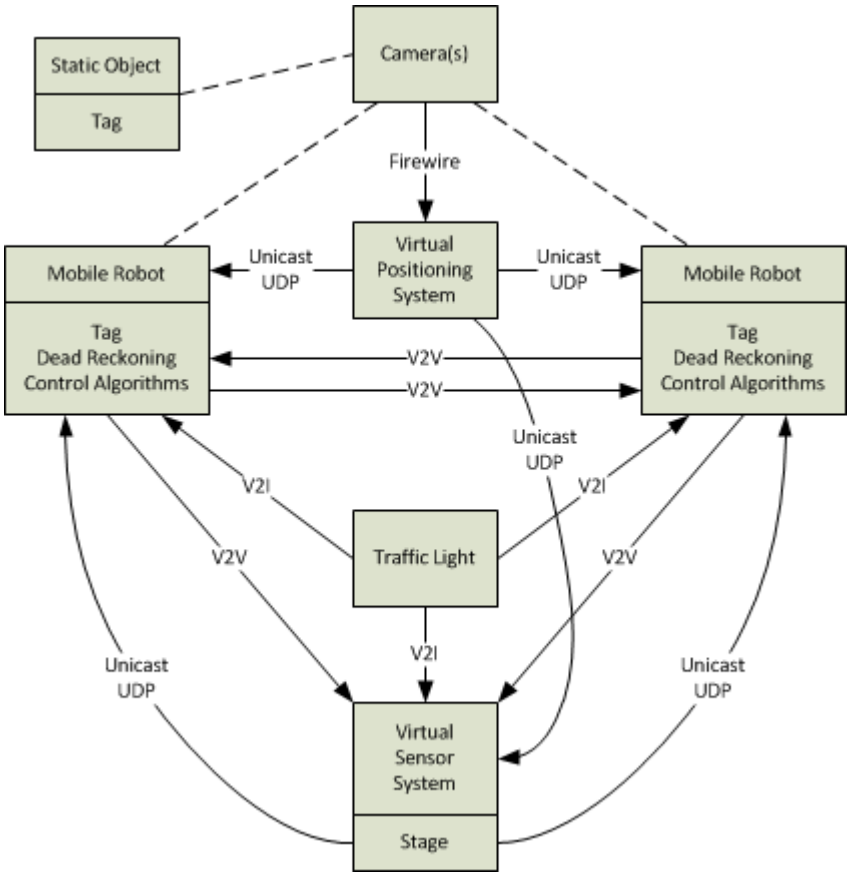

Figure 12: System Architecture

3.2.2. Linking Real and Virtual Objects

In the Virtual Sensor System, a link was created between the real objects in the testbed and the virtual objects placed in the Stage simulation environment so that the virtual objects were placed in the same location as their physical counterparts. To accomplish this goal, tags recognized by the Virtual Positioning System were affixed to the top of the physical objects in the testbed. As they system detected the tags, it would

transmit Unicast UDP packets to the computer running the Virtual Sensor System which contained the tag's identification number as well as its position, orientation, and both rotational and linear velocities. A separate application running on this machine received these data packets and updated a shared memory structure containing state information for all thirty-two tags supported by the Virtual Position System.

For the virtual object to appear in the simulation environment, a model was added to the testbed world file read by Stage. To create the actual link from the physical object, the Virtual Sensor System would look for objects with the name "tag" followed by a number between one and thirty-two. This number corresponds to the tag identification number contained in the data packet received from the Virtual Positioning System. During each update of the system, the state of the tag of each linked object was read from a shared memory structure and was used to update the virtual position of that object. Figure 13(a) shows a picture of a box seen in the physical testbed with the tag placed on its top surface. The corresponding virtual object is shown in Figure 13(b).

One of the benefits of this object linking paradigm was that the virtual object did not have to look exactly the same as the physical object. For instance, a simple box could be used in the real testbed to represent a much more complex building in the virtual environment. The tag could also be simply placed on the ground and used to represent a tree standing alongside one of the roads in SimVille. This latter example is shown in Figure 14.

Figure 13: (a) Box Used in Testbed to Represent a Building. (b) Box Model used in Virtual Sensor System.

Robot models were handled slightly differently than other objects in the Virtual Sensor System. Rather than receiving the tag information directly from the Virtual Positioning System, the Virtual Sensor System received state information directly from each robot. This allowed the robots to use onboard sensors including wheel encoders and inertial sensors to augment the state information obtained from the Virtual Positioning System. After calculating its new state, the individual robots transmitted this information as a V2V packet which was read by an external program that would update a shared memory structure with the new state information. This structure was read by the Virtual Sensor System to update the state of the robot models.

Models were linked to the corresponding physical iRobot Create robot by being named "roomba" followed by the robot identification number. Support for the Pioneer and Khepera robots was not implemented. During each update of the system, each robot's state was updated according to the information obtained from the V2V messages through the shared memory structure. Figure 15 shows a Create robot both in the real

testbed and in the virtual environment. In this image, the virtual Create robot looks very similar to the physical robot. This does not have to be the case because just as a tree model could be linked to a physical tag, the robot's state could be linked to other shapes like a model of a car.

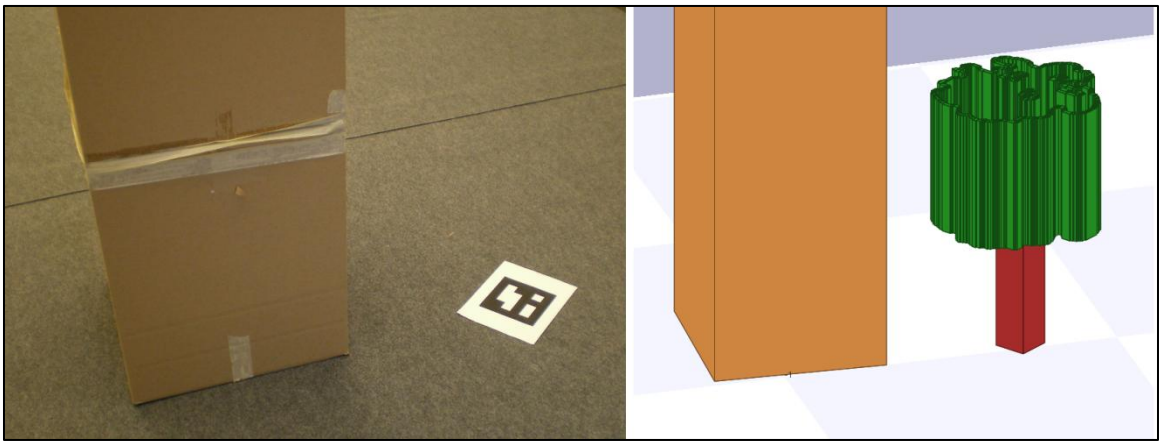

Figure 14: Tag Used to Represent Virtual Trees

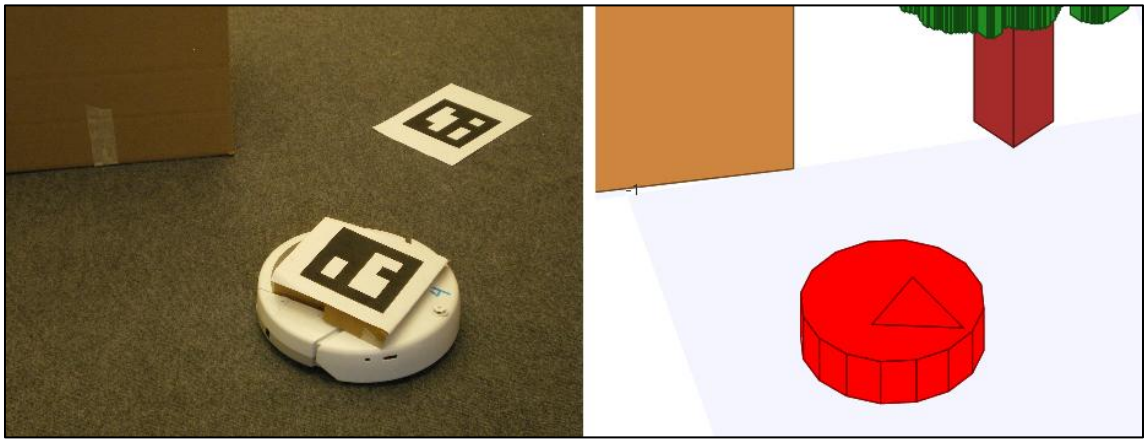

Figure 15: iRobot Creates in Physical Testbed and in the Virtual Sensor System

3.2.3. Traffic Light

To add to the visual aspect of the Virtual Sensor System, a traffic light was implemented for use with SimVille. The physical traffic light transmitted V2I information to all other agents in the testbed. A simulator was also created to generate these messages for situations in which the physical traffic light could not be used. The same program used by the robots to receive this data was used by the Virtual Sensor System to determine the state of the light for updating the representation of the light in Stage. Figure 16 shows the physical traffic light in SimVille as well as its representation in the Virtual Sensor System.

Two different traffic light styles were created for use in the Virtual Sensor System. A simplified traffic light provided a single light for each direction of the intersection rather than the typical three. This allowed the proper color of light to be seen in both the default two dimensional view as well as the three dimensional view. For more realistic visualizations, a standard traffic light model was created with three lights on each of the four sides. Since the red light would occlude the status of the other two lights in the two dimensional view, a small indicator was placed on the top of this traffic light model and was updated to show the current light state in any given direction. Figure 17 shows these two traffic light configurations.

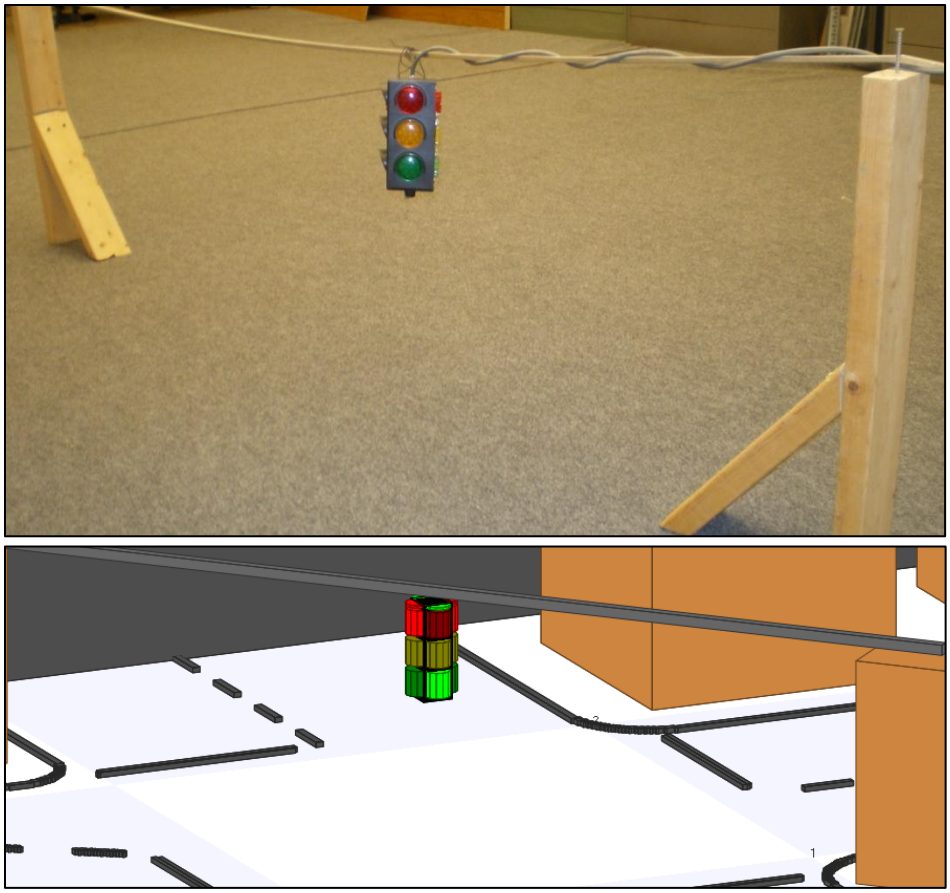

Figure 16: Real and Virtual Traffic Light

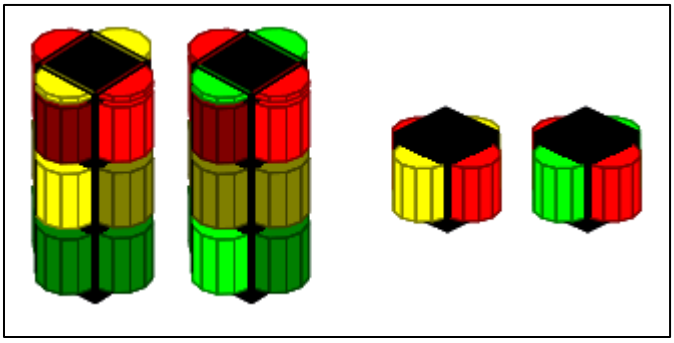

Figure 17: Two Different Traffic Light Representations

In order to decrease the rendering time of a simulation step, Stage utilized the display list paradigm [28] of OpenGL. During initialization of the models, Stage stored

all of the drawing calls so that they could be reused in each subsequent frame. The problem with this implementation was that once a model was initialized, no part of its model could be changed. To circumvent this issue, two traffic light models were used instead of a single model, one model configured with a red light on two opposite sides and yellow on the other two and one model configured with red and green. One of the models was chosen during the rendering of each frame and was rotated 90° if necessary in order to obtain the appropriate traffic light representation. The other model was placed underneath the ground plane of the virtual environment (typically at the z = 0 plane) so that it would not be seen during the simulation as seen in Figure 18.

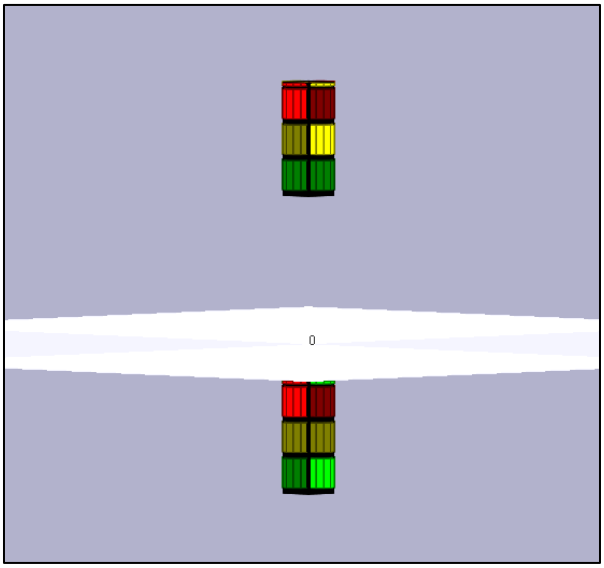

Figure 18: Secondary Traffic Light Model Hidden From View

One major drawback with this implementation was that each of the two models was required to be defined in Stage's world definition file. Also, since the models could not be dynamically updated, the virtual traffic light did not support the flashing red and

flashing yellow states. In this implementation, only one traffic light was supported. Any other traffic light models would not be initialized.

3.3. Sensors

3.3.1. General Sensor Structure

The creation of each new sensor required the implementation of three separate parts. First, a simulation model needed to be already available or created within the Stage environment. Then, two interfaces needed to be built, one to add the sensor into the Virtual Sensor System and another to allow sensor data to be transmitted to individual robots.

The interface to the Virtual Sensor System required an initialization phase where the sensor manager would search for and initialize all instances of that sensor attached to each robot in the simulation. During this phase, the Unicast UDP communication sockets were created.

This interface was also required to implement a callback function that would be executed during each time step of the simulation as Stage updated the state of each specific model. The callback function would obtain any data from the sensor, translate that data into the specified Unicast UDP packet structure, and transmit the packet to the robotic vehicle.

So that each robotic vehicle was able to be equipped with several separate sensors of the same type, the data for each sensor was transmitted to a unique port number. The global interface provided the declaration of the base port numbers for all of the sensors of that type and a function that was used to calculated the transmission port number given

the base port number, the robot identification number, and the sensor identification number. A structure was also provided to define the data representation of a sensor reading.

3.3.2. Laser Range Finder

A laser range finder utilizes Light Detection and Ranging (LIDAR) technology in order to determine the locations of objects within its field of view. These devices typically consist of a single laser beam that is reflected off of a spinning mirror in order to generate a known number of rays in a plane. These rays reflect off of nearby objects and are detected by photo sensors internal to the device. The distance to an object along a ray is calculated from these measurements. A commercial laser range finder is shown in Figure 19. The device on the left was the same model that was mounted to the Pioneer robot (Sick LMS-200 [29]) and updates at ~75 Hz, has a maximum range of 80 m, and costs over $6000. The device on the right is a much smaller model from Hokuyo [30] which could be mounted to a Create robot. This sensor only updates at a rate of 10 Hz with a maximum range of 5.6 m and still costs over $1500. The virtual laser range finder was created because this price point was not practical for full deployment to all of the robots.

Figure 19: Two Commercial Laser Range Finders
[http://www.sick.com/, http://www.hokuyo-aut.jp/]

The implementation of the laser range finder used Stage's *ModelLaser* class for the low level laser simulation. This class provided several configuration parameters that could be modified within the Stage world definition file. The field of view was modified using *fov*. The variables *range_min and range_max* were used to change the minimum and maximum range for each ray of the laser. *Samples* defined how many individual ray trajectories were calculated within the laser's field of view. The *resolution* variable was used in order to decrease the number of calculations performed for each ray. If *resolution* was set to be greater that one, only every nth ray intersection would be calculated. The remaining range values were then calculated using linear interpolation. The ray intersection calculations were already rather quick since Stage traverses its internal occupancy grid to find intersections between the rays and objects in the world as opposed to doing complete three dimensional ray-polygon intersections.

Since several robotic vehicles would appear in the simulation, modifications were made to how Stage displayed the laser data since it draws every laser in the same blue color. The parameter *laser_color* was added within *ModelLaser* so that a unique laser color could be set within the world definition file for runtime changes. A sample laser definition is shown in Figure 20.

```
define roombalaser laser
(
      range_max 0.75
      fov 180.0
      samples 361
      size [ 0.156 0.155 0.19 ]
)
roomba
(
      name "roomba0"
      color "steel blue"
      roombalaser
      (
            pose [ 0 -0.19 -0.025 -90 ]
            color "blue"
            laser_color "blue"
      )
)
```

Figure 20: Example Laser Range Finder Configuration

The Unicast UDP structure provided the number of rays that were cast for the specific sensor as well as the distance value measured along that ray. This structure supports a maximum of 361 so that it can be transmitted using only a single UDP packet. This number allows the user to have 0.5° resolution over a 180° field of view. Each of the range values was represented in millimeters by a sixteen bit unsigned integer to

provide a maximum range of 65.535 meters. Figure 21 is a screen capture of the Virtual Sensor System showing a Create robot equipped with a single laser range finder configured as shown above.

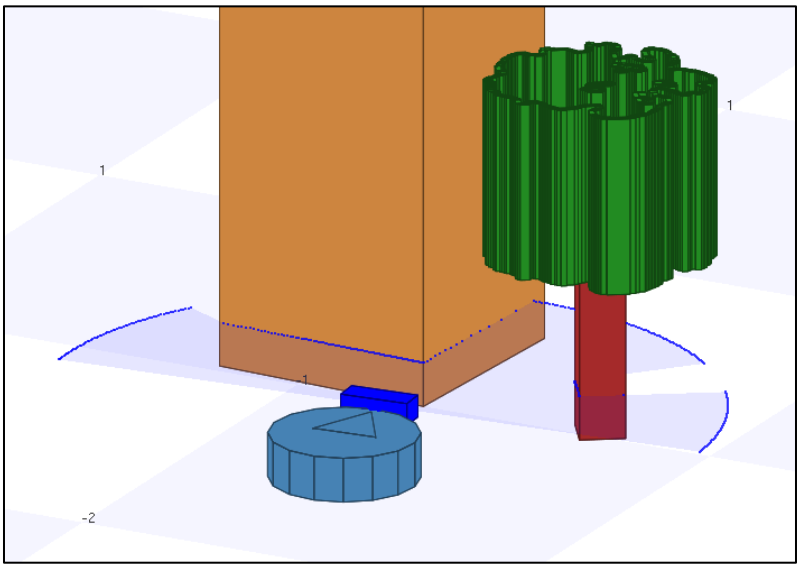

Figure 21: Create Robot with Laser Range Finder

3.3.3. Magnetometer

A magnetometer is used to measure the strength and direction of a magnetic field. The Earth has its own magnetic field which can be measured in order to roughly estimate the orientation of an object. Early research conducted in the field of automated highway systems used several magnetometers mounted to a car in order to measure the magnetic field of small magnets embedded in the roadway. These magnets were typically placed in the center of the roadway. The magnetometers were mounted in such a way that the

car's offset with the center of the lane was measured and used as input into its autonomous driving system.

Since Stage did not contain a magnetometer model, one was created for it. For simplicity, the magnetic field of the Earth was ignored in the model because this sensor was not intended to be used as a compass. A *magnetic_moment* property was added to each model so that any object could be used as a magnet. A *magnetometer_return* property was also added in order for the sensor model to determine if a specific object generated a magnetic field. This was modeled after the approach used for the laser and other models available in Stage. Figure 22 shows an example configuration of a magnet.

```
define magnet model
(
      magneticmoment 100000
      magnetometer_return 1

      color "blue"
      size [ 0.05 0.05 0.10 ]
)
magnet
(
      pose [ 0 0 -0.09 0 ]
)
```

Figure 22: Example Magnet Configuration

The magnetic field of each object was modeled by the magnetic field generated by a cylindrical magnet placed at the center of that object with a given magnetic moment. Equation $|B| = \frac{\mu_0 M}{4\pi r^5}\sqrt{(3xz)^2 + (3yz)^2 + (2z^2 - x^2 - y^2)^2}$ (1 [31] was used to

determine the magnitude of the magnitude of magnetic field at a point $P = (x, y, z)$ referenced to the center of the magnet.

$$|B| = \frac{\mu_0 M}{4\pi r^5}\sqrt{(3xz)^2 + (3yz)^2 + (2z^2 - x^2 - y^2)^2} \qquad (1)$$

where μ_0 is permeability and M is the magnetic moment.

The world file definition was based off of the ranger model provided by Stage. In this description, a single device was configured to have one or more individual sensors. The properties *scount* and *spose* determined the number of individual sensors and their locations. The sensitivity of each sensor was initialized though the *ssensitivity* property. The size of the box drawn for each sensor was determined by the *ssize* property. Figure 23 shows an example configuration of a magnetometer with five individual sensors.

```
define roombamagnetometer magnetometer
(
      scount 5
      spose[0] [ -0.15  0 0 ]
      spose[0] [ -0.075 0 0 ]
      spose[0] [  0.00  0 0 ]
      spose[0] [  0.075 0 0 ]
      spose[0] [  0.15  0 0 ]
      ssensitivity [ -10000 10000 ]
      ssize [ 0.01 0.05 ]
      # shape definition removed for simplicity.
)

roomba
(
      name "roomba0"
      color "red"
      roombamagnetometer
      (
            pose [ 0 -0.2 -0.04 0 ]
      )
)
```

Figure 23: Example Magnetometer Configuration

The UDP packet structure allowed for sixteen sensors to be configured for each magnetometer. The magnitude of the magnetic field measured by each sensor was represented by a signed thirty-two bit integer in units of 10^{-4} Gauss. This allowed for magnetic field measurements between -214748.3648 Gauss and 214748.3647 Gauss. Figure 24 shows a five sensor magnetometer mounted to the front of a Create robot. The small squares in front of the robot are magnets.

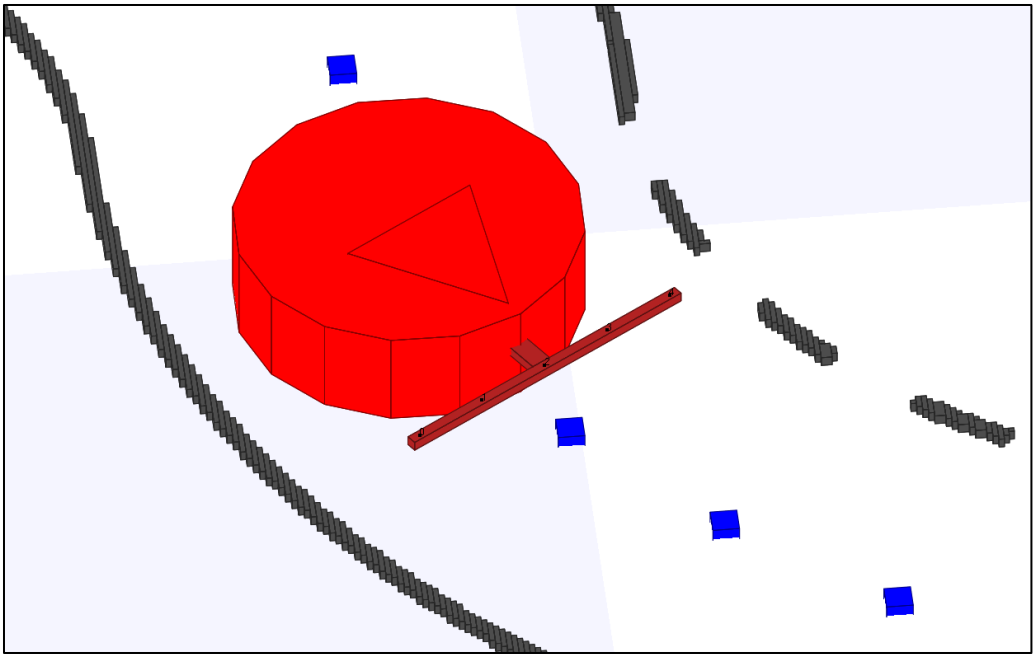

Figure 24: Create Robot with Five Sensor Magnetometer

3.3.4. Lane Edge Sensor

A lane edge sensor is a fictitious sensor that has been used for teaching basic autonomous vehicle control concepts [32]. This sensor provides its user with the distance to the edge of the lane. It is possible to generate this measurement using a fusion of different sensor data commonly consisting of a camera or through an observer. For this sensor, the user is more interested in the actual measurement output rather than how the data is actually obtained.

The lane edge sensor was used as an example to show how the flexibility of the Virtual Sensor System allowed for the production of a sensor measurement in a way different from how the measurement would be collected in the real world. This implementation was based around the laser model provided by Stage. When processing the world definition file, Stage turns any visible object into a volume. Any road layout in

the testbed was loaded into Stage as an image which was processed into a set of rectangular prisms whose geometry was drawn rather than a single textured quadrilateral. Since the lane edges had a thickness and were added into Stage's occupancy grid, a laser was used to collide with the objects.

The laser model was configured to cast out several rays within a narrow field of view. The model was placed very close to the ground so that the rays would collide with the rather thin road layout geometry. During the initialization of the Virtual Sensor System, only lasers named "laneedge" were initialized as lane edge sensors. All other laser models were assumed to be laser range finders. After each update of the simulation, the range values were obtained for each of these rays. Based on the location of the sensor on the robot and the state of the robot itself, the ranges returned by the laser model were transformed into the three dimensional intersection points. All range values close to the maximum range were discarded.

A line was fit to these intersection points. The slope and the y-intercept were calculated using linear regression. The line was then extrapolated in order to find the distance to the lane edge perpendicular to the robot. This method was used in order to account for cases where the lane edge was dashed or when the sensor was not placed perpendicular to the forward direction of the robot. Figure 25 shows graphically how the lane edge distance was calculated. If the application of the sensor did not require this robust of a calculation, the sensor was configured using only a single ray who's range was used as the lane edge measurement.

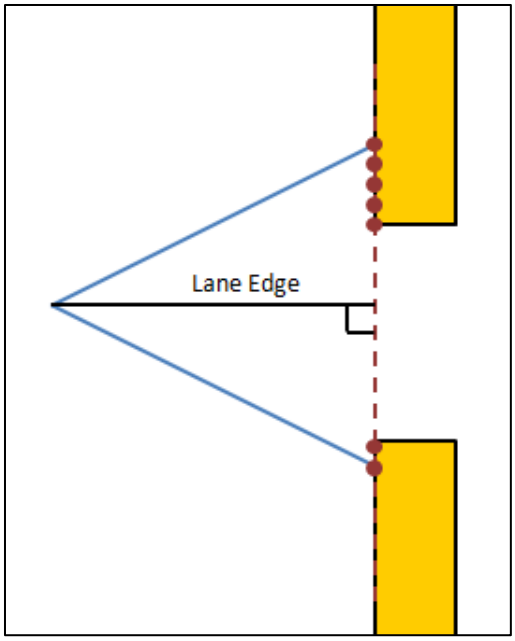

Figure 25: Graphical Representation of Lane Edge Calculation

The UDP packet structure contained only a single sixteen bit unsigned integer representing the distance from the sensor to the lane edge. The measurement was transmitted in millimeters allowing for a maximum lane edge distance of 65.535 meters. An example configuration for both a single ray and a multiple ray lane edge sensor is shown in Figure 26. For the single ray, Stage requires that at least two samples are defined. If this was detected, the Virtual Sensor System only interpreted the range from the first of the two. Figure 27 shows two configurations of the lane edge sensor on a Create robot, one with a single laser measurement and one with multiple measurements. Figure 28 shows the multiple ray configuration of the lane edge sensor on a curved road with a dashed lane edge.

```
define roombalaneedge laser
(
	range_max 0.5
	fov 10.0
	samples 10

	# for single ray sensor use:
	fov 1.0
	samples 2

	size [ 0.06 0.03 0.01 ]
	name "laneedge"
	# shape definition removed for simplicity.
)

roomba
(
	name "roomba0"
	color "steel blue"
	roombalaneedge
	(
		pose [ 0 -0.225 -0.1 0 ]
		color "blue"
		laser_color "blue"
	)
)
```

Figure 26: Example Lane Edge Sensor Configurations

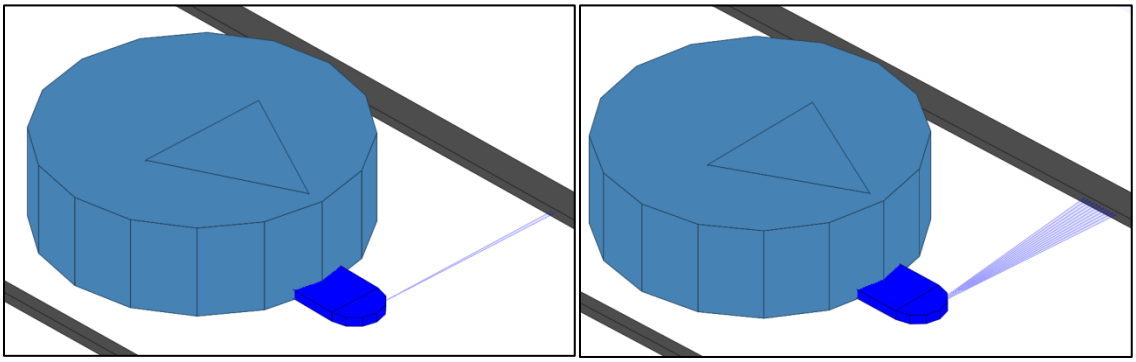

Figure 27: Create Robot with Two Different Lane Edge Sensor Configurations

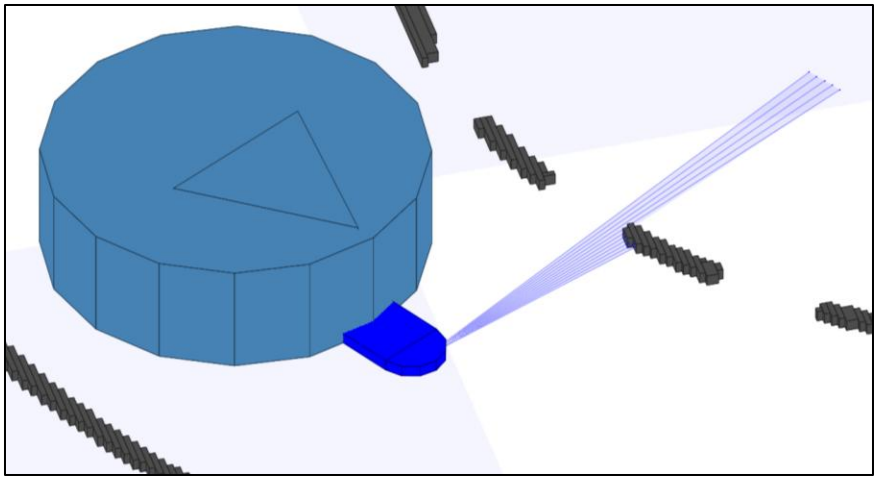

Figure 28: Lane Edge Sensor on Dashed Lane Edge

Chapter 4: Sensor Test Scenarios and Results

4.1. Introduction

After the Virtual Sensor System was fully implemented, it was necessary to test the functionality and practicality of the system. In order to do this, simple test scenarios were completed using the iRobot Create robots to test the three sensors implemented in the system. A path following scenario was designed with two perpendicular straight paths connected by a smooth 90° turn. Both the lane edge sensor and magnetometer were used individually to follow this path. The laser range finder was tested separately in an area mapping and obstacle avoidance scenario. For this test, a three meter by three meter area was sectioned off for the robot to navigate through. Several real and virtual obstacles were placed within this area for the robot to avoid.

The following sections describe how the Virtual Sensor System was used by the Create robot in the CITR lab. The common control logic used in all three sensor use cases is first described because the same basic structure was utilized in all three control applications. Details about the path following scenario and the two sensor suites used to navigate the test path are given. The results for each of the two tests are provided. The implementation of the area mapping and obstacle avoidance scenario is described as well.

4.2. General Control Logic Setup

Though the scenarios for which they were created differed, the three control programs followed the same basic structure. Figure 29 shows the simple finite state machine that was used to control the flow of the control application. Through the use of command line parameters, the experimenter was given the ability to decide whether the robot would immediately begin executing its control logic or wait for a start signal. Although the use of this start signal may be more beneficial when used in multi-robot scenarios, it can also be used to start logging applications or to give the user enough time to start a video camera. The Control Logic state was a meta-state [33] meaning that a secondary finite state machine was embedded within this state.

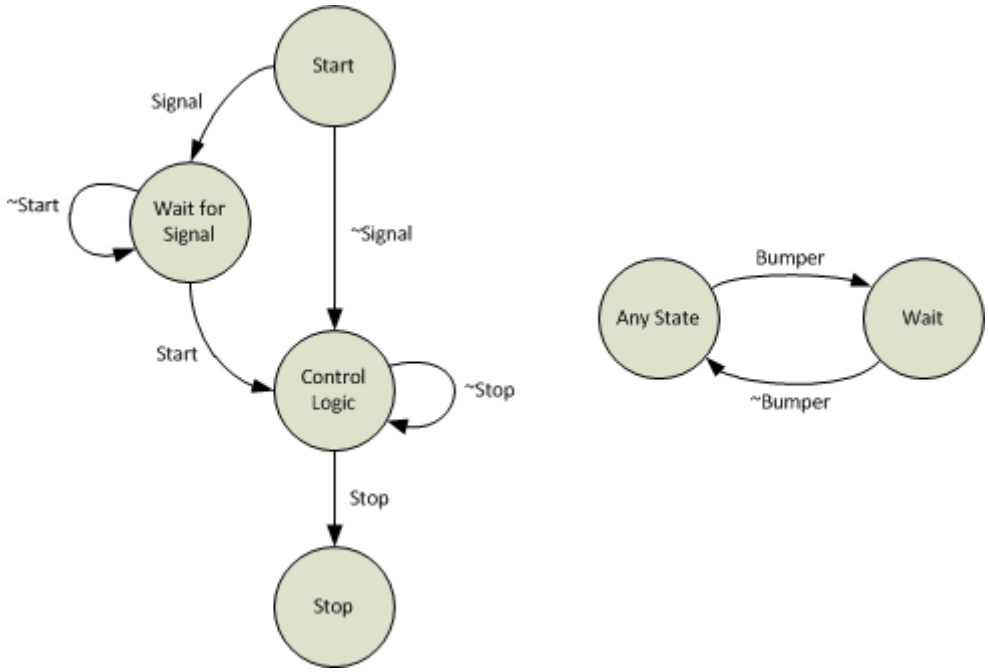

Figure 29: Common Control Logic State Machine

A component called *BumperProxy* was provided by the Player interface to allow the control application to access the bumper mechanism of the Create. This interface was used so that if the robot collided with any object during testing, the robot would move into a Wait state where it stopped driving until the bumper was released at which time the application would return to the previous state. This provided enough time for a supervisor to either remove the obstacle or stop the control code if it was the root of the error. The Player interface also provided a *Position2dProxy* which encapsulated the specific robots driving commands by allowing the user to change both the linear and rotation speeds of the robot.

4.3. Path Following

4.3.1. Path Description

Path following is a fundamental component of intelligent transportation systems thus making it an ideal scenario to test the functionality of the lane edge sensor and magnetometer. Figure 30 shows the simple path used for testing. This path was originally a segment from the SimVille environment that was isolated and rotated in order to achieve the orientation shown. Before each test, the Create was placed at the left most portion of segment one. This straight path was 2.72 meters in length and led into a 90° left turn with an approximate radius of 0.93 meters to the center of the lane. The final section was 1.349 meters in length was added to see how the controllers would recover from navigating through the curve.

This path was created entirely in the simulation environment of the Virtual Sensor System. Tags for the Virtual Positioning System were placed in the testbed and linked to small blocks in the simulation world in order to locate the Create in the proper place along the path. Figure 31 shows these tags in the testbed.

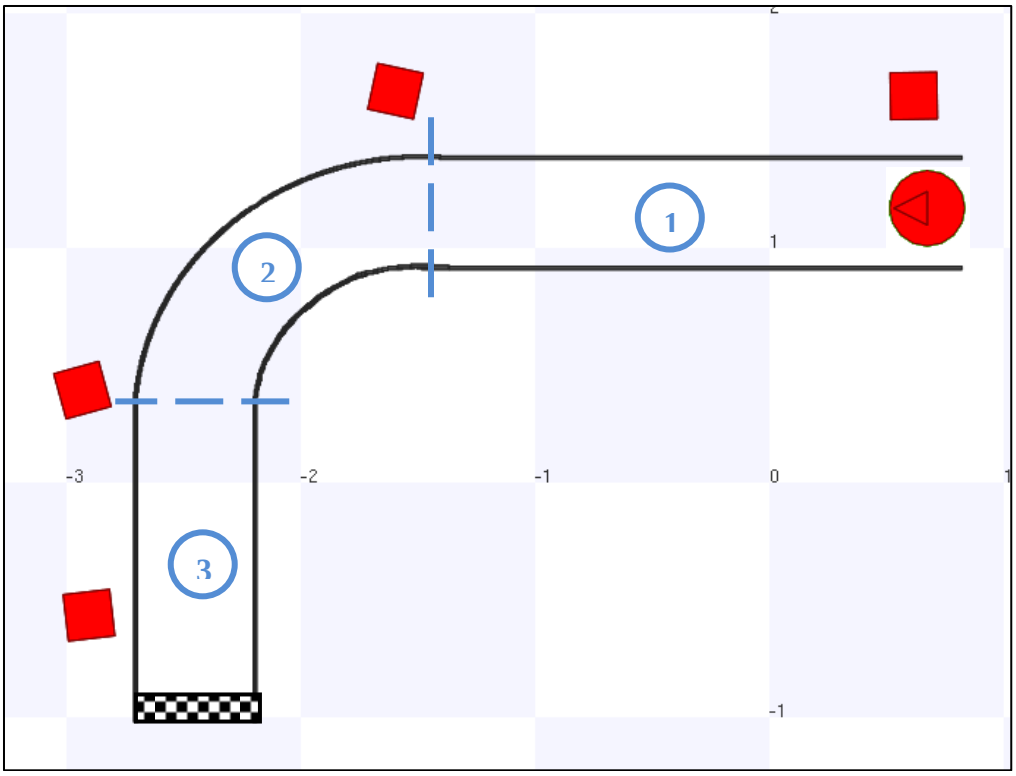

Figure 30: Test Path Layout

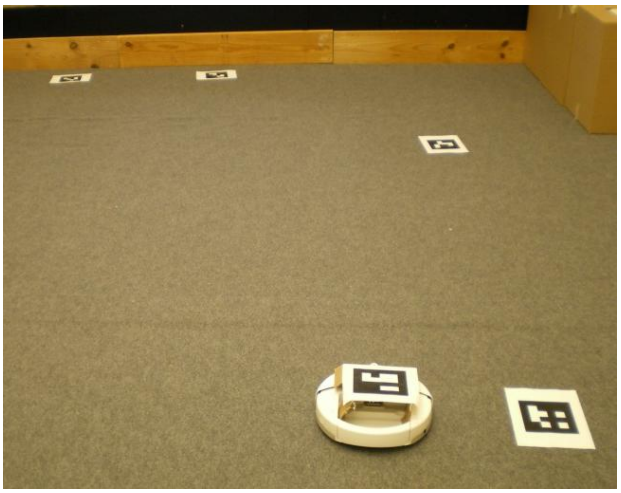

Figure 31: Physical Testbed

4.3.2. Lane Edge Sensor Tests

A lane edge sensor was placed 22.5 centimeters from the center of the Create. This sensor was composed of sixty one rays in a thirty degree, left pointing field of view. Figure 32 shows an overhead view of this setup.

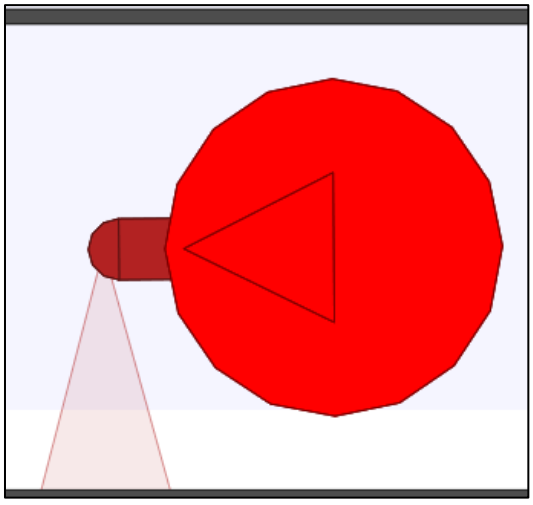

Figure 32: Lane Edge Sensor Setup

Since each of the three segments of the path had different lane widths, the control algorithm was required to have knowledge about the path in order to attempt to travel in the center of the lane during the entire test. If this was not intended, a mean lane width could be selected such that the robot would be within the lane throughout the entire path.

A proportional and a proportional-integral controller were used to control the steering rate of the robot. The linear velocity of the robot was set at a constant 0.1 m/s. The error, Equations $z(t) = distance\ to\ lane\ edge$ (2 and $e(t) = z(t) - l_{ed}$ (3, at each time step was calculated from the difference between the current lane edge measurement and half of the lane width. As described before, the lane width changed according to the section in which the robot was maneuvering. The steering input for the proportional controller is shown in Equation $u(t) = K_P e(t)$ (4, and the input for the proportional-integral controller is shown in Equation $u(t) = K_p e(t) + K_I \int e(t)dt$ (5. The integral of the error was calculated using the trapezoidal rule, Equation $\int_{t-\Delta t}^{t} e(t)dt \approx \frac{\Delta t}{2}(e(t-\Delta t) + e(t))$ (6, using the elapsed time between iterations of the controller—about 0.1 seconds—and the error calculated in the previous iteration. In some cases, the integral term can grow significantly causing the computer representation of the number to overflow. A common fix for this is to periodically clear the integral term. This was not actually needed during testing of these controllers due to the oscillation apparent in the path causing the integral value to be close to zero. The relatively short elapsed time of a test run also made this unnecessary.

$$z(t) = distance\ to\ lane\ edge \qquad (2)$$

$$e(t) = z(t) - l_{ed} \qquad (3)$$

where l_{ed} is half of the lane width for the current section of the path.

$$u(t) = K_P e(t) \qquad (4)$$

where K_P is the proportional gain.

$$u(t) = K_p e(t) + K_I \int e(t)dt \qquad (5)$$

where K_P is the proportional gain, and K_I is the integral gain.

$$\int_{t-\Delta t}^{t} e(t)dt \approx \frac{\Delta t}{2}(e(t-\Delta t) + e(t)) \qquad (6)$$

Because of several nondeterministic factors in each run, three separate runs were performed for each gain value while tuning the controllers. Each set was analyzed in order to determine the next gain value to be tested. Through this method, a proportional gain, K_P, of 0.5 was used in both controllers and an integral gain, K_I, of 0.015 was used. The addition of derivative control did not significantly affect the robots trajectory along the path and was thus not included in the final controllers. Figure 33 shows three trajectories of the Create using only the proportional controller. The trajectories resulting from the use of the proportional-integral controller can be seen in Figure 34.

Several statistics were calculated to compare the results of the individual controllers. The mean, standard deviation, minimum, and maximum were determined for both the initial error vector and the absolute value of the error. A mean close to zero was

desired for the signed error while the mean of the absolute value of the error showed the average deviation of the robot's trajectory to either side of the path.

Table 1 shows these values for the proportional controller. The results of the proportional-integral controller are shown in Table 2. All values in both tables are given in meters.

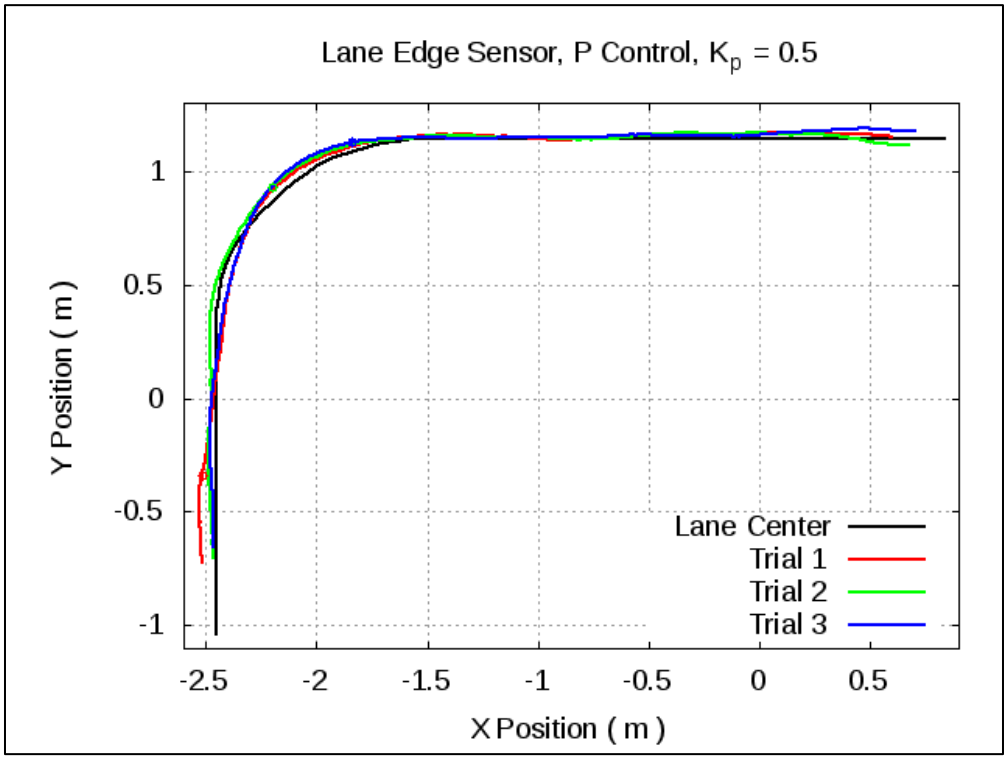

Figure 33: Proportional Controller Trajectories

Table 1: Proportional Controller Statistics

Trial	Mean	Std. Dev.	Minimum	Maximum	Abs. Mean	Abs. Std. Dev.	Abs. Minimum	Abs. Maximum
1	0.027	0.074	-0.194	0.871	0.048	0.063	0.000	0.871
2	0.030	0.061	-0.161	0.184	0.048	0.048	0.000	0.184

3	0.022	0.045	-0.133	0.169	0.035	0.036	0.000	0.169

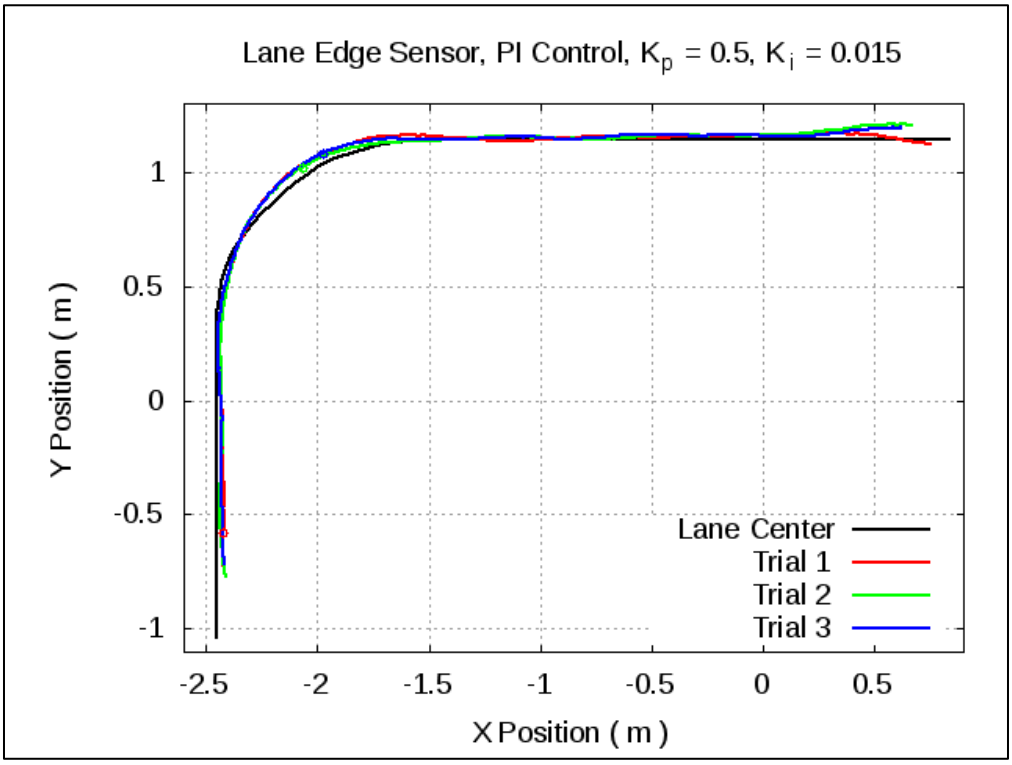

Figure 34: Proportional-Integral Controller Trajectories

Table 2: Proportional-Integral Controller Statistics

Trial	Mean	Std. Dev.	Minimum	Maximum	Abs. Mean	Abs. Std. Dev.	Abs. Minimum	Abs. Maximum
1	0.017	0.052	-0.190	0.169	0.041	0.036	0.001	0.190
2	0.015	0.050	-0.165	0.230	0.038	0.035	0.000	0.230
3	0.017	0.053	-0.145	0.184	0.041	0.037	0.000	0.184

4.3.3. Magnetometer Tests

A magnetometer sensor array composed of five individual magnetometers was placed in front of the Create such that the center sensor was twenty centimeters from the

center of the robot. Two sensors were placed 7.5 cm on either side of the center sensor. The remaining two sensors were placed 15 cm from the center, one on either side. Figure 35 shows an overhead view of this configuration.

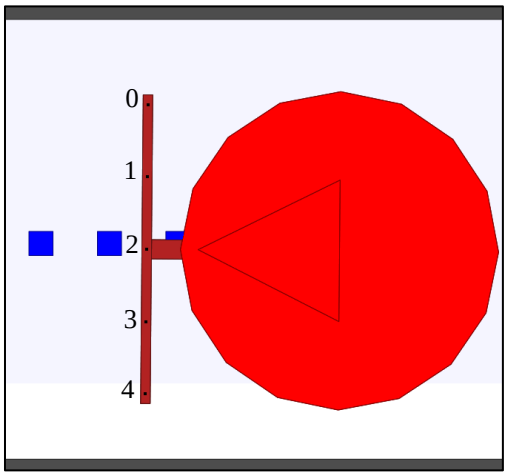

Figure 35: Magnetometer Setup

Magnets were placed in the center of the lane along the entire path approximately 7.1 centimeters apart. With the 1/7th scale of the Create robots, this would equate to 0.5 meters between magnets [34]. Each magnet was 2.5 cm wide and 10 cm long with a magnetic moment of 10000 m^2A. Figure 36 shows how the magnets were placed along the path.

Since the string of magnets marks out the center of the lane, a priori information about the lane width was not necessary like it was in the lane edge sensor test scenario. This knowledge would be necessary in situations where the required path was more complex and a route planner component was included in the control algorithm.

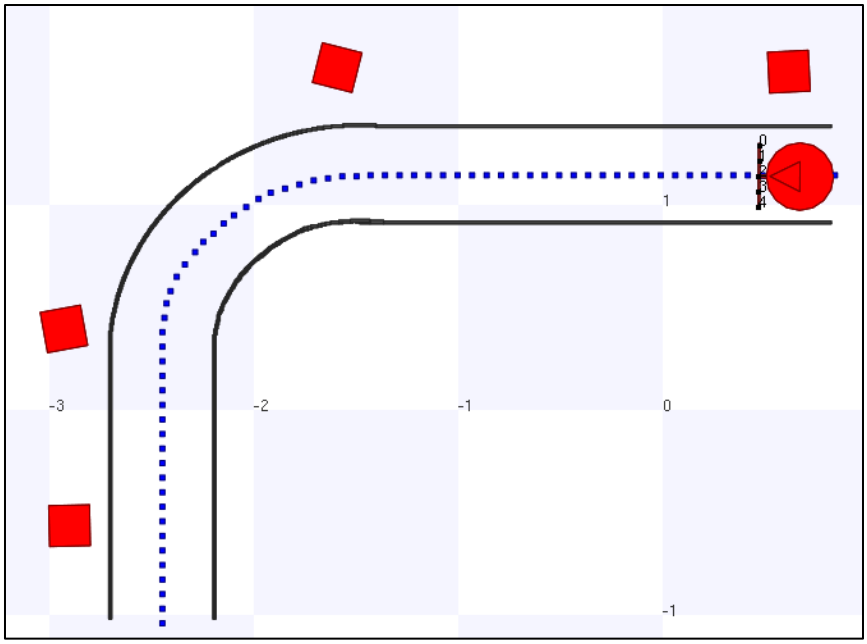

Figure 36: Test Path with Magnets

Similar to the lane edge sensor test scenario, a proportional controller and a proportional-integral controller were used to control the steering rate of the Create. The Create's velocity was again set to a constant 0.1 m/s. Because the magnetic field of the magnetic field was calculated using an ideal model, the error signal, Equations $z(t) = [\, z_0(t),\ z_1(t), z_2(t), z_3(t), z_4(t)\,]^T$ (7 and $e(t) = z_4(t) - z_0(t)$ (8, for the controllers was chosen as the difference between the measurements of the two outermost magnetometers in the array. This was not too unreasonable because the major missing component of the measurement was the magnetic field of the Earth. The two sensors were placed close enough together that the magnetic field measured of the Earth at the two points would be roughly the same. Figure 37 shows the magnetic field of six magnets in a line measured at a distance of 7.5 cm above the top surface of the magnet.

This distance was the same as the distance between the magnetometers on the Create and the magnets embedded in the lane. This separation distance provided an almost continuous magnetic field to be measured by the magnetometer. Since the robot was always moving forward and the center of rotation of the robot was not close to the center magnetometer sensor, a zero error would result only in the case where the center sensor was at the maximum point of the field and the centerline of all five sensors was perpendicular to the centerline of the magnets. The integral of the error was calculated using the trapezoidal rule.

$$z(t) = [\, z_0(t),\ z_1(t), z_2(t), z_3(t), z_4(t)\,]^T \qquad (7)$$

where z_i is the magnetic field measurement from the *i*th sensor.

$$e(t) = z_4(t) - z_0(t) \qquad (8)$$

The same method was used for tuning the two controllers as was used for the lane edge sensor controllers. The proportional gain, K_P, of both controllers was 0.01 and the integral gain, K_I, for the proportional-integral controller was 0.006. Figure 38 shows the trajectories of three test runs using only proportional control. The results for the proportional-integral controller are shown in Figure 39.

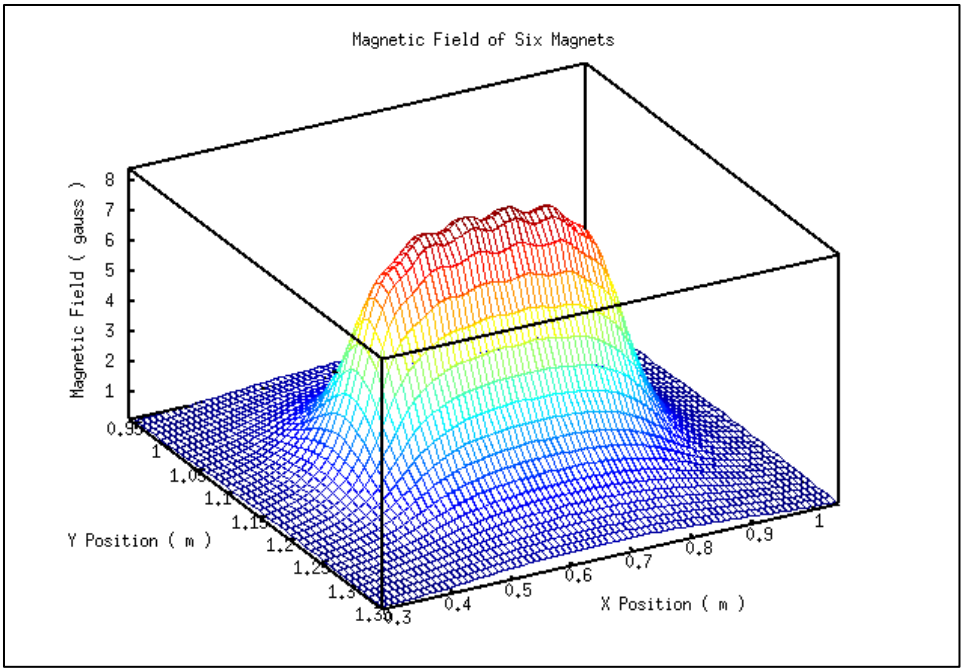

Figure 37: Magnetic Field of a Line of Discrete Magnets

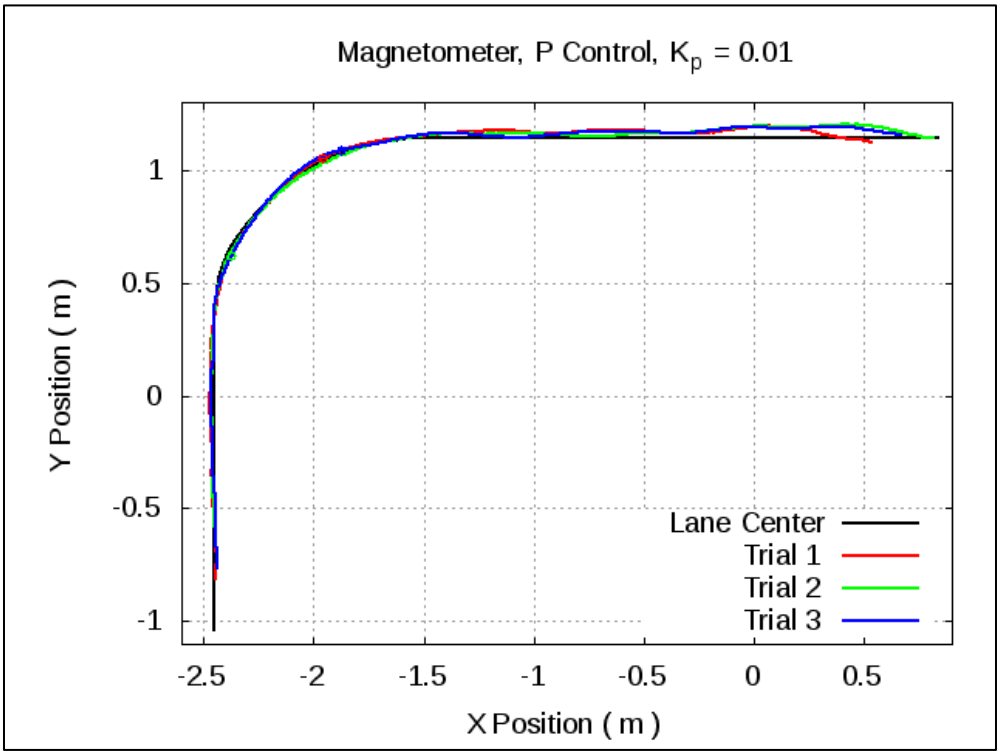

Figure 38: Proportional Controller Trajectories

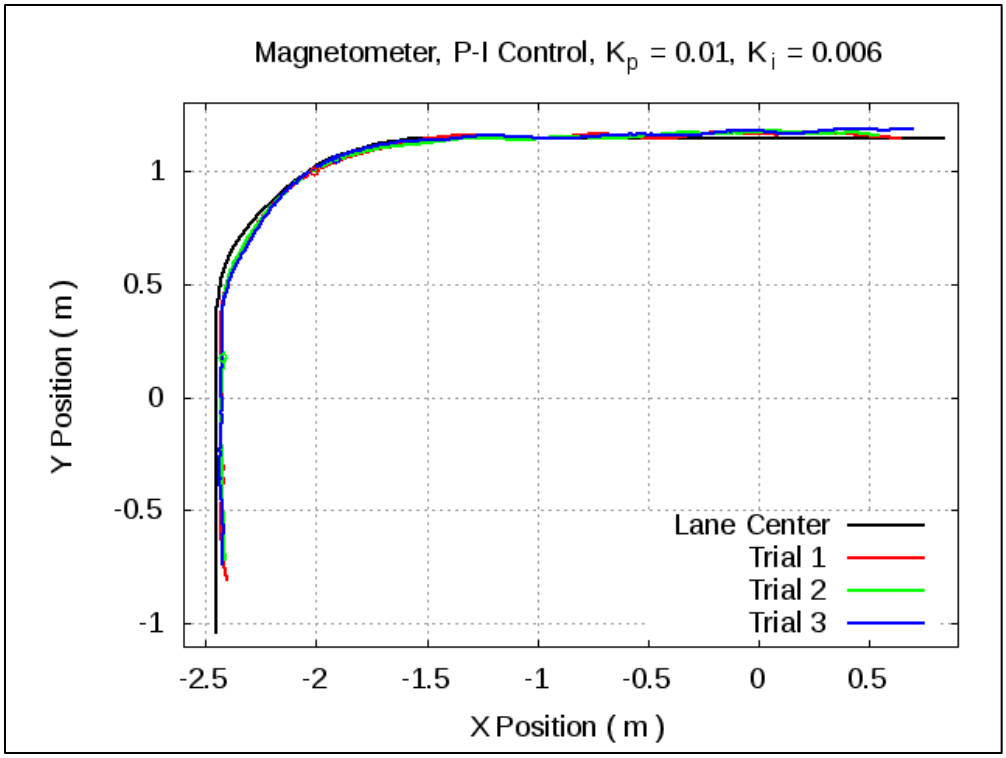

Figure 39: Proportional-Integral Controller Trajectories

Table 3 shows statistics for the proportional controller. The results of the proportional-integral controller are shown in Table 4. All values in both tables are given in Gauss.

Table 3: Proportional Controller Statistics

Trial	Mean	Std. Dev.	Minimum	Maximum	Abs. Mean	Abs. Std. Dev.	Abs. Minimum	Abs. Maximum
1	0.171	0.302	-0.595	1.057	0.272	0.215	0.001	1.057
2	0.126	0.331	-0.827	1.043	0.284	0.212	0.002	1.043
3	0.161	0.363	-0.657	1.183	0.315	0.241	0.003	1.183

Table 4: Proportional-Integral Controller Statistics

Trial	Mean	Std. Dev.	Minimum	Maximum	Abs. Mean	Abs. Std. Dev.	Abs. Minimum	Abs. Maximum
1	0.058	0.340	-0.647	0.949	0.291	0.183	0.002	0.949
2	0.050	0.332	-0.995	0.864	0.273	0.195	0.006	0.995
3	0.058	0.350	-0.603	1.138	0.306	0.180	0.003	1.138

4.4. Area Mapping and Obstacle Avoidance

4.4.1. Environment Setup

To test the functionality of the laser range finder in the Virtual Sensor System, an area was created in which the robot would navigate using only data obtained from a front mounted laser range finder to avoid obstacles in the area and generate a map of the environment. The only other data available was that obtained from the Virtual Positioning System and its other onboard sensors. A three meter square area was blocked out using virtual walls in the Virtual Sensor System. Positioning markers were placed in near the corners of this area for visualization in the physical area. Three physical objects were placed randomly in this area each one approximately fifty centimeters square. A virtual tree was also included; its position was linked to another positioning marker placed in the physical area. Figure 40 shows an overhead view of the world representation in the Virtual Sensor System. Figure 41 shows the physical objects.

The Create robot used in this scenario equipped with a laser range finder virtually mounted 19 cm from the center of the robot. This front facing sensor was configured to have 361 rays within a 180° field of view providing the robot with 0.5° resolution. The maximum distance measured by each ray was set to 0.75 m.

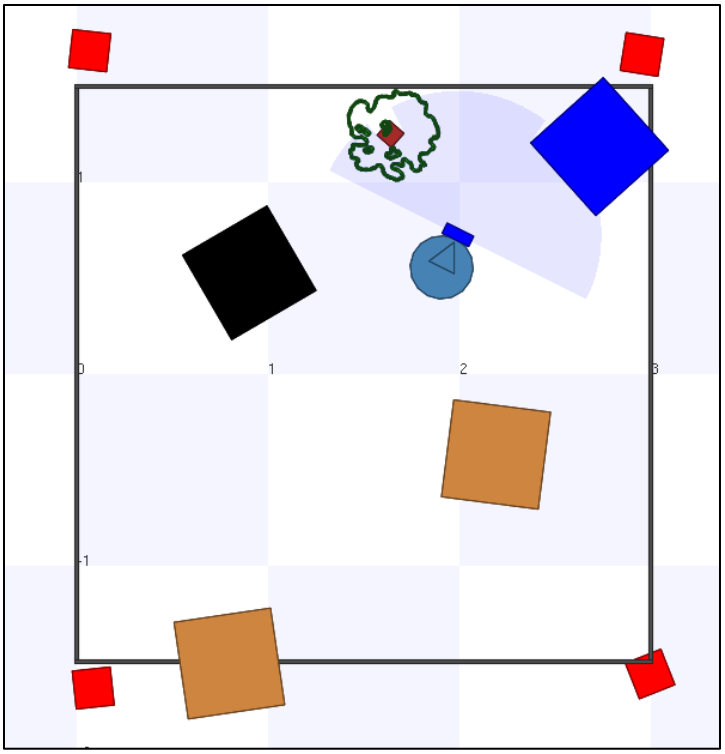

Figure 40: Area Mapping Virtual Environment

Figure 41: Physical Testbed of the Area Mapping Scenario

4.4.2. Map Representation

For the robots representation of the environment, an occupancy grid [35] with three centimeter resolution was used. An occupancy grid is an approach for representing an area which divides an area into squares of a fixed dimension. Each of these squares is assigned a probability based on the certainty that it is occupied by an object. For this simple implementation, each cell was initially recorded to be unexplored, and then as the robot navigated through the environment, each cell explored by the robot was updated to being either occupied or not occupied based on data obtained from the laser range finder.

Because it was necessary for the map representation to be transmitted over Unicast UDP, the map representation was packed in such a way to minimize the number of bytes that required transmission. Two bits were used to represent the state of each cell since there were only three possible states. Sixteen cells representing a four by four grid of three centimeter cells were packed into thirty-two bit unsigned integers. Figure 42 shows how these cells were arranged within the unsigned integer. Bitwise operations were used to mask and shift the two bit segments of the integer when updating the state of a cell or retrieving that cells current state.

A thirty row square matrix of these integers was used to represent a 3.6 meter by 3.6 meter area. This matrix plus two eight bit unsigned integers storing the number of rows and columns used in the matrix was able to be transmitted in only two UDP packets. The full map representation was transmitted at roughly 1 Hz.

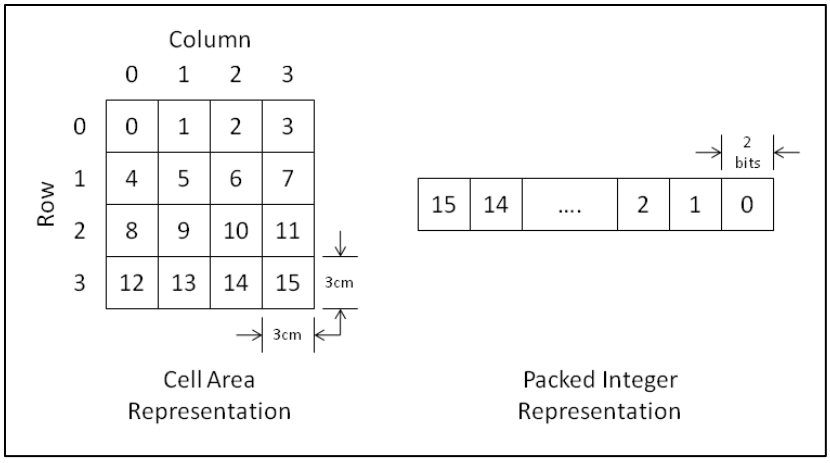

Figure 42: Cell Area Representation to Integer Representation

4.4.3. Updating the Map

To update the map representation of the world, the robot received laser range finder updates at approximately 13 Hz. For each ray in the sensor's field of view, the end point *(P_x, P_y)* of the ray was calculated using Equation $\begin{bmatrix} P_x \\ P_y \end{bmatrix} = \begin{bmatrix} L_x \\ L_y \end{bmatrix} + r \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} + \begin{bmatrix} O_x \\ O_y \end{bmatrix}$ (9.

$$\begin{bmatrix} P_x \\ P_y \end{bmatrix} = \begin{bmatrix} L_x \\ L_y \end{bmatrix} + r \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix} + \begin{bmatrix} O_x \\ O_y \end{bmatrix} \qquad (9)$$

where *(L_x, L_y)* is the position of the laser range finder in global coordinates, r is the range measurement of the individual ray obtained from the Virtual Sensor System, θ is the sum of the angle of the ray, θ_{ray}, and the robot's yaw angle, θ_{robot}, and *(O_x,O_y)* is the offset required to place the physical area into the map area.

The offset used for testing was *(O_x,O_y) = (0.25, -1.75)* to ensure that P_x was greater than zero and P_y was less than zero. The absolute value of P_y was then taken so that P_y was greater than zero. Once the position of the endpoint of the ray was calculated, Equation $\begin{bmatrix} column \\ row \end{bmatrix} = \left\lfloor \frac{1}{3\ cm} \begin{bmatrix} P_x \\ |P_y| \end{bmatrix} \right\rfloor$ (10 was used to calculate the row and column of the cell of the map containing the point. The cell's state was updated to be occupied if the range of the ray was less than the maximum distance and otherwise, it was stored to be not occupied.

$$\begin{bmatrix} column \\ row \end{bmatrix} = \left\lfloor \frac{1}{3\ cm} \begin{bmatrix} P_x \\ |P_y| \end{bmatrix} \right\rfloor \qquad (10)$$

This method only updated the cells on the perimeter of the laser range finder's field of view, however, all of the cells in the polygon created by the ray endpoints and the sensors positions are known to be unoccupied. To add this information to the environment map, all of the cells along each ray needed to be updated. This was accomplished through a line rasterization technique [36] using the row and column coordinates for the calculated endpoint of the ray and the laser range finder position. Figure 43 shows an example ray and its raster graphics representation. First, the change in row value and change in column value were calculated. The algorithm iterated over either the columns or the rows depending on the maximum delta value. The parametric line equation, shown in Equation $P(\ t\ ) = P_0 + t(P_1 - P_0),\ 0 \leq t \leq 1$ (11, was used to determine the cell that lie on the line connecting the two points. If the algorithm

was iterating over the column values, the parametric variable was calculated using Equation $t = \frac{column - C_{min}}{C_{max} - C_{min}}$ (12, and then plugged into Equation $P(t) = P_0 + t(P_1 - P_0),\ 0 \leq t \leq 1$ (11 to calculate the corresponding row value. Each cell was then updated to an unoccupied state.

$$P(t) = P_0 + t(P_1 - P_0),\ 0 \leq t \leq 1 \qquad (11)$$

$$t = \frac{column - C_{min}}{C_{max} - C_{min}} \qquad (12)$$

where C_{min} is the minimum column coordinate and C_{max} is the maximum column coordinate.

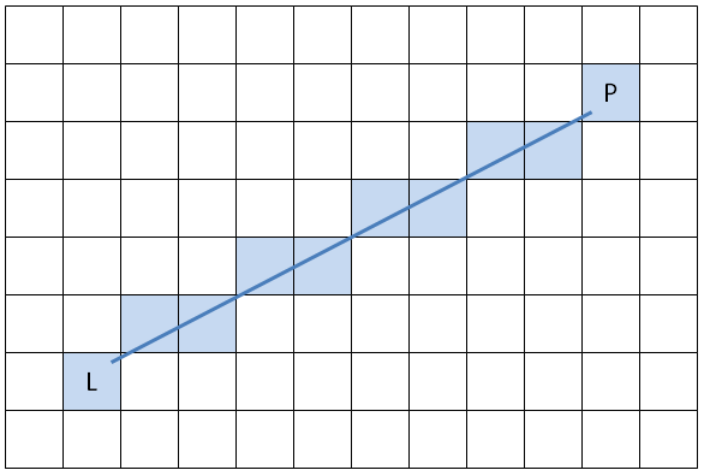

Figure 43: Rasterization of Laser Range Finder Ray

4.4.4. Visualization of the Map

Since the map representation was transmitted over Unicast UDP, any agent on the same network would be able to receive and use this data in some manner. This agent

could be running on any hardware with networking capabilities and written in any programming language that supports the network transmissions. One way would be to draw a visual representation of the map so that the user could determine if it was updating as desired. Such visualization programs would be independent agents and could vary in complexity based on the need of the user.

Two types of visualization programs were implemented for testing purposes. The first was an ASCII based display that was viewed inside of a terminal window. Figure 44 shows a screen capture of this program displaying a map generated by a Create robot in the environment described above. This image was rotated clockwise 90°. For this display, the integer representation of the state of each cell was printed to the screen. Zero was used for unexplored, one for occupied, and a space for the unoccupied cells.

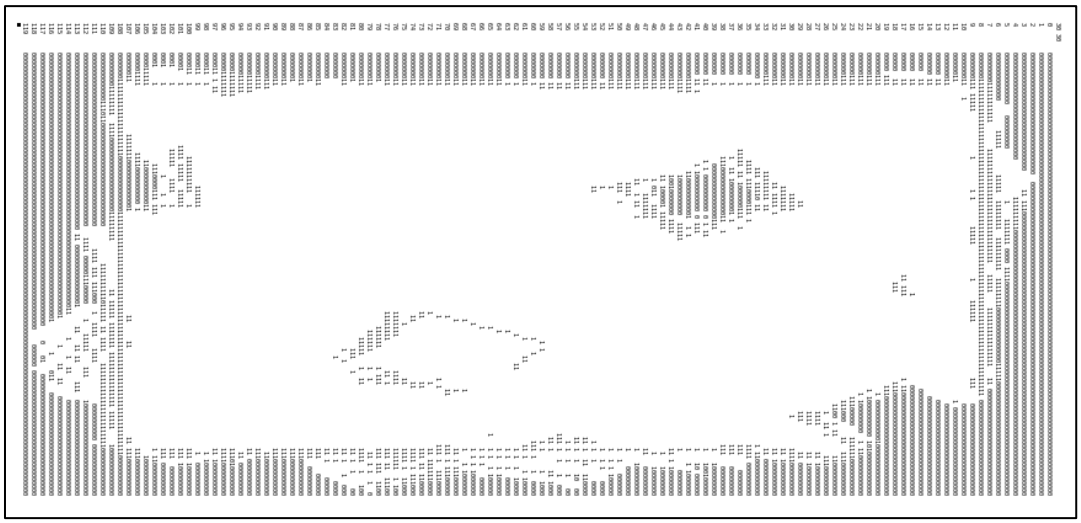

Figure 44: ASCII Map Visualization Program

A second simple graphical based display was also created to show another map visualization possibility. For this display, each cell was drawn as a four pixel by four

pixel square. The color of each cell corresponded to the cell's state. Dark gray was used for unexplored cells, blue for occupied, and white for unoccupied. A single pixel wide grid was also drawn so that individual cells could be distinguished. Figure 45 shows a screen capture of this program displaying the same map that was displayed by the ASCII visualization program.

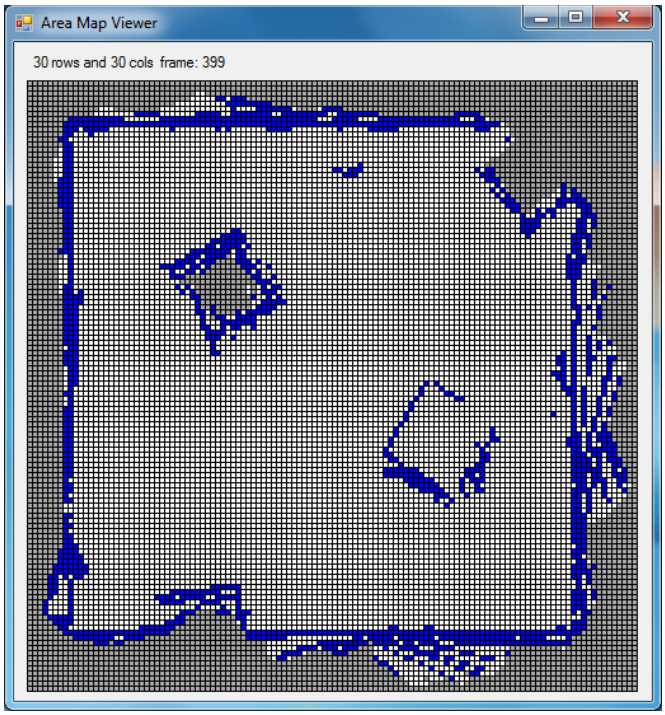

Figure 45: Simple Map Visualization Program

4.4.5. Obstacle Avoidance

The control logic state for the obstacle avoidance was actually a meta-state containing a simple state machine. This state machine is shown in Figure 46. Once the control logic starts, the Create begins driving forward at a speed of 0.1 m/s. The robot continuously processed the data from the laser range finder and would maintain this speed until an object in front of the robot was determined to be less than 0.4 m away.

The robot then reduced its speed to 0.05 m/s until the object was less than 0.2 m away. When this occurred, the robot found the minimum distance in the left and right thirds of the 180° field of view. The robot then turns in the direction having the largest minimum object distance until the object was no longer in the front third of the view. Once this happened, the robot returned to driving straight at a speed of 0.1 m/s.

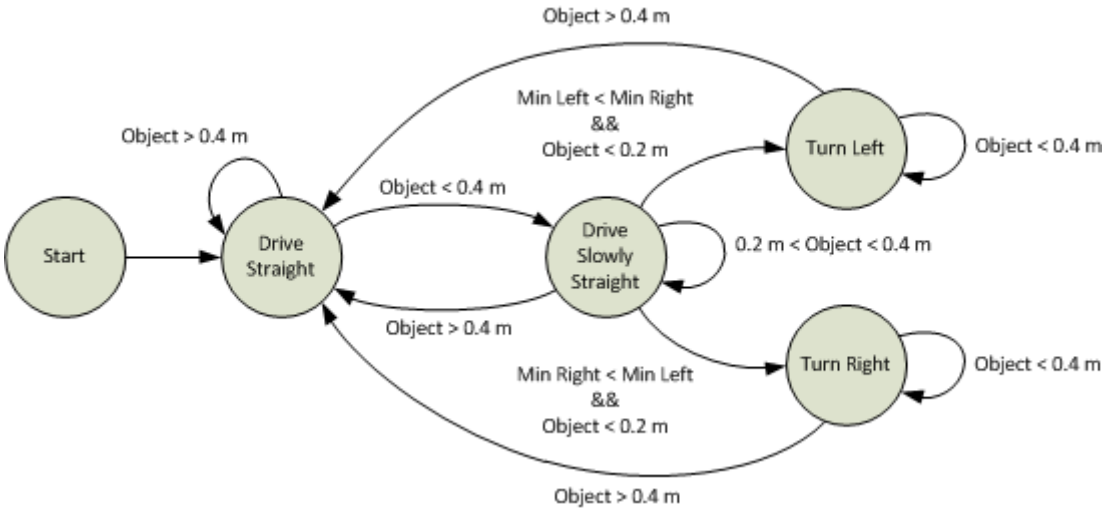

Figure 46: Obstacle Avoidance State Machine

During testing, this algorithm was able to map the simple area with few problems. Because of the simplicity of the algorithm, the robot was susceptible to getting caught in tight places or dead ends as the robot the side obstacles would become front obstacles as it turned in either direction causing the robot to turn in the opposite direction. In this way the robot would oscillate between turning left and right in the area. Some areas may have gone unexplored while others would be explored multiple times because the robot does not favor directions containing a high concentration of unexplored cells.

Chapter 5: Discussion and Conclusion

5.1. Discussion and Comparison

In the path following scenario, both controllers, one using the lane edge sensor and the other a magnetometer array, performed best in the straight portions of the path as expected. In the curved section, the lane edge controller significantly overshot the first half of the curve and then undercut the second. The addition of the integral term helped but did not completely remove this issue. This was likely due to the linear approximation that was used to estimate the distance to the lane edge in the sensor model since the magnetometer controller did not suffer from the same overshooting issues.

The addition of an integral term to the path following controllers showed an error reduction of almost 50% compared to proportional control alone. Although oscillation was still seen in the paths of both controllers, even using only proportional control resulted in a path that was sufficiently within the edges of the lane. This oscillation seemed to be caused by the transmission and computation delays of the virtual sensor data. This delay proved to be a significant issue during the debugging of the control logic and tuning of the controllers. The robots were slowed down to have a linear velocity of 0.1 m/s. When scaled for full-size, this was only about 1.5 miles per hour, a very unrealistic speed.

The oscillation could also be attributed to the Virtual Sensor System using only object state information obtained directly from the Virtual Positioning System and the individual robots at a rate of approximately 10 Hz. If the timing of the systems were out of sync or if a data packet was not received, sensor data may be calculated using the same state of the virtual environment causing the controller to react to the new data as if its previous action had no effect on the system. This would cause the robot to over-correct thus inducing oscillation.

5.2. Conclusion and Future Work

In this thesis, the Virtual Sensor System was created on top of the Stage two-dimensional simulation environment for use with the existing infrastructure in the Control and Intelligent Transportation Research Laboratory at The Ohio State University. This system received state information from the Virtual Positioning System and from several robotic vehicles in order to update a simulation environment representing the testbed. It was possible to represent objects in the physical world with virtual models as well as add virtual objects to the environment without a corresponding physical representation. Also, the virtual models were not required to be at all related to the object used to represent it in the physical world. The Virtual Sensor System also provided the ability to include interactive components such as a traffic light to enhance the visual aspects of the simulation environment.

The primary objective of the Virtual Sensor System was to allow for physical robots to be equipped with virtual sensors in order provide additional inputs to more complex control algorithms. For this purpose, three separate sensors, a laser range finder,

a magnetometer array, and a lane edge sensor, were created. The implementation of these sensors showed that the simulation of the sensor could closely resemble how the sensor behaves in the real world, like the laser range finder, or could utilize the benefits of the simulation environment to generate the data in a way that is much more computationally efficient like the lane edge sensor.

In order to verify the usability of the Virtual Sensor System, two different test scenarios were designed where a control algorithm used the data provided by a virtual sensor to complete the challenge. Two path following controllers were created to show possible uses for the lane edge sensor and the magnetometer array. The laser range finder was utilized in an obstacle avoidance scenario where the robot traversed through the test area and navigating around obstacles while at the same time generating a map representation for the areas already explored.

This research resulted in a system that proved usable in simple Intelligent Transportation applications; however, there were some aspects that could be improved. The most important would be to introduce an extended Kalman filter or similar virtual sensor to estimate the state of the physical objects in the environment. This would provide more accurate simulation results because updates would rely on the continuous state output of the Kalman filter instead of the periodic measurements obtained from the Virtual Sensor Systems and the individual robot's position estimator. This would also allow for sensor update frequencies to vary between sensors though a computationally efficient implementation would require modification of the Stage Simulator to allow for selective updating of objects in the environment.

The ease of use of the Virtual Sensor System can be increased by abstracting the virtual sensor data into the existing sensor interfaces in Player. This decreases the number of interfaces that need to be learned by a developer. This modification would also make the virtual sensor measurements indistinguishable from real sensor measurements allowing a control algorithm to remain unmodified if a physical sensor was purchased to replace the virtual one.

Minor modifications could also be made to Stage's rendering system to allow for objects that can visually update dynamically. Such a change would reduce the complexity of the traffic light component by removing the need for a second hidden model as well as increase the functionality because blinking light states can easily be handled.

Bibliography

[1] E. Wilson, "Virtual Sensor Technology for Process Optimization," in *Symposium on Computers and Controls in the metals Industry*, St. Petersburg Beach, Florida, December, 1997.

[2] G. Heredia and A. Ollero, "Virtual Sensor for Failure Detection, Identification and Recovery in the Transition Phase of a Morphing Aircraft," *Sensors*, vol. 10, no. 3, pp. 2188-2201, March 2010.

[3] J. Stéphant, A. Charara, and D. Meizel, "Virtual Sensor: Application to Vehicle Sideslip Angle and Transversal Forces," *IEEE Transactions on Industrial Electronics*, vol. 51, no. 2, pp. 278-289, April 2004.

[4] D. Dailey and F. Cathey, "Virtual Speed Sensors using Transit Vehicles as Traffic Probes," in *Proceedings of the IEEE 5th International Conference on Intelligent Transportation Systems*, Singapore, 2002, pp. 560-565.

[5] D. Dailey and F. Cathey, "Deployment of a Virtual Sensor System, based on Transit Probes, in an Operational Traffic Management System," University of Washington, Seattle, Technical Report WA-RD 660.1, 2006.

[6] R. Kumar, J. Shin, L. Iftode, and U. Ramachandran, "Mobile Virtual Sensors: A Scalable Programming and Execution Framework for Smart Surveillance," in

Proceedings of the 5th Workshop on Embedded Network Sensors (Hot EmNets 2008), June 2008.

[7] E. Gat, M. Slack, D. Miller, and R. Fiby, "Path Planning and Execution Monitoring for a Planetary Rover," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1990, pp. 20-25.

[8] S. Kabadayi, A. Pridgen, and C. Julien, "Virtual Sensors: Abstracting Data from Physical Sensors," in *Proceedings of the 2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks*, 2006.

[9] Z. Xiang and Ü Özgüner, "Environmental Perception and Multi-sensor Data Fusion for Off-road Autonomous Vehicles," in *Proceedings of the 8th international IEEE Conference on Intelligent Transportation Systems*, Vienna, Austria, 2005, pp. 584-589.

[10] K. Redmill, J. Martin, and Ü. Özgüner, "Sensor and Data Fusion Design and Evaluation with a Virtual Environment Simulator," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, Dearborn, Michigan, 2000, pp. 668-674.

[11] (2010) The Player/Stage Project. [Online]. http://playerstage.sourceforge.net/

[12] T. Collett and B. MacDonald, "Augmented Reality Visualisation for Player," in *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, Orlando, Florida, 2006, pp. 3954-3959.

[13] K. Dixon, J. Dolan, W. Huang, C. Paredis, and P. Khosla, "RAVE: A Real and Virtual Environment for Multiple Mobile Robot Systems," in *Proceedings of the*

IEEE/RSJ International Conference on Intelligent Robots and Systems, 1999, pp. 1360-1367.

[14] (2010) Control & Intelligent Transportation Research Lab. [Online]. http://www2.ece.ohio-state.edu/citr/

[15] S. Biddlestone, A. Kurt, M. Vernier, K. Redmill, and Ü. Özgüner, "An Indoor Intelligent Transportation Testbed for Urban Traffic Scenarios," in *Proceedings of the International IEEE Conference on Intelligent Transportation Systems*, St. Louis, Missouri, 2009.

[16] SAE International, DRAFT SAE J2735 Dedicated Short Range Communications (DSRC) Message Set Dictionary Rev 29, available at http://www.itsware.net/ITSschemas/DSRC/.

[17] (2010) Gumstix. [Online]. http://www.gumstix.com/

[18] (2010) Point Grey. [Online]. http://www.ptgrey.com/

[19] C. Doppler. (2010) Handheld Augmented Reality. [Online]. http://studierstube.icg.tu-graz.ac.at/handheld_ar/artoolkitplus.php

[20] iRobot. (2010) Education & Research Robots. [Online]. http://www.irobot.com/create/

[21] iRobot, iRobot Create Open Interface, available at http://www.irobot.com/filelibrary/create/Create%20Open%20Interface_v2.pdf.

[22] INC. Mobile Robots. (2010) Intelligent Mobile Robotic Platforms. [Online]. http://www.mobilerobots.com/

[23] K-Team Corporation. (2010) Mobile Robotics. [Online]. http://www.k-team.com/

[24] Laird Technologies. (2010, July) Laird Technologies. [Online]. http://www.lairdtech.com/

[25] DARPA. (2007, November) Urban Challenge. [Online]. http://www.darpa.mil/grandchallenge/

[26] DARPA, Route Network Definition File (RNDF) and Mission Data, 2007, available at http://www.darpa.mil/grandchallenge/docs/rndf_mdf_formats_031407.pdf.

[27] Australian Government Defense Science and Technology Organisation. (2010) MAGIC 2010: Super-smart robots wanted for interational challenge. [Online]. http://www.dsto.defence.gov.au/MAGIC2010/

[28] OpenGL, Display Lists, 1997, available at http://www.opengl.org/documentation/specs/version1.1/glspec1.1/node123.html.

[29] SICK. (2010, July) Sensor Intelligence. [Online]. http://www.sick.com/

[30] HOKUYO. (2010, July) HOKUYO. [Online]. http://www.hokuyo-aut.jp/

[31] H. Xu, C. Wang, and R. Yang, "Extended Kalman Filter Based Magnetic Guidance for Intelligent Vehicles," in *Proceedings of the Intelligent Vehicles Symposium 2006*, Tokyo, Japan, 2006, pp. 169-175.

[32] Ü. Özgüner, Autonomy in Vehicles, 2008, Lecture notes from ECE 753.02 at The Ohio State University.

[33] Ü. Özgüner et al., "Simulation and Testing Environments for the DARPA Urban Challenge," in *Proceedings of the IEEE International Conference on Vehicular*

Electronics and Safety, Columbus, OH, 2008, pp. 222-226.

[34] DY. Im, YJ. Ryoo, YY. Jung, J. Lee, and YH. Chang, "Development of Steering Control System for Autonomous Vehicle Using Array Magnetic Sensors," in *Proceedings of the International Conference on Control, Automation and Systems*, Seoul, Korea, 2007, pp. 690-693.

[35] A. Elfes, "Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation," Carnegie Mellon University, Pittsburgh, PA, Doctoral Thesis 1989.

[36] L. McMillan. (1996) Line-Drawing Algorithms. [Online]. http://www.cs.unc.edu/~mcmillan/comp136/Lecture6/Lines.html