4-27-2012

Team Robockey: Final Design Report

Rachel Alley
Trinity University

Alex Butler
Trinity University

Matt Galla
Trinity University

John Kerr
Trinity University

Nathan Richison
Trinity University

TRINITY UNIVERSITY

Final Design Report

ENGR-4381

4/27/2012

Team Robockey

Rachel Alley, Alex Butler, Matt Galla, John Kerr, and Nathan Richison
Dr. Kevin Nickels, Advisor

This report describes the design of the Trinity Robockey infrastructure and the team of two autonomous robots. The methodology for testing the effectiveness of the project's final design is detailed. These tests are designed to provide evidence of whether or not the Robockey project met the objectives outlined in the project charter. The results, final conclusions, and recommendations are provided.

Executive Summary

The purpose of this project was to create the foundation for robotic hockey (Robockey) competitions at Trinity as well as to create a team of autonomous robots to participate in a competition against a team fielded by Trinity faculty and staff. This project was motivated by a similar project undertaken by the mechanical engineering and applied mechanics department at the University of Pennsylvania. Robockey is a multidisciplinary project that may be developed into either an embedded systems project which would utilize a student's electrical engineering and programming skills or a mechatronics project utilizing a student's electrical engineering, mechanical engineering, programming, and design based knowledge.

Several items were provided at the start of the project so that the robots could be built and tested within the allotted time. These items include a 240cm by 120cm rink, a robot localization system, a puck with infrared LEDs built in, and a processor and wireless communication card.

The final Robockey team consists of two robots. Each robot is circular in shape with two tiers, two powered wheels, and two ball casters for balance. The robot chassis is made from Plexiglas cut into a circular shape with cutouts for the wheels and puck bay. The robots hold the puck in the semicircular puck bay. A mechanical switch within this puck bay indicates to the robot when it has the puck in its possession. The processor serves as the brain of each robot, controlling movement, finding the puck, and determining if the robot has possession of the puck. The processor also receives location coordinates specific to the attached robot. The electrical systems of the robot are powered by a rechargeable battery pack that sits on the lower tier of the chassis.

A series of tests were established in order to evaluate the final project design. Each test was designed to evaluate the final design's effectiveness or to provide evidence of how the final design met the project objectives. This set of objectives included: creating a set of rules for a Trinity Robockey competition based upon the rules set forth by the University of Pennsylvania Robockey project that are clear enough to be understood by at least 3 students not affiliated with the project; creating a set of requirements necessary for each robot to enter the competition, creating a team of autonomous robots that will defeat the team fielded by Trinity faculty with at least a 70% success rate; and finally holding a public competition by April 3rd 2012. In comparing the results of these tests to the objectives, a set of regulations were made, a set of rules were made and approved by at least 3 students not affiliated with the project, a team of autonomous robots were built, and these robots were able to defeat a team fielded by Dr. Nickels with a 75% success rate. Unfortunately the public competition was not held as originally intended, nevertheless this project has been be ruled a success.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

The problem at hand was to create a team of autonomous robot hockey (Robockey) players. The idea for this project was taken from a similar project in a mechatronics class at the University of Pennsylvania. Robockey is a multidisciplinary project that has the potential to be developed into either an embedded systems project which would utilize a student's electrical engineering and programming skills or a mechatronics project utilizing a student's electrical engineering, mechanical engineering, programming, and design based knowledge.

Since this is the first time the project has been undertaken here at Trinity, the group also had create the infrastructure for competition including creating a set of rules and regulations for gameplay. Several items were provided at the start of the project to help establish the Robockey infrastructure. These items include a 240cm by 120cm rink, a robot localization system, a puck with infrared LEDs built in, and a processor and wireless communication card.

There are several requirements and goals this project must meet in order to be called "successful." The budget must not exceed the $1200 the engineering department has provided. Each team of robots must consist of up to three players which must not act maliciously. This means that they cannot physically cause harm to the opposing team, interfere with the wireless communication, constrain the puck, or intentionally send out false "puck signals" to the other team. Each robot has to be shorter than 13cm and fit within a 15cm diameter and be fully autonomous. The robots must have an unique 12cm by 12cm glyph attached to their top for the visual locator, and must respond to at least three commands: play, detangle, and stop. Finally our team must beat the faculty's team 70% of the time.

The design of each robot was broken down into three categories, chassis construction, hardware, and software. Chassis construction accounts for the physical layout and design of the robot. Hardware encompasses the control architecture as well as the power, motor control, ifrared detection, and puck detection circuitry. Software includes the programming used to control the microcontroller, the MATLAB simulator, and the base station which is responsible for tracking the robots and sending the play, pause, and detangle commands. These three design subsystems are then integrated to create a successful Robockey player.

This report details the design of the Trinity Robockey infrastructure and team of two autonomous robot players. The report also describes all the testing and results used to measure the effectiveness of the project's final design. Based on these tests and results conclusions and recommendations for future work are offered.

2 Design Description

This section provides a detailed overview of the final robot design including the robot chassis and construction, hardware and software.

2.1 Chassis/Robot Construction

Figures 1 and 2 show different views of a three dimensional model of each robot that is used in this design. The chassis is constructed of Plexiglas because this material is durable and lightweight enough to not put too much strain on the motors. The robots measure 15 cm in diameter and 10 cm in height. They are double layered because there was not enough space on just one layer to fit all of the required components.

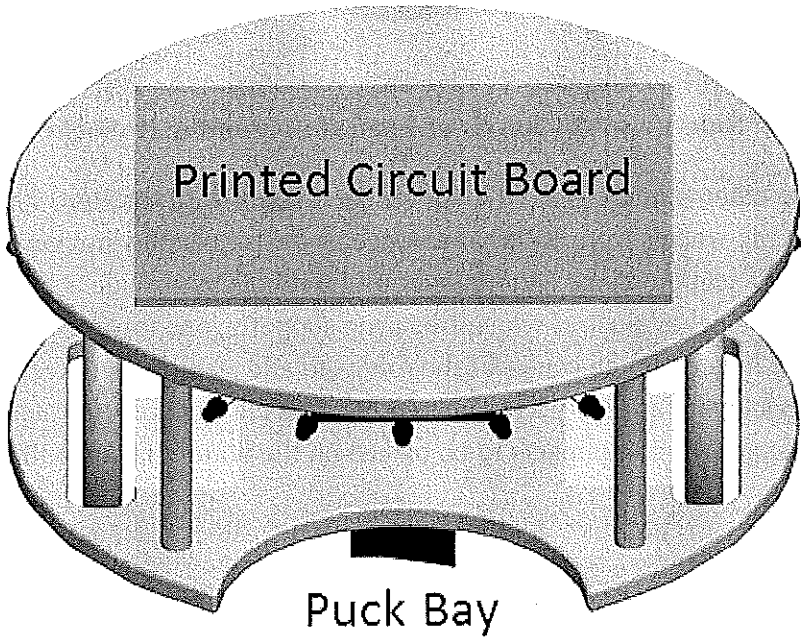


Figure 1: Front View of Chassis of Each Robot on the Robockey Team

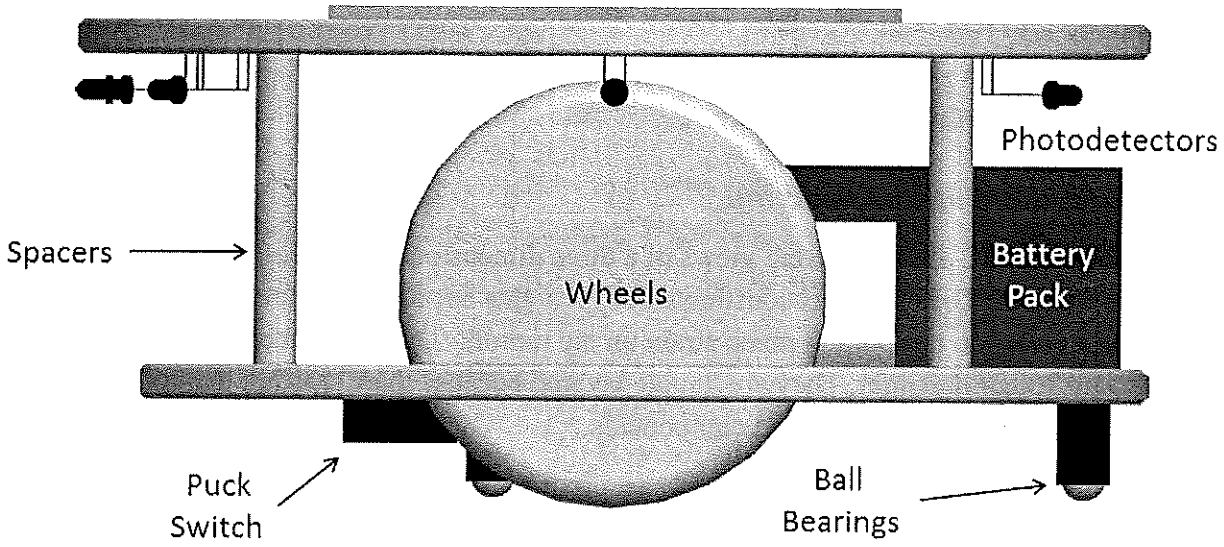


Figure 2: Side View of Chassis of Each Robot on the Robockey Team

There are two cutouts in the bottom layer for the wheels to fit through. This configuration was used to protect the wheels from external damage as the robots play hockey. A semi-circular hole is cut in the front section of the bottom layer of the robots to provide a space for the robot to gather and control the hockey puck. Two ball-bearing supports, one towards the front and one towards the rear, are located in the center of the robots to add stability and keep the robots from falling forward or backward as they play. The two Plexiglas layers are separated by four metal risers which allow space for components to be placed on the bottom layer of the robot.

2.2 Hardware

This section provides a description of the overall control board and a detailed description of each hardware subsystem.

2.2.1 Control Board

The hardware can be broken up into several categories: the power source, the motor driver (h-bridge), the puck detection, the puck sensors (photodetectors), and the processor (M2 board) (Fig. 1). The power source is a 12V battery that has circuitry to regulate the 12V into 9V and 5V. This is because many components need different supply voltages to work properly. The motor driver takes digital inputs from the processor to drive the permanent magnet direct current (PMDC) motors (Fig. 3). The puck detection gives a single digital input to the processor when a puck is captured in the robots puck bay. The puck sensors are built in the top tier of each robot in a 360 degree formation (see §2.2 for details), and can sense the specific frequency of IR light, 940 nm, the puck emits. This is sent to the processor as an analog voltage. The processor, M2 board [7], is the brains of the robot where the programming and logic is stored. Finally, an on/off

LED is built into the circuitry to give off a light when the robot is powered on. This is to prevent the robot from accidentally being left on when not in use, thus draining the battery.

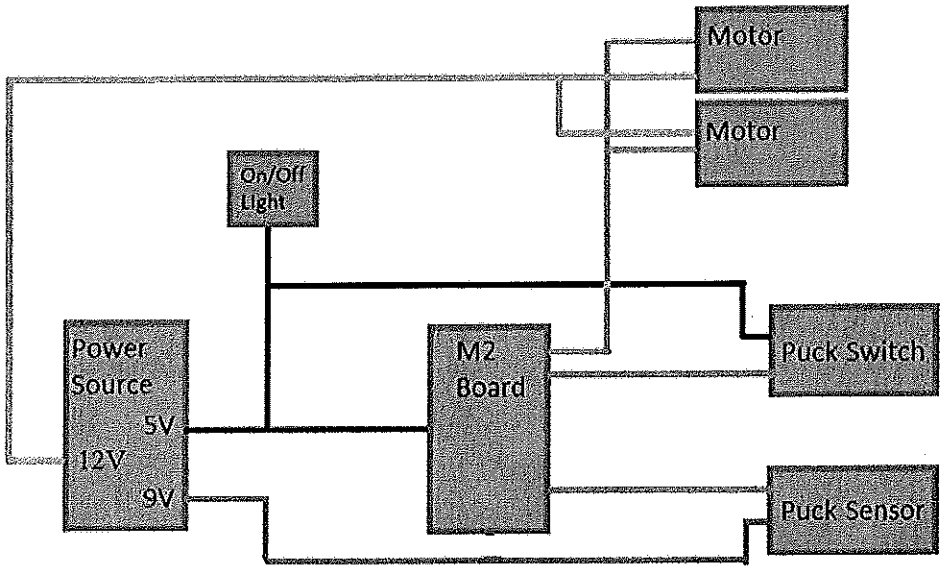


Figure 3: Block diagram of robots hardware.

2.2.2 Power Source

The power source is a 12 V Nickel Metal Hydride rechargeable battery pack for each robot. In order to get this voltage down to 9 and 5 V for other components in the circuitry two voltage regulators are used. These take a somewhat variable DC voltage and produce a smaller constant DC voltage. As seen in Fig. 4, the 12V from the battery source is used to power the 9V regulator (LM7809), and the 9V coming from the 7809 is supplied to the 5V regulator (LM7805). This is called a ladder formation for the purpose of this paper and is implemented to prevent overheating of the 7805.

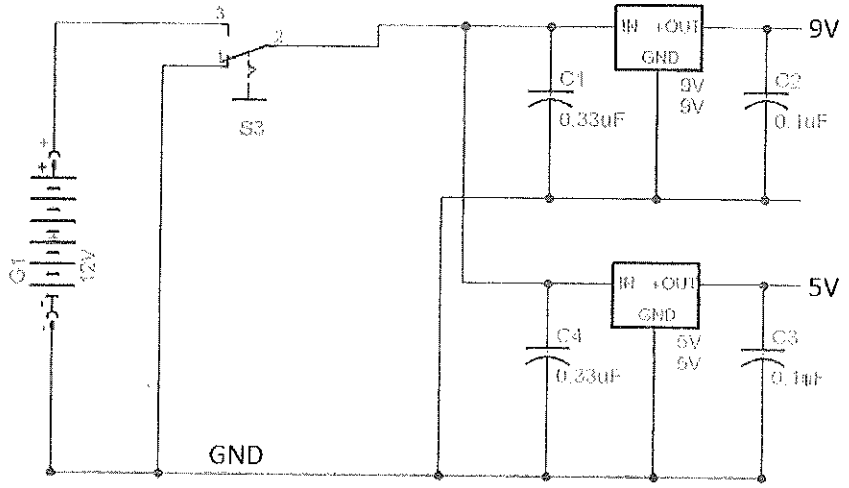


Figure 4: Circuit schematic of power source.

2.2.3 Motors

The motors, GM9 from Solarbotics, were chosen because they have a high torque for a small motor. This is due to the fact that they have a 143:1 gear ratio that increases the angular velocity by 143 times from the original gear attached to the rotor. These permanent magnet direct current (PMDC) motors use permanent magnets surrounding the rotor. The shaft is then magnetically polarized to turn away from the stationary magnet (Fig. 5). This is useful because it creates linear relationships between torque, speed, and current drained (Fig. 6).

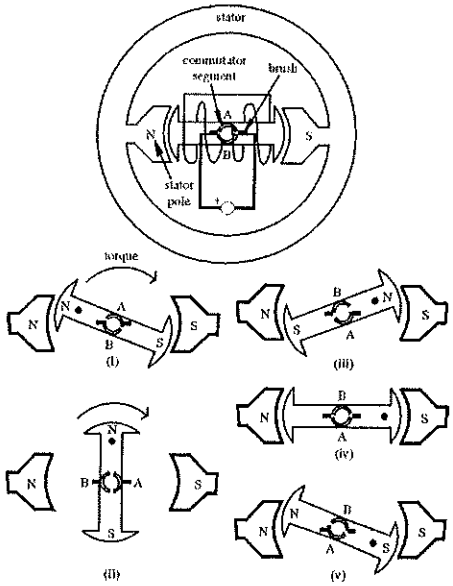


Figure 5: Drawing of how a permanent magnet motor functions. [3]

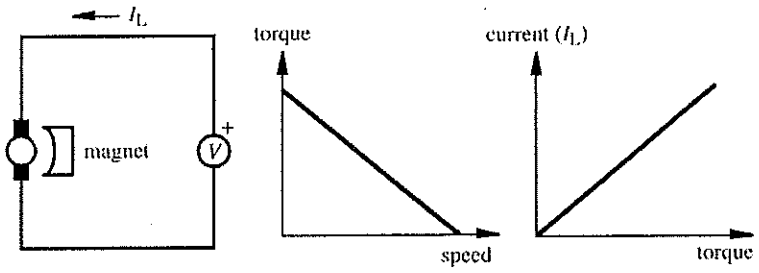


Figure 6: Various relationships for PMDC motors. [3]

Also since these motors are used for other classes like Digital Logic and Embedded Systems it will be easier for students to understand in a Mechatronics class room.

The motors are run by h-bridges that receive two digital signals produced by the microcontroller (Fig. 7). These signals, called pulse width modulation and direction, control the speed and whether the motor runs clockwise or counter-clockwise respectively (Fig. 8)

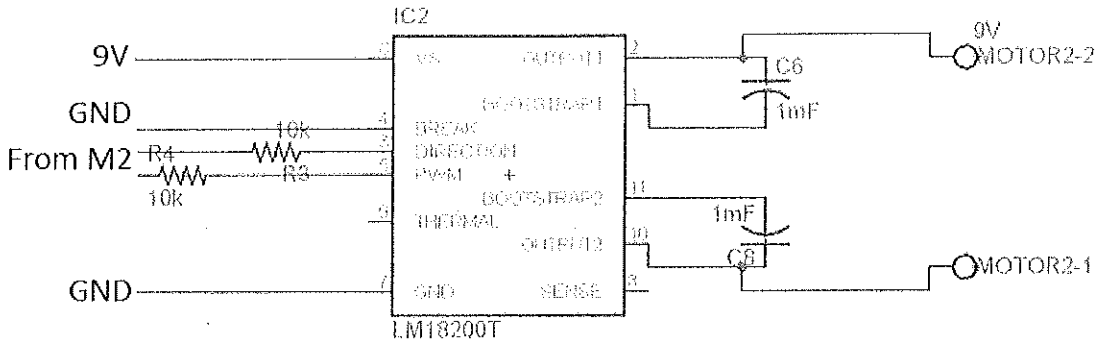


Figure 7: Circuit schematic for motor driver.

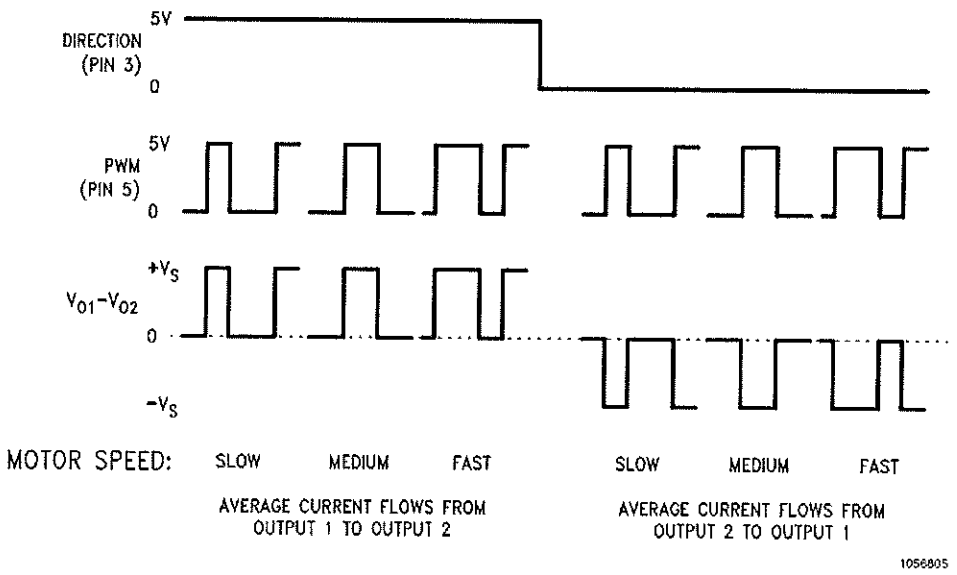


Figure 8: Sign/magnitude PWM control [1]

2.2.4 Puck Detection

A normally closed switch is connected to a pull-up resistor, and if the switch is closed 0V is sent to the processor (M2) and if not the M2 receives a +5V (Fig. 9).

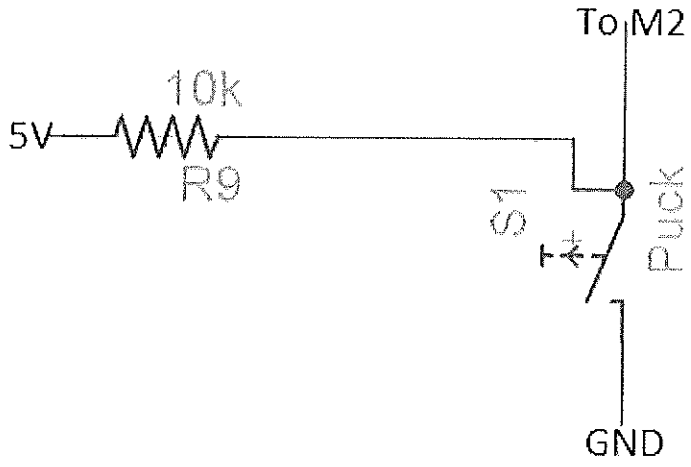


Figure 9: Circuit schematic for puck bay detection.

2.2.5 Photo-Detectors

The robot uses LTR-4206 photo-detectors which detect 940 nm wavelength light (IR) emitted by the LTE-4206 IR LEDs on the puck. Figure 10 shows the circuit that contains the photo-detector. The stronger the IR signal the detector detects, the less resistance there is leading

to ground. Therefore, the stronger the signal, the closer to 0V the output will be. The 6.8V Zener diode (1N4736) allows a path for current to flow if the output voltage gets higher than 6.8 volts. Theoretically this allows the output to stay under a maximum of 6.8 volts. However, in testing, the maximum output voltages have only been up to 5.6 volts. Since the output needs to be restricted to around 5 volts to protect the M2, these values were determined to be acceptable.

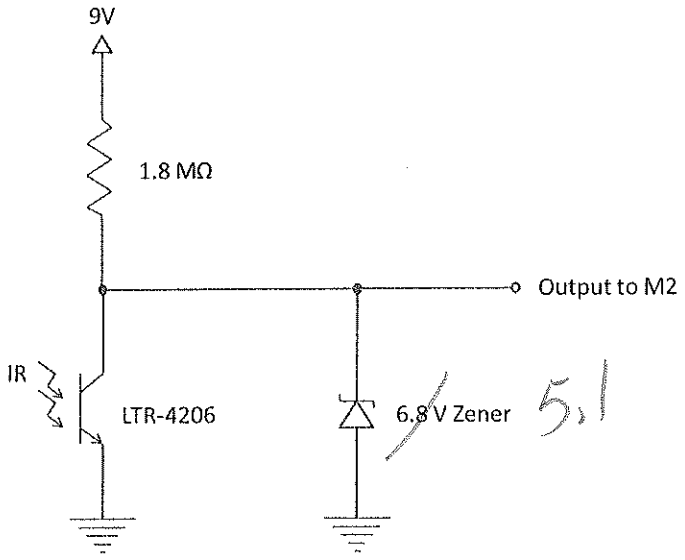


Figure 10: Photo-detector Circuit

2.2.6 M2 Microcontroller

The brain of each robot is an M2 microcontroller, which controls movement, puck detection, photo-detection, and receives location coordinates of the robot it is connected to. The M2 includes an on-board analog to digital converter and utilizes the atmega32u4 microprocessor which contains 32 kB of programmable flash memory, 1 kB of EEPROM, and 2.5 kB of SRAM.

Figure 11 shows a pin-out of the M2, including labels describing pin capabilities and pin connections to different components on the robot. Most pins on the M2 have multiple capabilities as far as what the hardware allows. This adds flexibility in the design when choosing which pin to use for which purpose. The function of each pin is designated in the software when the M2 is programmed.

The motors of the robot require a pulse width modulated signal to determine the speed at which they should run. The M2 has hardcoded pulse width modulation (PWM) capabilities, and can output the signal via any pin with an internal timer connected to it. These pins include B5, B6, B7, C6, C7, D0, and D7. Pin D0 for the left motor and pin B6 for the right motor were chosen because this left other pins open which were more restrictive and were required for other purposes. The direction control output to each motor could be output from any GPIO pin (General Purpose Input/Output). Every pin on the M2 has GPIO capabilities except for V+ and

Ground. Pin D2 for the left motor direction and pin D1 for the right motor direction were chosen because these pins are unneeded by other components.

One other function which required more specific pins was taking in the analog signals from the 9 photo-detectors. These inputs required an analog to digital conversion (ADC) which could only be performed on 12 specific pins. In the pin-out in Fig. 11, each ADC pin is labeled with an ADC #, along with a description of which photo-detector it is attached to (if any). Figure 12 shows a top view layout of the robot with descriptions of ADC pins used for photo-detectors from the pin-out in Fig. 11. Although all of the pins labeled with an ADC# can be used as ADCs, only one conversion can be processed at any one time. The reason for this is that there is only one analog to digital converter on the M2. Therefore, the M2 has to be programmed to set which pin, out of these 12, it is reading at any given time.

The pin used for puck detection can be any GPIO pin because no conversion or manipulation of the voltage received is required; pin E6 was chosen. Finally, the location of each robot was received wirelessly from another M2, which was connected to the base station computer. The Nordic wireless RF communications module is connected to each M2 which allows each M2 to communicate wirelessly with one another.

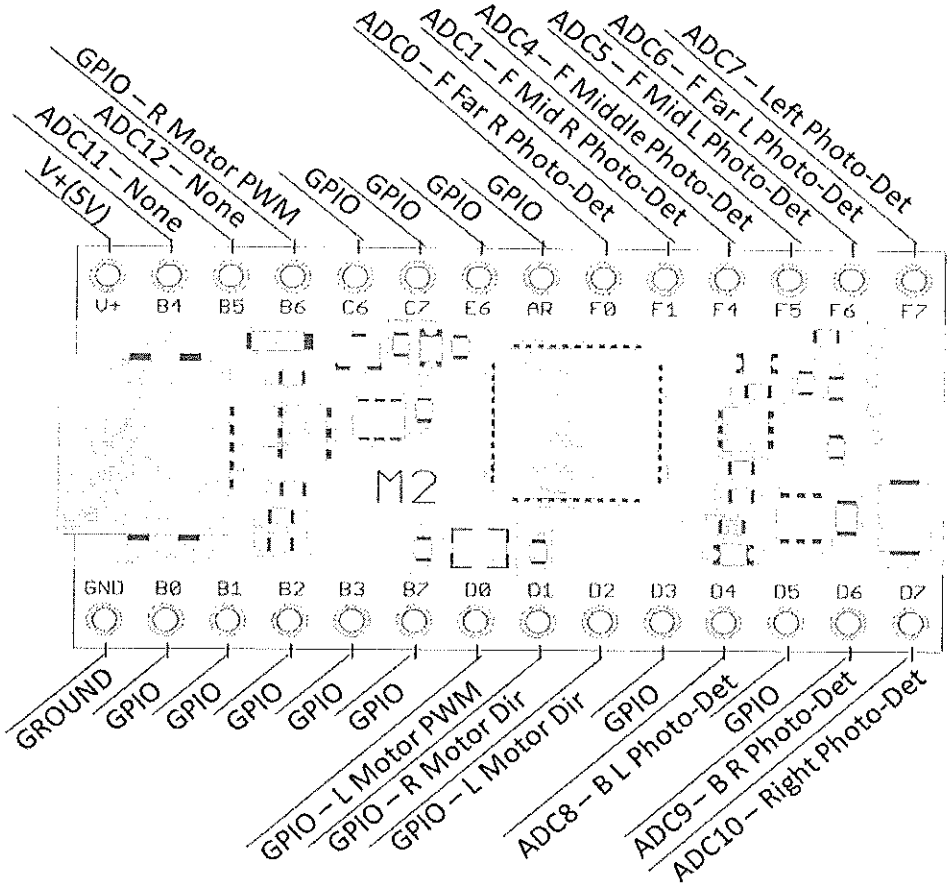


Figure 11: M2 Pin-out

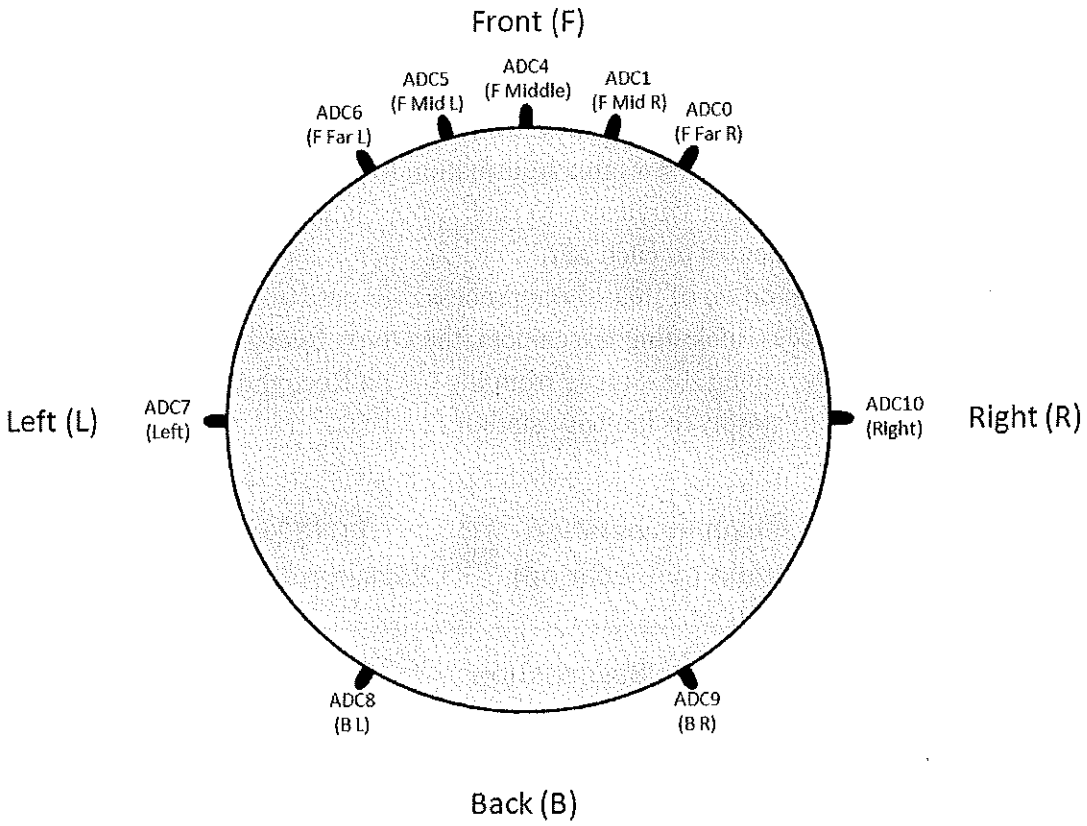


Figure 12: Top View of Robot with Sensor Layout and Pin Descriptions

2.3 Software

The software used to write and compile the code which provides the logic control of the robots is called AVR Studios 6. All code written to control the M2 was written in the C programming language. The program that allows the robots to know the current location they are at is called Glyph Recognition Studio. Each robot is fitted with a unique glyph, which is a picture made of alternating black and white squares to create a symbol. Figure 13 shows examples of these glyphs.

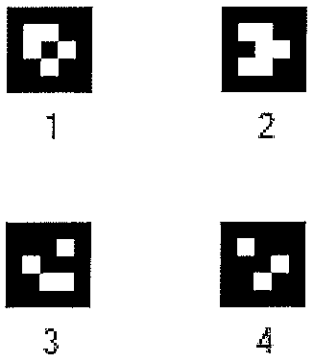


Figure 13: Robot Glyphs Used in (X,Y) Position Tracking

These glyphs are then seen by a camera above the hockey rink and sent via USB to Glyph Recognition Studio on a PC. An algorithm within Glyph Recognition Studio then computes the two-dimensional location of each glyph and transmits this information, as well as any commands entered from a user, to a separate M2 also connected via USB to the PC. This M2 is loaded with a program called tracker_m2 which allows this information to be received from Glyph Recognition studio and also sent out wirelessly to each robot via the wireless RF communications modules connected to each M2.

Figure 14 shows a breakdown of this whole process. For ease of writing, the components directly controlling this process, including the camera, the Glyph Recognition Studio program, and the M2 responsible for relaying the signals from the Glyph Recognition Studio program to the robots, will be referred to simply as the base station.

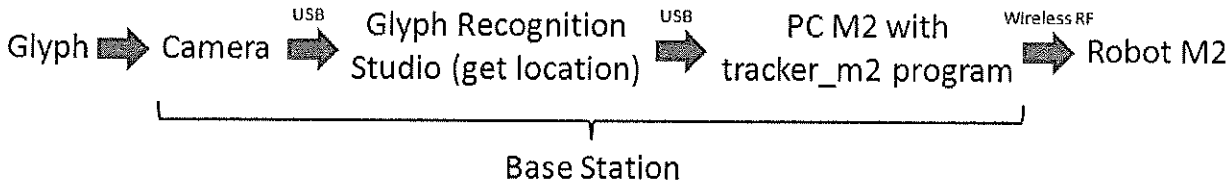


Figure 14: Description of Robot Location Gathering

The base station is also able to send out programmed commands to either all the robots at once, or a single robot of the operator's choosing. The commands currently programmed are play, pause, and detangle. Play allows every robot to start playing hockey at once, pause stops the robots from moving or performing any other functions, and detangle allows stuck robots to become unstuck if the game is being hindered. Figure 15 shows the flowchart for the main program that each robot on the Robockey team is programmed with. Figure 16 shows the flowchart for the find puck function within the main program and Fig. 17 shows the flowchart for the score function.

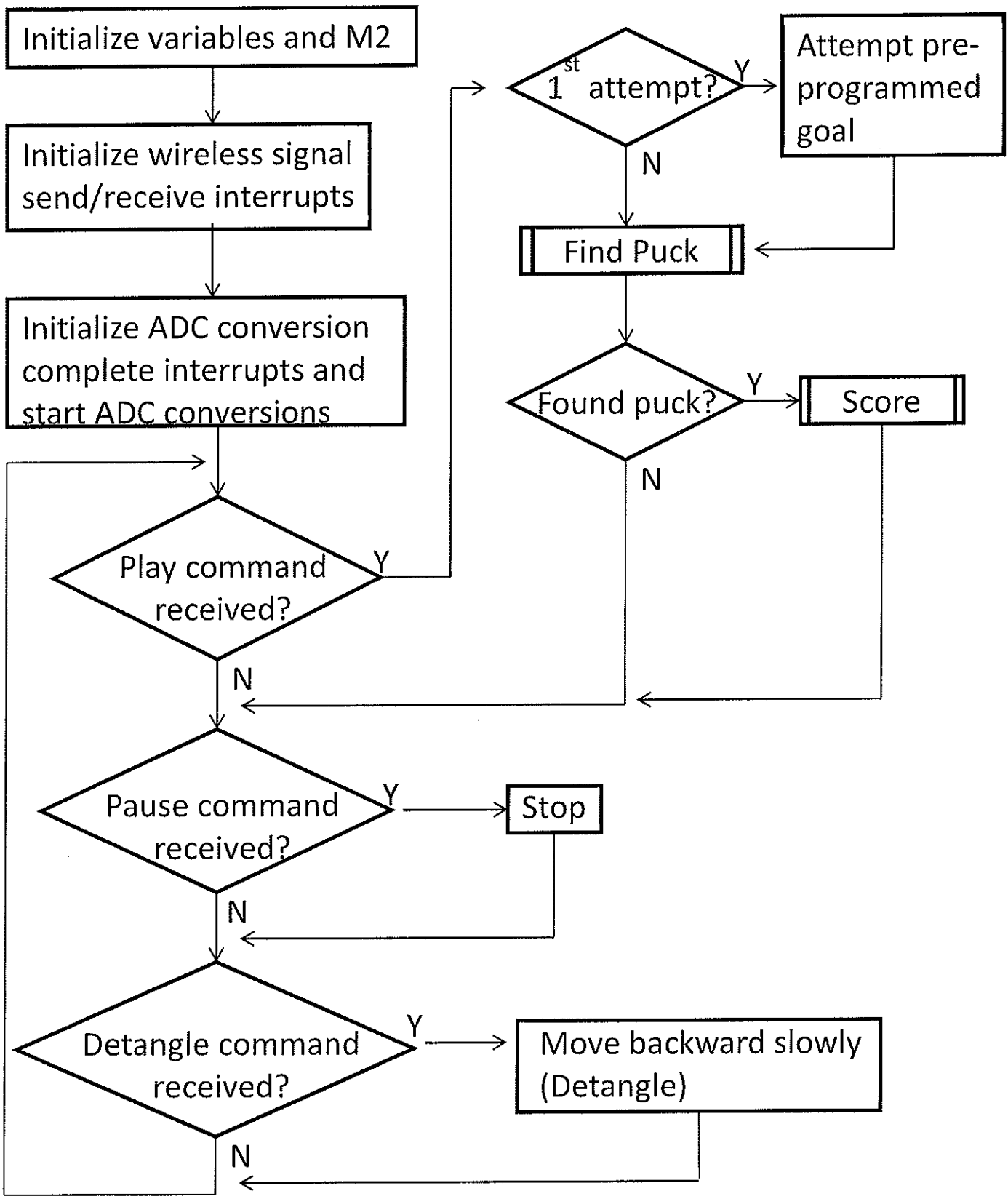


Figure 15: Flowchart for Main Control Program of Each Robockey Robot

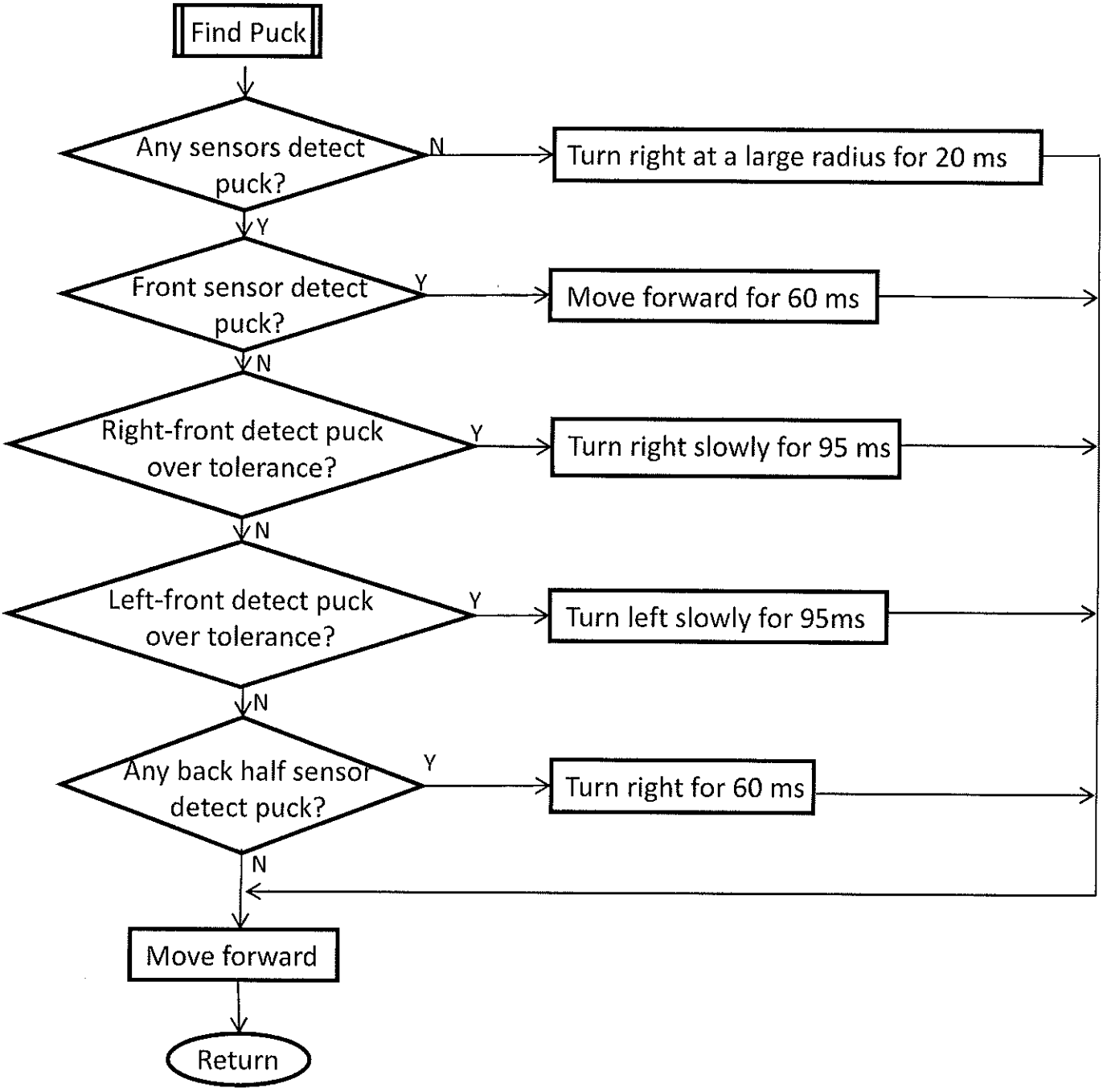


Figure 16: Flowchart for Find Puck Function Located Within Main Control Program

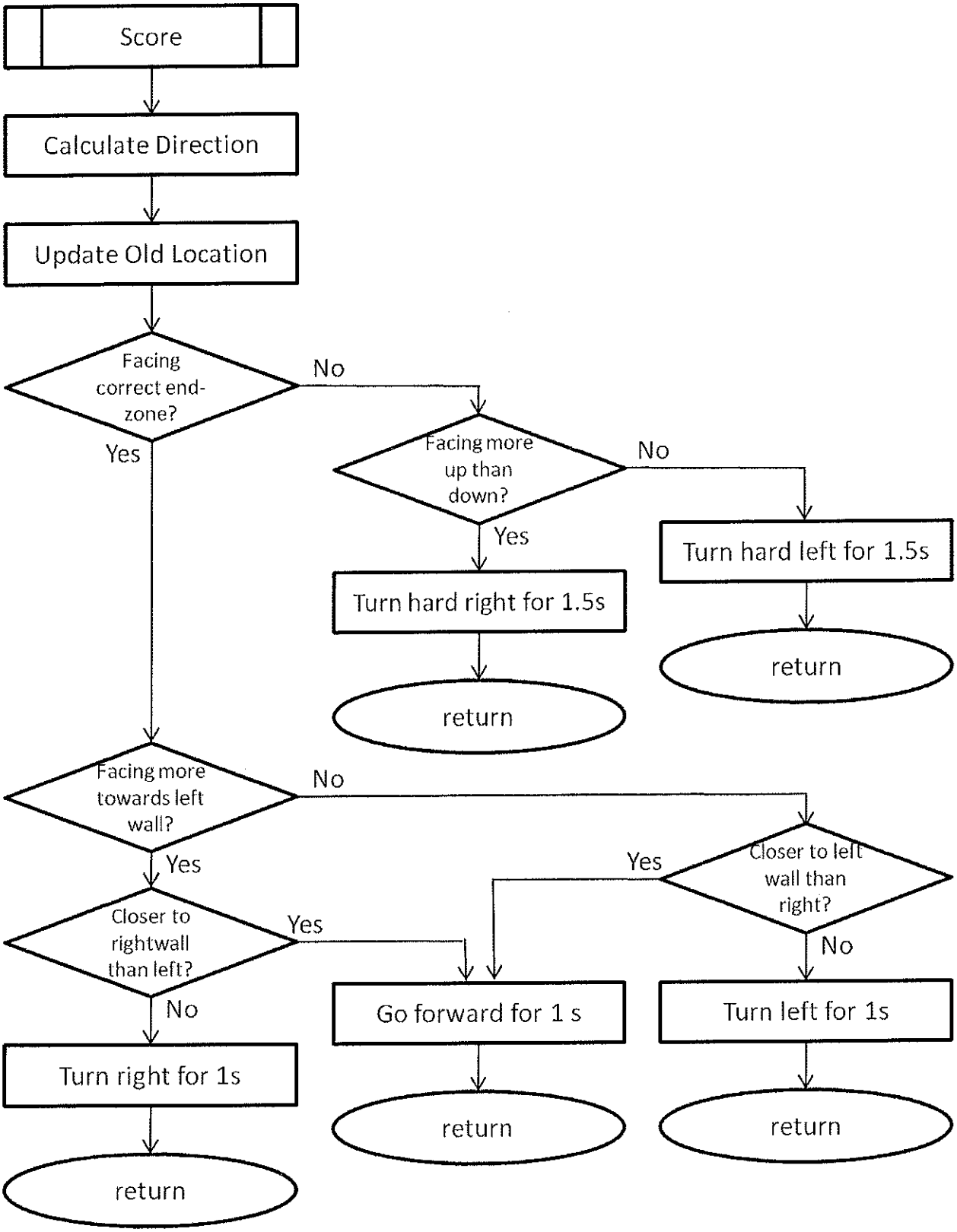


Figure 17: Flowchart for Score Function Located Within Main Control Program

Commented code for the main program, along with all included functions, is given in Appendix F.8. Commands for play, pause, and detangle, as well as location updates are sent as 5 unit length arrays. These arrays can be seen in the code. Table 1 shows a list of which arrays are sent when each command is entered and sent from the base station by a user. The location is broken up into two characters for each coordinate. The first character is the low bit for both coordinates.

Table 1: The commands sent from the base station to the robots.

Command	message[0]	message[1]	message[2]	message[3]	message[4]
Play	0xA1	0	0	0	0
Update Location	0xA2	0xXX(low)	0xXX(high)	0xYY(low)	0xYY(high)
Pause	0xA4	0	0	0	0
Detangle	0xA5	0	0	0	0

3 Methods

This section details the methodology for testing the effectiveness of the projects components and final including test setup and apparatus.

3.1 Hardware

The hardware methods section outlines the testing of the hardware components including the power source, motors, puck detection, and photodetectors.

3.1.1 Power Source

To test the regulators a multimeter and an oscilloscope is attached to measure the input and output voltage of each regulator. This is then checked that the voltage out is a constant 9 or 5V based on a 12V DC source.

3.1.2 Motors

The motor driver, the h-bridge (LMD18200T), is the most robust and expensive component in the circuitry. Due to this fact a series of tests were performed to optimize its use for the robots.

The first test was inspired by Fig. 18 from the components datasheet. This test finds the relation between the frequency of the PWM signal and the current supplied to the h-bridge. By finding a working frequency with a low current drain the battery life can be greatly improved.

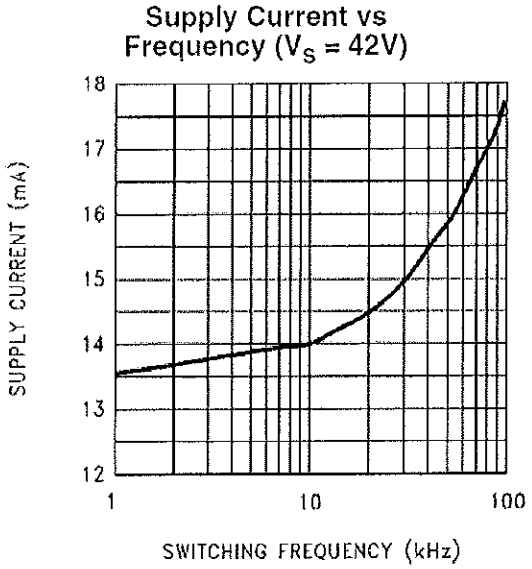


Figure 18: The response of current drained to PWM frequency [1].

This shows that the current drain the h-bridge places upon the battery is directly correlated to the switching frequency of the pulse width modulation signal that controls the speed of the motor. Separate tests were performed in order to reproduce this graph for the conditions the h-bridge will face for the robots in this tournament. The duty cycle of the PWM was set to 50%, and the

supply voltage to around 12V. Then the frequency was adjusted using a function generator as a digital multimeter (DMM) displayed the current going into the h-bridge. An oscilloscope was also used to display the input signal to make sure there were no irregularities.

The next test performed dealt with the relation between supply voltage to the h-bridge and the resulting speed of the motor. As the supply voltage decreases so does the motor speed because there is less available power to supply current into the motor. The real question is by how much does a decrease in voltage cause a resulting decrease in speed, and when does the motor lockout? According to the application note [12] the motor should cut-off at 10V. When a motor is cut-off or locked out this means that the controlling device does not have enough power supplied to it to function. Therefore, the device completely stops proving current and voltage to the motors. The supply voltage to the h-bridge is connected to an adjustable power source on the protoboard that is measured by a DMM. The voltage is then slowly adjusted and the motor speed is judged by watching the motor spin and listening to decreases or increases in noise.

The final test for the h-bridge was to see if the theoretical power drain explained in the application note [12] is similar to the actual loss of power experienced through the component. The three types of power dissipation are quiescent, conductive, and switching.

The quiescent is the largest of these three types and deals with the power drain exhibited across the h-bridge when it is powered, but not told to cause any work on the motor. The conductive deals with the actual equivalent resistances of the transistor switches in the device, and the switching deals with the power dissipated when turning these transistors on and off.

Using the equations and values given by the application note [12] the total theoretical power drain is calculated. Then using the same set-up the actual power drain is calculated by finding power loss across each component. There is no table for this data because these values could change based on different settings. Different tests could be performed with various supply voltages, supply currents, PWM duty cycles, and PWM frequencies. However, this test proved that the theoretical equations for power can be used instead of taking multiple measurements.

3.1.3 Puck Detection

To test the puck detection circuit a program was created that turned on a green LED built into the M2 on whenever it received a digital logic 1, or 5V, from the puck detection circuit.

3.1.4 Photo-Detectors

Tests were conducted to determine the effect of an IR emitter's position (angle and distance) on a single photo-detector. To test this, an IR emitter was placed at three different distances, 6 inches, 3 feet, and 6 feet. At each of these distances, the photo-detector was rotated in 15° increments so that a 180 degree field of vision could be recorded. These results are shown in Appendix E and discussed further in section 4.1.4.

3.2 Software

This section describes the methodology used for testing the software used in controlling the M2 microcontroller, the MATLAB Simulator, and the base station.

3.2.1 M2 Microcontroller

Many tests were performed to verify that every program written on the M2 was working properly and behaved as expected. Additionally, functionality of the Base-Station program and the MATLAB simulator was verified with some simple observational tests. Many of these tests consisted of simply executing the code and observing the resulting behavior of the M2 and comparing it to the expected results. When possible, more quantitative data was used to better ascertain the success of particular programs, but this was often not feasible.

Some programs written for the M2 were only created to learn more about the M2's limitations or confirm that features were accessible and working. Specifically, these early programs verified control of the on-board LEDs, confirmed the accuracy of the analog-to-digital converters packaged within the M2, determined the extent of usable memory, tested the M2 microcontroller's ability to communicate with a computer via USB, and tested the accuracy and reliability of the wireless communication.

The LED control was verified by observing the M2 as it ran a simple program that consisted of the on-board LED blinking at a rate slow enough for the human eye to easily detect. The C code for this test is given Appendix F.1.

Once LED control was established, the analog-to-digital converter (ADC) accuracy was analyzed by writing a program that used an ADC input to control the duty cycle of a pulse width modulated (PWM) signal to the LED. When the M2 ran this program and pulled input from a potentiometer, it was possible to test the ADC accuracy by varying the potentiometer setting and verifying that the LED intensity increased or decreased appropriately. The C code for this test is given Appendix F.2.

The memory of the M2 was tested with a simple program that called for the M2 to store an initial value, flooding the memory with a large amount of extraneous data and confirming that each of these data points could be altered and recalled accurately, and finally recall the original value. One on-board LED was lit to indicate success, the other to indicate failure. This program was systematically altered to increase the amount of extraneous data then ran on the M2 until a failure point was isolated. The final version of the C code for this test is given Appendix F.3.

To confirm that the M2 could communicate with a computer via a USB connection, a simple program was written to print the numbers 1 through 10 on the computer screen in order. This was tested through visual confirmation that the numbers appeared on the computer screen when the program ran on the M2. The C code for this test is given Appendix F.4.

Finally, the wireless communication was tested with a pair of programs loaded on two separate M2 microcontrollers. One program instructed the M2 to initialize a variable to hold a value of 1 and then blink the on-board LED once. The value was then incremented and passed

via the wireless card to another M2, which was to run a separate program ultimately ending in a wireless transfer of the same value incremented again. Again, the M2 was instructed to blink the on-board LED the number of times equal to the received value, before incrementing and passing the value back to the other M2, thereby completing the loop. The program downloaded to the other M2 contained similar instructions, commanding the other M2 to blink the on-board LED the number of times equal to the value received via wireless transmission, increment the value, and finally send the increased value back over the wireless transmission. These two C programs are given Appendix F.5. and Appendix F.6.

None of the above described programs were directly used in the final robot code. The final robot code was constructed in pieces in the forms of smaller programs each designed to complete one specific task. These tasks were receiving the wireless command from the base station to commence the game, facing and later moving towards the puck, and then navigating to the appropriate goal. Each of these programs was tested individually by evaluating the performance of the robot at completing the appropriate task when loaded with one of the programs.

3.2.2 Simulator

The MATLAB simulator was served as a tool in testing navigation logic without worrying about uncertainties introduced by the environment or faulty physical or electrical components. The MATLAB simulator graphically displayed a rink, a puck, and one robot player. It also calculated environmental variables such as the location of the rink walls, the puck location and velocity, the robot location and velocity, and a parameter that represented friction. A method within the simulator represented the logic that could be implemented on within the M2 on the robot. This method used only the robot location and which direction the puck was relative to the robot and gave outputs in the form of wheel speeds and directions. By limiting the inputs, this method simulates all of the variables available to the robot. The logic used in the simulator calculated the robot's direction based on previous location data, and commanded the robot to rotate and then progress towards a point just behind the puck relative to the intended goal.

3.2.3 Base Station

Various aspects of the base station program were tested at different times. The accuracy of the camera was verified by observing that the coordinates displayed by a given glyph changed appropriately when the glyph was moved around the rink. The accuracy of the wireless commands sent by the base station code was verified by programming a separate M2 to display the received commands on a computer via a USB connection. The reliability of the glyph recognition aspect of the code was determined through visual confirmation, as the program displayed the camera feed and highlighted identified glyphs. Finally, the rate at which the base station code sent data concerning the location of each robot was measured indirectly by measuring the time between consecutive blinks of the on-board LED on the base station M2, which would blink once for each command sent.

3.3 Rules and Regulations

According to the project charter a set of rules and regulations were to be established for Trinity Robockey. These rules and regulations were to be based upon those set forth by the University of Pennsylvania Robockey project and can be found in appendix G. In order for these objectives to be met a set of rules and a set of regulations that govern Trinity Robockey must be created; furthermore, the rules must be clear enough as to be understood by at least three students not affiliated with the project. In order to evaluate whether this objective was met a survey was created. This survey presented each rule documented by the group and the survey taker was asked to state whether each rule or regulation was clear or unclear. If a negative response (i.e. unclear) was provided the survey taker was asked to provide an explanation and if possible suggestions for improvement. This survey was then placed on the Student Announcements forum on the Engineering Students T-learn page so that it was accessible to engineering students across all class levels.

4 Results

This section details the results of the aforementioned tests of the projects components and final design.

4.1 Hardware

This section describes the results of the hardware testing including those performed on the power source, motor driver, puck detection, and photodection circuitry.

4.1.1 Power Source

The power sources (7805 and 7809) work in the ladder formation and produce a constant voltage. However, without heat sinks these sources overheat at 12V. This tends to create a large amount of noise as an output.

4.1.2 Motors

Increasing the voltage does increase the motor speed, but this excess speed is not necessary to have a robot fit within the design constraints. The extra resulting cost, weight, and size of batteries with larger voltages would not be worth the gain in speed. For that reason a 12V rechargeable battery was chosen to power the robots.

The cut-off voltage was 0.5V larger than the 10 V voltage (Table 3) described in the application note [12]. However, this should not have a large impact on performance because the batteries chosen are rechargeable and 1.5V larger than the lockout voltage.

Another factor that affects motor performance is the PWM signal frequency. As can be seen by Table 2 and Fig. 19, current drain drops for frequencies above 3 kHz. For this reason we used a PWM frequency of 3 kHz.

The difference between the theoretical and actual power drain was only 4mW, with the actual power drain being larger. This shows how the theoretical power drain equations can be used instead of measuring the voltage and current leaving and entering the h-bridge.

Table 2: Current drain with respect to frequency of PWM signal.

Frequency	Units	Current Into H-Bridge	Units	Notes
0.009794	kHz	0.320	A	very jumpy
0.0999	kHz	0.310	A	fast
1.007	kHz	0.095	A	fast
2.013	kHz	0.071	A	fast
2.959	kHz	0.066	A	fast
4.026	kHz	0.062	A	fast
5.06	kHz	0.060	A	start noticing speed decreasing
6.02	kHz	0.059	A	
7.082	kHz	0.058	A	
7.993	kHz	0.057	A	
9.033	kHz	0.054	A	
10.04	kHz	0.050	A	slowed down drastically
10.26	kHz	0.048	A	
99.1	kHz	0.045	A	virtually stopped

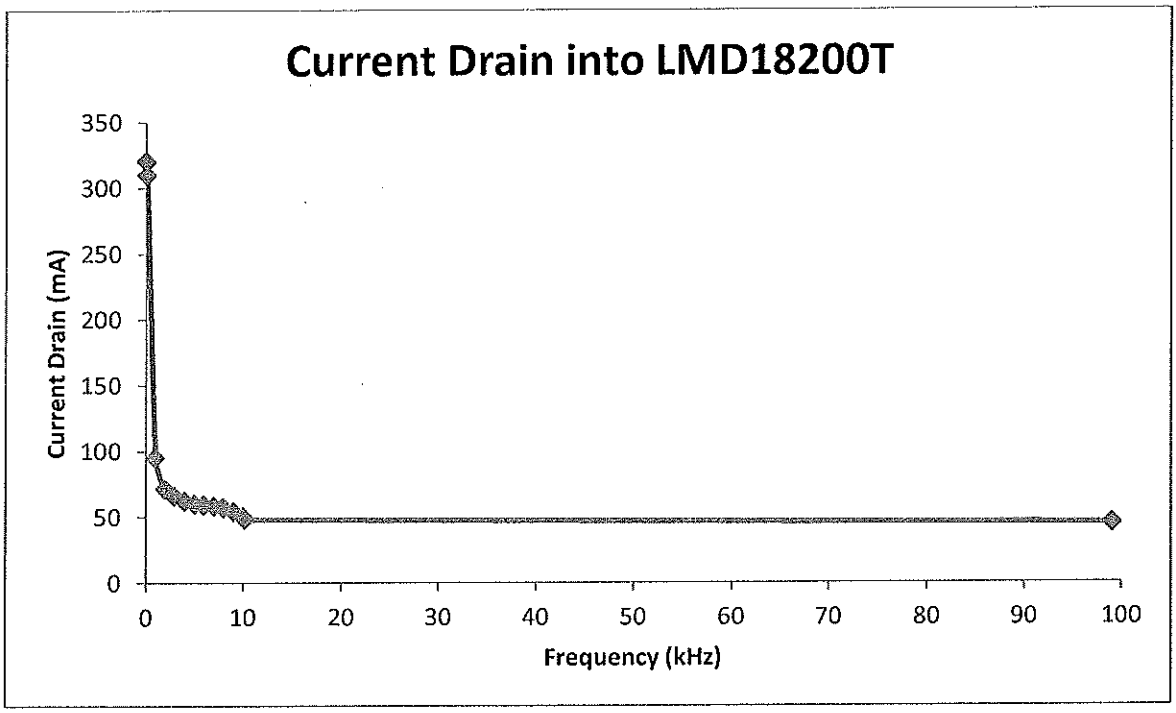


Figure 19: Current drain with respect to frequency of PWM signal (V_s=12V).

Table 3: Notes on the supply voltage to the h-bridge.

V_{IN} (V)	Notes
10.50	Cut-off frequency (no movement)
10.67	Start to see motor speed up
11.00	Motor close to normal speed at 12V
11.68	
11.79	
11.91	
12.08	
15.13	Motor goes even faster no problems overheating
19.58	Fastest possible on board, and no problems with speed
Final Notes	If this project was done in the future the power source should be the largest possible voltage without overloading the bootstrap capacitors.

4.1.3 Puck Detection

There were no irregularities with puck detection during the prototype phase. During the prototype tests the puck was placed in the puck bay and a constant 5V was sent to the M2 without any complications. The only concerns are making sure that the switch is not impeded by any other object on the chassis and that the wiring has no breaks going from the switch to the circuit board. However, when implementing theses switches on the final design there was the occasional noise, but its effect was not significant enough to impede the robots.

4.1.4 Photo-Detectors

Appendix E shows the tests results for close range angle measurements (6 inches), and also for mid (3 feet) to far (6 feet) range angle measurements. The length of each line corresponds directly with the value of the voltage recorded, subtracted from the supply voltage (9 volts).

At close range, angle does not affect the photo-detector very much, but at mid and far ranges, there is a narrow angle of about ±15° in which good signal can be seen. Therefore, when placing sensors around the perimeter of the robot, placing them at 30° increments will provide the processor optimal voltage readings for locating the puck.

Due to restrictions on the number of input pins to the M2, a total of nine photo-detectors are located along the perimeter of each robot rather than the optimal 12 required for full 360° vision with sensors placed at 30° increments. Figure 20 shows the layout of the photo-detectors on each robot. The bulk of the photo-detectors are placed at the front at the optimal 30° spacing to allow the robot to maneuver more precisely once the general location of the puck is determined to be in front of it. With this configuration of photo-detectors, the robot can navigate its way to the puck fairly quickly and precisely.

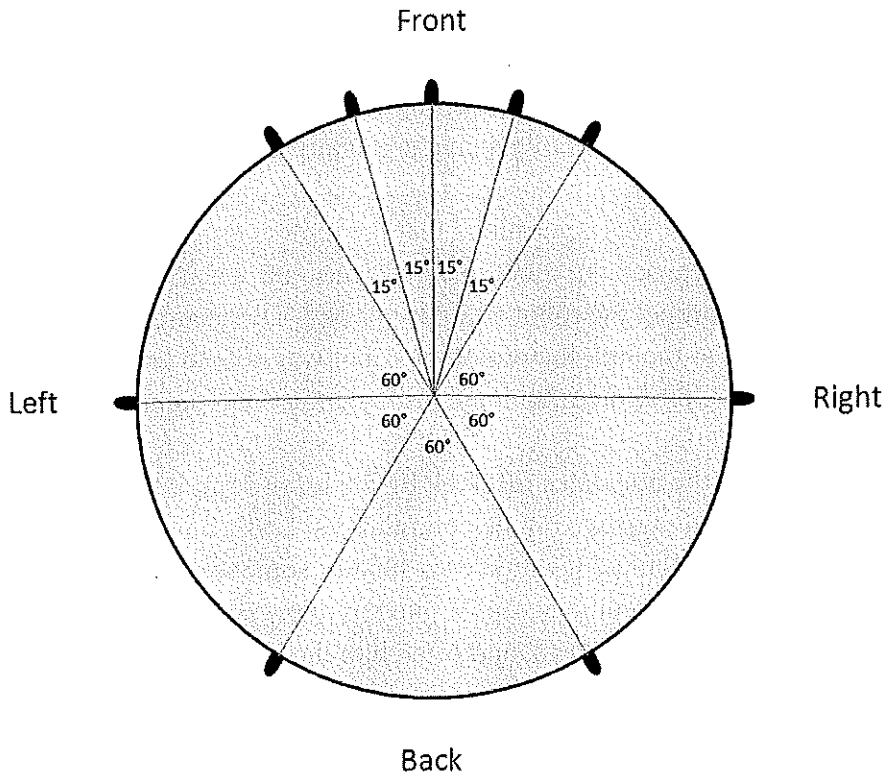


Figure 20: Top View of Robot with Photo-Detector Layout with Angles

4.2 Software

This section describes the results of the testing the software used in controlling the M2 microcontroller, the MATLAB Simulator, and the base station.

4.2.1 M2 Microcontroller

Through the tests described above, it was determined that the M2 microcontroller's LEDs could be controlled by the program running on the M2. It was concluded that the analog-to-digital converters accurately represented a voltage from 0v to 5v with an integer between 0 and 1023, which is a 10-bit resolution. The memory test demonstrated that the M2 could store and recall up to 16,383 integer-type variables, corresponding to approximately 132,000 bits of internal memory. If the settings are configured correctly, the M2 was successfully able to print the numbers 1 through 10 on a computer screen, demonstrating the ability to communicate via a USB connection. Finally, a pair of M2 microcontrollers successfully traded blinking their respective on-board LEDs an increasing, consecutive number of times, confirming that the wireless cards could accurately transmit data wirelessly.

Although factors such as speed, efficiency, and in some cases, accuracy, were not always of the best quality, the prototype robot was able to demonstrate the ability to perform the basic functions required to play a game of robot hockey, specifically locating the puck, controlling the

motors, and communicating with the base station. The prototype robot loaded with the program to respond to the wireless commands from the base station correctly shut off its motors until it received a play command from the base station program and began moving forward. This indicates that the robot is able to correctly receive and interpret wireless commands sent from the base station program.

On the prototype robot, it was determined that the motors did not respond equally to identical PWM inputs. This caused the robot to slowly turn left when the onboard M2 was signaling for the wheels to propel the robot straight forward. Disregarding this tendency for the robot to always aim a little to the left, the prototype robot was able to locate and approach the puck, demonstrating that the programming logic utilizing the ADC signal from each of the IR sensors was sound.

By carefully relaying the messages received by the robot from the base station back to an additional M2 that printed the message on the computer via USB, it was determined that the base station does not always give accurate location coordinates, sometimes sending an old location before updating with the most recent location. While too many consecutive old locations caused the robot to wander blindly across the rink and often into the wall, the program that instructed the robot to use the location data to navigate towards the appropriate goal was able to navigate the robot in the direction of the goal when a fairly consistent stream of up-to-date data was received from the base station.

The results of these tests indicate the possibility of a robot that can combine all of the above behaviors and play a full game of hockey.

4.2.2 Simulator

The MATLAB simulation was found to be successful as well. Navigation logic implemented in the simulator were accurately played out in the simulator GUI and the logic later used in the robot itself correctly instructed the simulated robot to make moves that resulted in a goal. A screenshot of the final simulator program can be seen Fig. 21.

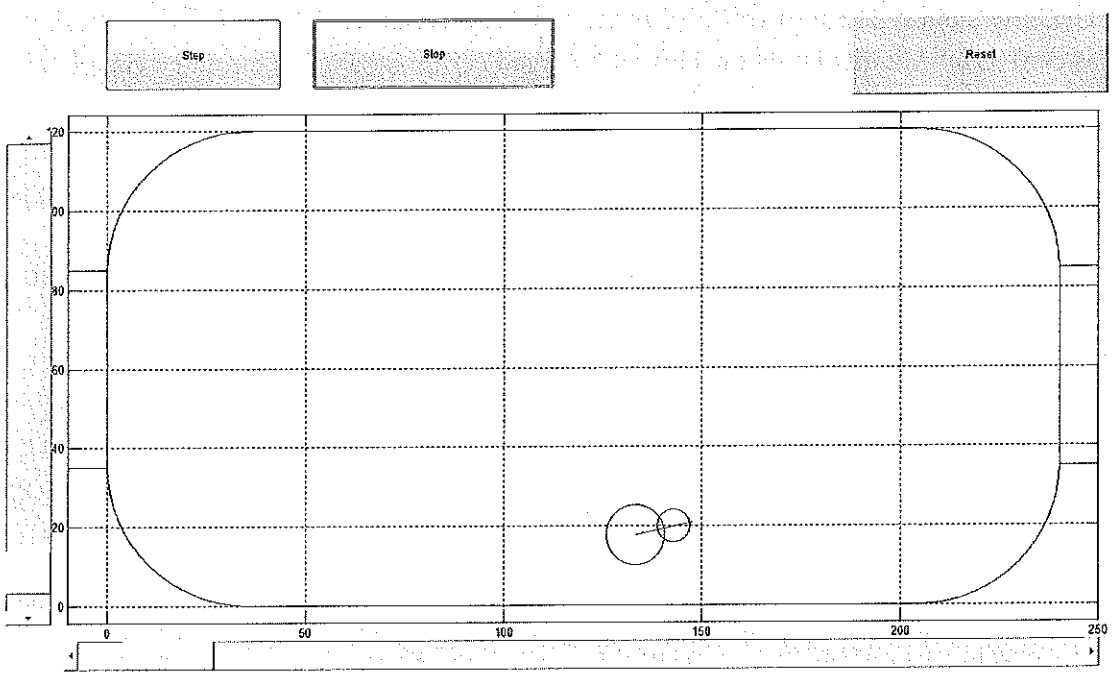


Figure 21: User Interface for MATLAB Robockey Simulator

4.2.3 Base Station

The various features of the base station program were also verified. A coordinate system that covered the entire field was extrapolated from coordinates obtained from the base station program for a single glyph and found to be consistent with further coordinates obtained from the base station program. The generated coordinate system is shown in Fig. 22.

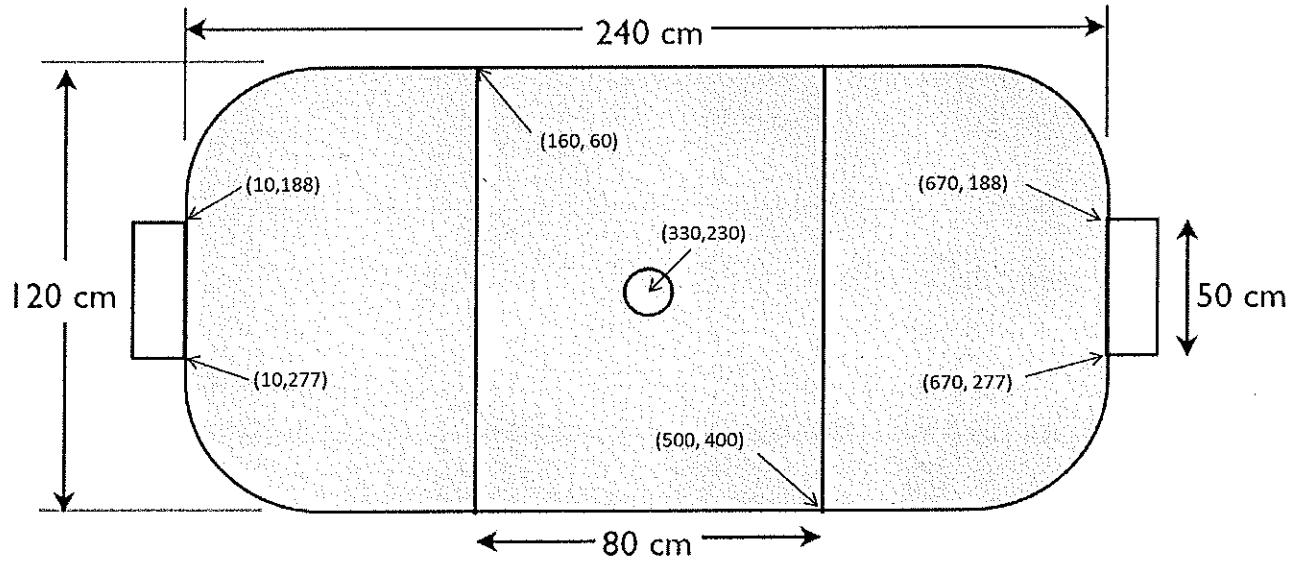


Figure 22: Rink Coordinate Map

The base station program was also found to correctly send specified commands via the wireless card as an M2 displayed the correct received commands on a computer via a USB connection.

While the base station code was initially less reliable at identifying glyphs, it was discovered that closing the aperture on the camera gave better results. In either case, glyphs that were identified by the base station program were always correctly labeled.

Finally, it was determined that the base station program would send out location data to each robot once every 5 seconds, regardless of the refresh rate on the camera. This was determined by observing the green LED on the base station M2, which was programmed to blink on and off once for each message sent. However, for unclear reasons, the location updates were found to be more out of date the longer the base station sent data without being reset. This meant the received command gave a correct, but old location of the glyph, even though the glyph itself has since moved several inches away.

4.3 Rules and Regulations

After excluding obviously non serious responses (based on the negative response explanation provided by the survey taker), each rule was evaluated by at least 11 people and most received a positive response rate of 90% or higher with rule 7 being the exception. Figure 23 provides a visual representation of the positive and negative response rates for each of the 8 rules defined in Table 4. After reviewing the short answer responses the confusion regarding rule 7 was due to jargon, therefore definitions were put in place to clarify this issue. The project objectives as outlined in the project charter stated that a set of regulations and rules must be created and at least three people outside the project must understand the rules. The complete list of rules and regulations can be found in Appendix G and with at least 81% approval for each rule, at least 8 people evaluated each rule as clear, thus these objectives were successfully completed.

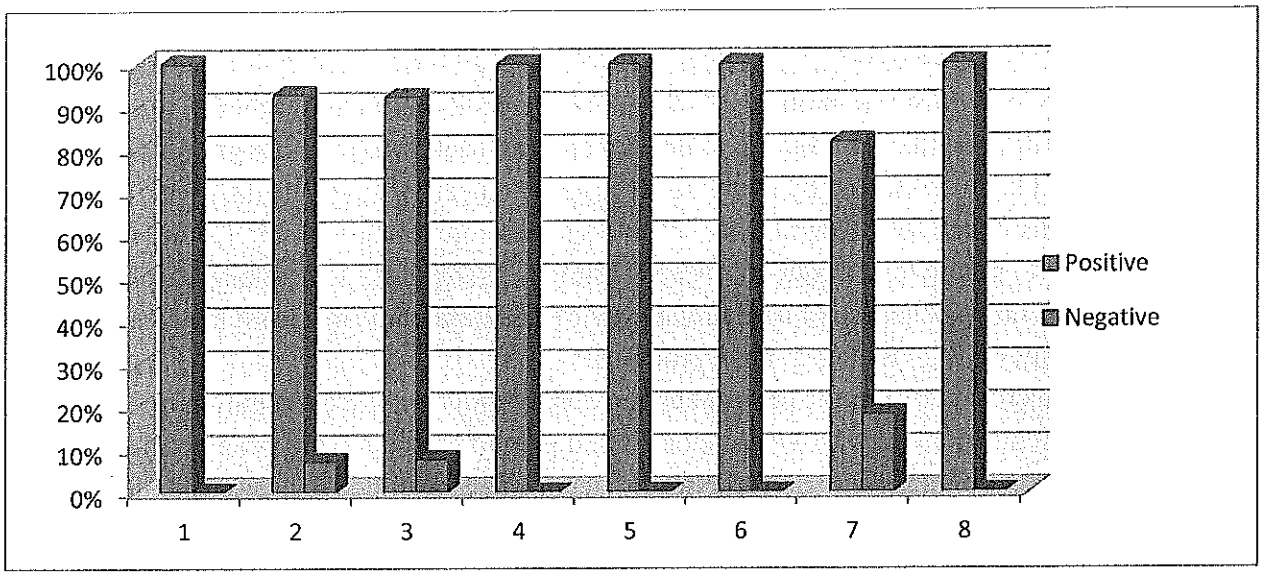


Figure 23: Results of Survey Regarding Clarity of Rules

Table 4: Rule and Rule Number used in Rules Clarity Survey

Rule No.	Rule
1	Before the start of the game, a coin is tossed and the winning team chooses which side of the rink they start on
2	A game consists of three 2 minute periods and there is a 30 second break between periods during which teams may interact with their robots.
3	There will be one overtime period if necessary. After one overtime period, the game goes into sudden death (First team to score a goal wins).
4	Each period starts with the puck in the center and all robots on their respective sides behind a line 40 cm from the center of rink
5	After a goal is scored, the robots and puck return to their starting positions. The timer is paused until game play resumes.
6	The Trinity Robockey tournament is single elimination
7	Each team is randomly assigned a seed number. If there is an uneven number of teams, higher seeded teams may have a first round bye
8	There is a five minute break between games for teams to pack up and the next teams to set up

5 Conclusions and Recommendations

In comparing what we accomplished to the objectives laid out in our project charter, we were successful in everything we set out to do. We created a set of rules and regulations, seen in Appendix G, and had it reviewed by a number of other engineering students via an online survey with very good feedback. The result of our series of games competing with the faculty team was an overall success, mainly due to our strategy adjustment after game 1. The main reason for the strategy adjustment was because of the unreliability of coordinate updates from the robot tracking algorithm through the base station computer.

The last objective stated in the project charter was to hold a public competition open to any engineering students to come watch. While our initial plan to hold a separate competition after our ten round testing matches against Dr. Nickels' team was abandoned due to schedule slips, we still fulfilled this objective by allowing students to come watch our testing. We also realized that trying to move our whole infrastructure including the rink and base station computer to a bigger room for the public competition would have been far too difficult, particularly because the coordinate system the robots use to determine the location of the goals is based upon the exact location in which the rink is placed.

Throughout this project we have learned many things that would be helpful for teams wanting to participate in the Robockey competition in future years. It is our hope that they can use these lessons to avoid many of the mistakes we made along the path.

The first thing the group would do differently if given the opportunity is to focus more attention and research to the kicking mechanism during the initial design phase. We came up with several options in our decision matrix but had we conducted more research we could have eliminated several options, including the one we ended up choosing, which were not suitable solutions. A successful kicking mechanism would have to be very compact yet very powerful. Our final design did not include a kicking mechanism, and we do not believe that it hindered our performance in a significant way. It will be up to the discretion of future groups as to whether they want to spend the time designing a workable solution.

One problem we ran into with our prototype robot was poor motor performance and quick battery drain, which was determined to be caused by high current drain by the H-bridge circuits. As described in section 3.1.2, the current drain the H-bridge places upon the circuitry is a function of the frequency of the PWM signal. In order to keep a good speed while lowering the current drain a frequency around 3 kHz is suggested. There are theoretical power equations to find the power drained by the h-bridge, but using these equations in this application is similar to using a sledgehammer to hammer in a nail. The equations are based on low level physics and electronics thus taking a lot more into account than necessary for actual application. Unless there is a major problem with the h-bridge draining a large amount of current these equations should not be used.

Never checked.

We found that the 5V and 9V regulators would occasionally get very hot when the robot was powered on for an extended amount of time, and this caused fluctuations in the output

voltage. In order to avoid the voltage fluctuations from the voltage regulators heat sinks should be attached to the 7805 and 7809 voltage regulators, and they should not be placed next to each other.

Another recommendation we have for future groups is to check their printed circuit board (PCB) design very carefully before placing an order, paying special attention to pin spacing on components. We made a mistake on our first set of ordered PCBs in which the pins where the M2 microprocessor was supposed to go were too small and also they were surface mount contacts instead of drilled through holes.

Our second PCB design fixed this problem and was also more compact and organized but it was still not perfect. We neglected to think about how we would attach it to our robot until they arrived and had to come up with a workaround.

We recommend future groups include standoff holes to attach the board to the robot using screws. We also recommend that students order one PCB at first and test it before ordering anymore because minor mistakes are likely even if careful attention is paid to detail and ordering a whole new set of boards is very expensive. Having to order a second set of boards also takes up valuable time, as it takes about 17-19 days from the time you submit your order for the boards to arrive. Our boards were ordered from BatchPCB. [2]

The overhead camera and tracking algorithm were listed under the assumptions in our project charter as systems which would be provided to us, but unfortunately it did not work as well as we had hoped. We recommend that future teams look into improving the performance of this system because accurate coordinate updates are necessary for sending the robot to a specific point on the rink.

We set up our schedule so that the first semester was focused on design and proof of concept tests of individual subsystems, building our first prototype robot, and ordering our printed circuit boards. We then planned to use the second semester to testing our prototype, construct our final robots, and program them to function as a team. We did order a set of PCBs before the holiday break, but because of the problems mentioned above, we had to order another set. We also faced schedule slippage because of unforeseen problems with lighting effecting the overhead camera and photodetectors. This caused work on the final robots to be pushed back and decreased the amount we were able to get done. Despite this, we still believe that our schedule was laid out in a well-organized manner.

We think our system of incrementally testing smaller systems before they are put together helped in the process of debugging because we could rule out many of the smaller systems as the cause of the problem. The formal proof of concept testing, documented in Appendix I, was also helpful in checking that all aspects of the prototype worked as they should.

We stayed within the departmental budget of $1200, and our final amount spent was $1080, as seen in the Budget in Appendix A. We could have cut down on costs in a few areas, but our mostly mistake was the one which caused us to have to order a new set of printed circuit boards. We recommend that groups purchase extras of any component that may be fried, because

we did this and it came in handy many times. It is better to spend extra money than to waste time ordering new components and waiting for them to arrive when deadlines are approaching.

6 Bibliography

[1] Institute of Electrical and Electronics Engineers, "IEEE Editorial Style Manual," Institute of Electrical and Electronics Engineers, 2007. [Online]. Available: http://www.ieee.org/web/publications/authors/transjnl. [Accessed 11 January 2008].

[2] "BatchPCB," SparkFun Electronics, [Online]. Available: http://batchpcb.com/index.php/Products. [Accessed 2012].

[3] Solarbotics, *Solarbotics Datasheet for GM9 Plastic Gear Motor,* Solarbotics.

[4] J. P. Fiene, *MEAM.Design : MEAM410-09C-Robockey,* 2012.

[5] J. P. Fiene, "THE M1: A CUSTOM MECHATRONICS PLATFORM FOR ROBOTICS EDUCATION," Montreal, 2010.

[6] J. P. Fiene, "The Robockey Cup: A Look at Mechatronics Education in 2009," 2010.

[7] Atmel, *Datasheet for ATmega16/32U4,* Atmel Corporation, 2010.

[8] LITEON, *GaAlAs T-1 Standard 3 Phase Infrared Emitting Diode.*

[9] J. P. Fiene, "Board Pinout & Functionality," 2012. [Online]. Available: http://medesign.seas.upenn.edu/index.php/Guides/MaEvArM-pins.

[10] D. G. A. a. M. B. Histand, Introduction to Mechatronics and Measurement Systems, 3rd ed., New York: McGraw-Hill, 2007.

[11] Texas Instruments, *Texas Instruments Datasheet for 3A, 55V, H-Bridge: The LMD18200,* Texas Instruments, 1999.

[12] Texas Instruments, *Texas Instruments Application Note for 3A, 55V, H-Bridge: The LMD18200. 1999. Texas Instruments,* Texas Instruments.

A Budget

Vendor	Item Description	Quantity	Cost
University of Pennsylvania	M2 Board	7	$140.00
SparkFun Electronics	Wireless Transceiver	7	$151.94
Trinity Engineering	H-Bridge	8	$141.68
Triniyt Engineering	H-Bridge Board	1	$12.00
HEB	AA Batteries	1	$10.78
Solarbotics	Wheels and Motors	20	$157.48
Batch PCB	Circuit Board - Batch 1	4	$175.71
Batch PCB	Circuit Board - Batch 2	4	$155.25
Mouser	Photodetectors	27	$10.80
LOWES	LED Light Bulb	1	$24.80
Trinity Engineering	Voltage Regulators	6	$6.69
Trinity Engineering	Resistors	45	$0.27
Trinity Engineering	Capacitors	30	$4.20
Trinity Engineering	Switches	9	$8.91
Trinity Engineering	Zener Diodes	27	$1.05
Trinity Engineering	LEDs	3	$0.24
Trinity Engineering	Plexiglass	1	$40.00
Intertex	Spacers	12	$11.98
Trinity Engineering	#2-56 Nylon Insert Lock Nut	20	$8.45
Trinity Engineering	#2-56 Hex Nut	12	$0.84
Trinity Engineering	Machine Screw #2-56 x 1/2"	20	$2.00
Trinity Engineering	Machine Screw #2-56 x 1"	12	$1.92
Trinity Engineering	Aluminum Sheet (0.40 thickness) 1.5" x 13"	1	$0.84
Trinity Engineering	Pololu Ball Caster with 3 / 8" Metal Ball	8	$11.96
Trinity Engineering	Plastic 4" Cable Tie	2	$0.06
Total			**$1,079.85**

B Bill of Materials

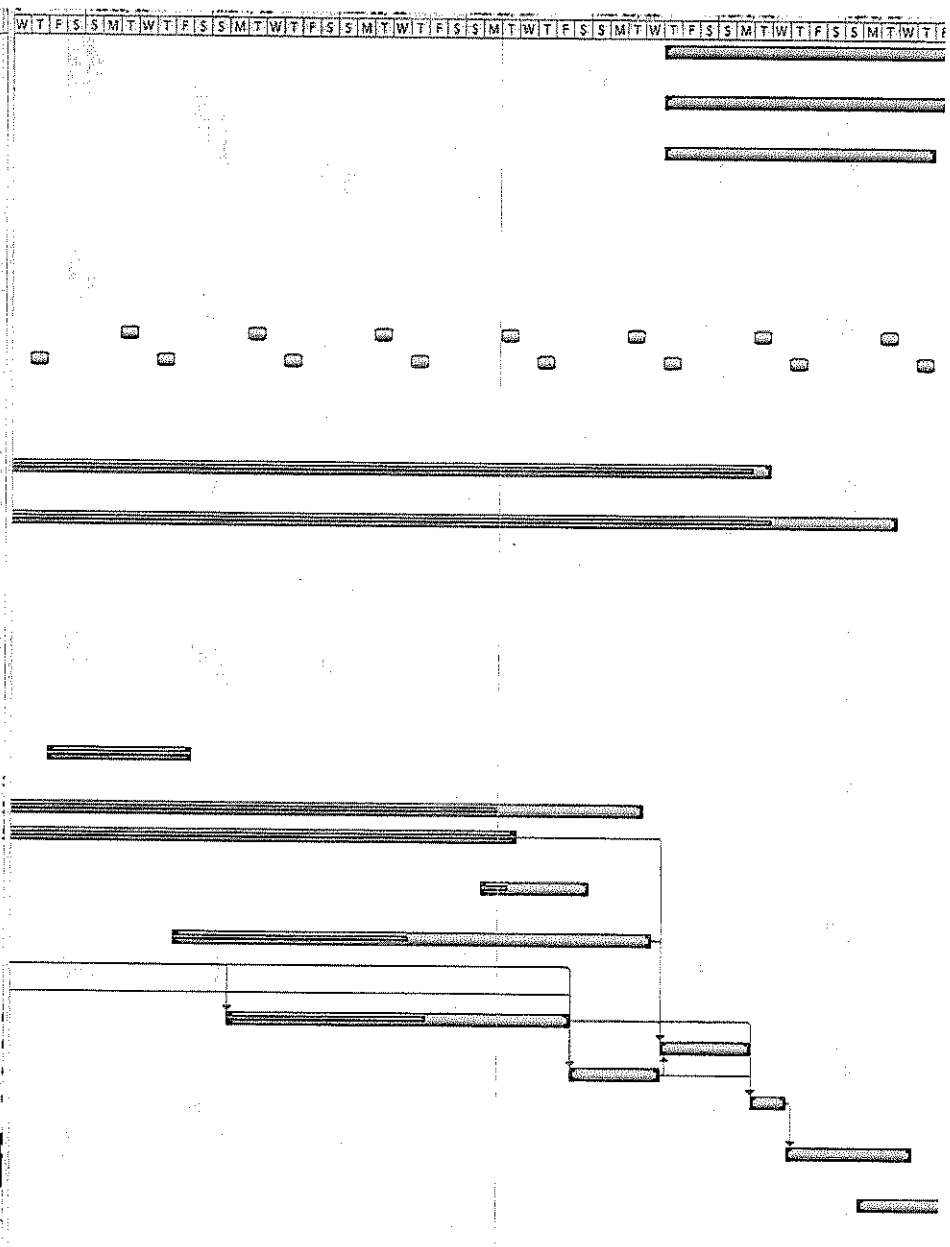
Item	Quantity	Cost/Unit	Total
Chassis Construction			
#2-56 Nylon Insert Lock Nut	10	$ 0.42	$ 4.23
#2-56 Hex Nut	6	$ 0.07	$ 0.41
Machine Screw #2-56 x 1/2"	10	$ 0.10	$ 1.00
Machine Screw #2-56 x 1"	6	$ 0.16	$ 0.96
Aluminum Sheet (0.40 thickness) 1.5" x 13"	1	$ 0.84	$ 0.84
6" x 12" Clear Acrylic Plexiglass Sheet (1/4" thickness)	1	$ 3.49	$ 3.49
Pololu Ball Caster with 3 / 8" Metal Ball	2	$ 2.99	$ 5.98
Aluminum Spacers	4	$ 1.00	$ 3.99
Plastic 4" Cable Tie	2	$ 0.03	$ 0.06
Chassis Construction Total			$ 20.96
Hardware			
M2 Board	1	$ 20.00	$ 20.00
Wireless Transceiver	1	$ 22.13	$ 22.13
H-Bridge	2	$ 17.71	$ 35.42
Wheels and Motors	2	$ 7.98	$ 15.96
PCB	1	$ 36.17	$ 36.17
Voltage Regulators	2	$ 1.12	$ 2.23
Resistors	15	$ 0.01	$ 0.09
Capacitors	10	$ 0.14	$ 1.40
Switches	3	$ 2.97	$ 8.91
Zener Diodes	9	$ 0.04	$ 0.35
LED	1	$ 0.08	$ 0.08
Photodetectors	9	$ 0.40	$ 3.60
Hardware Total			$ 146.34
Robot Total			**$ 167.30**

C List of Vendors

- Batch PCB
- HEB
- Intertex
- LOWES
- Mouser
- Solarbotics
- SparkFun Electronics
- Trinity Engineering
- University of Pennsylvania

D Schedule

		Task Mode	Task Name	Duration	Start	Finish
7			1.4.5.3 Final Report, Final (Internal)	14 days	Thu 3/29/12	Tue 4/17/12
8			1.4.5.3 Final Report, Final (External)	19 days	Thu 3/29/12	Tue 4/24/12
9			1.4.5.4 Final Presentation	11 days	Thu 3/29/12	Thu 4/12/12
10	✓		2.1.4 Project Plan Update (Internal)	9 days	Thu 1/12/12	Tue 1/24/12
11	✓		2.1.4 Project Plan Update (External)	14 days	Thu 1/12/12	Tue 1/31/12
12	✓		2.6 Project Poster (Internal)	4 days	Thu 1/12/12	Tue 1/17/12
13	✓		2.6 Project Poster (External)	9 days	Thu 1/12/12	Tue 1/24/12
14			**+ 2.4 Group Informal Meeting**	**71 days**	**Tue 1/17/12**	**Tue 4/24/12**
30			**+ 2.4 Group Formal Meeting**	**76 days**	**Thu 1/12/12**	**Thu 4/26/12**
47			2.3 CATME	1 day	Tue 2/21/12	Tue 2/21/12
48			2.3 CATME	1 day	Fri 4/27/12	Fri 4/27/12
49			2.5 Executive Summary	1 day	Tue 4/24/12	Tue 4/24/12
50			1.4.5.1 Bill of Materials (Internal)	59 days	Thu 1/12/12	Tue 4/3/12
51			1.4.5.1 Bill of Materials (External)	64 days	Thu 1/12/12	Tue 4/10/12
52						
53	✓		1.4.2.2 Order Motors	11 days	Thu 1/12/12	Thu 1/26/12
54	✓		1.4.2.3 Order Sensors	11 days	Thu 1/12/12	Thu 1/26/12
55	✓		1.4.2.4 Order Processors	11 days	Thu 1/12/12	Thu 1/26/12
56	✓		1.4.2.5 Order Kicking Mechanism	11 days	Thu 1/12/12	Thu 1/26/12
57	✓		1.4.2.6 Order Batteries	11 days	Thu 1/12/12	Thu 1/26/12
58	✓		1.4.2.7 Order PCB	6 days	Fri 2/24/12	Fri 3/2/12
59	✓		1.4.3.3 Solder M2 Boards	10 days	Fri 1/27/12	Thu 2/9/12
60			1.4.3.2 Solder Circuit Boards	33 days	Fri 2/10/12	Tue 3/27/12
61	✓		1.4.3.1 Build Robot Infrastructure (2 Robots)	38 days	Fri 1/27/12	Tue 3/20/12
62			1.4.3.1 Build Robot Infrastructure (3rd & 4th	6 days	Mon 3/19/12	Sat 3/24/12
63			1.4.4.1 Test Circuit Board	18.5 days	Fri 3/2/12	Wed 3/28/12
64	✓		1.4.4.2.1 Program Movement	11 days	Thu 1/12/12	Thu 1/26/12
65	✓		1.4.4.2.2 Program Finding Puck	11 days	Thu 1/12/12	Thu 1/26/12
66			1.4.4.2.3 Program Scoring	15 days	Mon 3/5/12	Fri 3/23/12
67			1.4.4.2.4 Program Teamwork	3 days	Thu 3/29/12	Mon 4/2/12
68			1.4.4.2.5 Program Goalie	4 days	Sat 3/24/12	Wed 3/28/12
69			1.4.4.2.6 Improve Programming	2 days	Tue 4/3/12	Wed 4/4/12
70			1.4.4.2.8 Compete Against Faculty	5 days	Thu 4/5/12	Wed 4/11/12
71			1.5.1 Cleanup	14 days	Mon 4/9/12	Thu 4/26/12
72			1.5.2 Clearance Form	1 day	Fri 4/27/12	Fri 4/27/12



E Photo-Detector Testing and Results

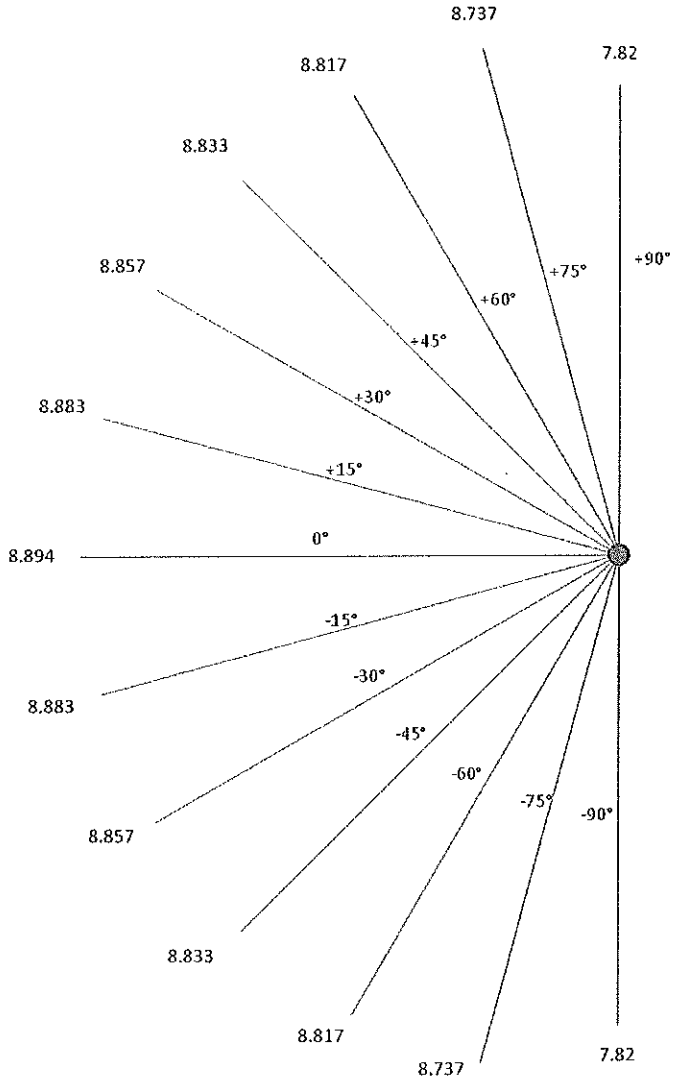


Figure 24: Close range (6" away) voltage readings for photo-detector readings at varying angles
***Note: Values are Difference from Source (9V)**

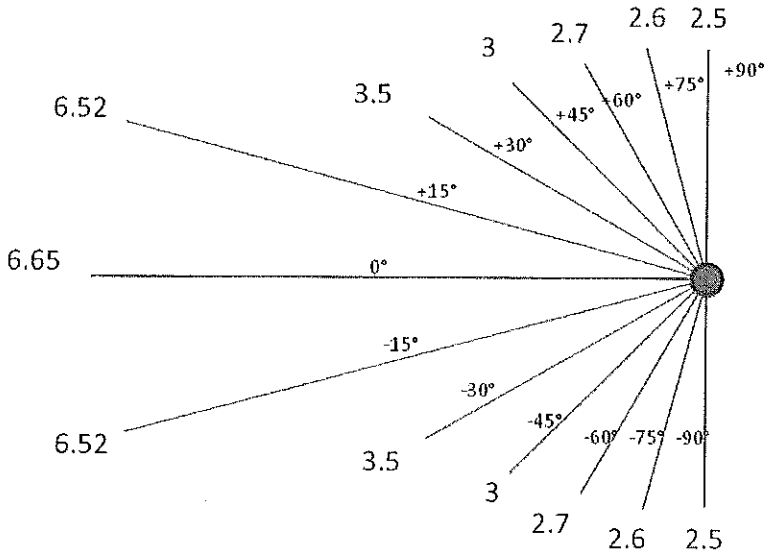


Figure 25: Mid range (3' away) voltage readings for photo-detector readings at varying angles
***Note: Values are Difference from Source (9V)**

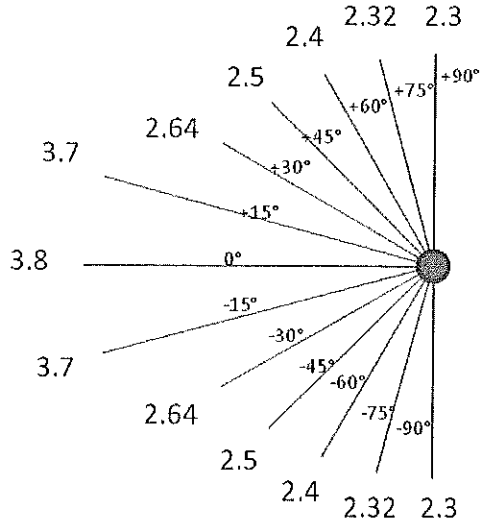


Figure 26: Long range (6' away) voltage readings for photo-detector readings at varying angles
***Note: Values are Difference from Source (9V)**

F Code

F. 1 - 1 Second LED Blink Program

```
// This program toggles the onboard green LED on and off.
// The toggle occurs once asecond, so a full blink occurs every 2 seconds.

#include "saast.h"

void init(void)
{
        //place all init methods here
        m_init();
        m_green(OFF);
        m_red(OFF);
}
int main(void)
{
        init(); // initialize the system

        while(1)                //infinite loop
        {
                m_wait(1000);           //wait for 1 seconds
                m_green(TOGGLE);  //TOGGLE=2 Turns green light on/off
        }
}
```

F.2 - ADC and PWM Test Program

```
// This program reads a voltage and converts it to a integer between 0 and 1023 using the 10-bit
resolution ADC
// It then creates a PWM based on the result to light the onboard LED
#include "saast.h"
//#include "m_general.h"

//Initialization
void init(void)
{
        m_init();
        m_green(OFF);
        m_red(OFF);
        m_usb_init();
}

int main(void)
{
```

```
        init(); // initialize the system
        //set up the PWM timer, using timer #1
        if(m_pwm_timer(1,3000)==0)
                m_red(ON);
        //the following statement activates a pin on timer #1
        //timer #1, pin #1 corresponds to pin B6 on the M2
        if(m_pwm_output(1,1,ON)==0)
                m_red(ON);

        while(1)// infinite loop
        {
                m_wait(100);
                int pwm=m_adc(D4)/10;//m_adc(PIN NUMBER)
                if(pwm>100)
                        pwm=100; //this prevents the pwn from being over 100, even though
integer division should prevent that
                if(m_pwm_duty(1,1,pwm)==0)
                        m_red(ON);

        }
}
```

F.3 - Memory Test Program

```
// This program tests the memory of the M2
// This program resulted in the red light coming on.
// However, if the size of test1 was changed to 1273, the red light did not come on.
#include "saast.h"

void init(void)
{
        //place all init methods here
        m_init();
        m_green(OFF);
        m_red(OFF);
}
int main(void)
{
        init(); // initialize the system

        m_green(OFF);

        int memorybust=31415; //the M2 was asked to remember this value and then recall it at
                                //the end of the program
        //the M2 was given a list of numbers to create
        int test1[1274];
        for(int i=0;i<1274;i++)
```

```
	{
		//first the M2 was required to simply store a number and recall it
		test1[i]=32767;
		if(test1[i]!=32767)
			m_red(ON);
		//second the M2 was required to edit the stored number and recall the edit
		test1[i]=0;
		if(test1[i]!=0)
			m_red(ON);
		//finally the M2 was required to edit the number again, but not recall it until the
		//end
		test1[i]=i+16385;
	}
	//the M2 was required to recall the first number of the array
	if(test1[0]!=16385)
		m_red(ON);
	//the M2 was required to recall the value it was given before the array was ever created
	if(memorybust!=31415)
		m_red(ON);
	m_green(ON);
	m_wait(100000);
}
```

F.4 – USB Connection Test Program

```
// Testing the USB connection

#include "saast.h"
#include "maevarm-usb.h"
//#include "m_general.h"

void init(void)
{
	//place all init methods here
	m_init();
	m_green(OFF);
	m_red(OFF);
	m_usb_init();
}
int main(void)
{
	init(); // initialize the system
	//initialize usb
	m_green(ON);
	// wait for a connection, note the semicolon!
	while(!m_usb_isconnected());
```

```
        m_green(OFF);
        m_wait(20000);
        //output the numbers 1-10
        int t=1;
        while(t<11)
        {
                //Send t as an integer over the USB connection
                m_usb_tx_int(t);
                t++;
        }
        // the wait here prevents the code from running over and over again, flooding the system.
        m_wait(314159);
}
```

F.5 – Wireless Communication Test Program (Ping)

```
// Testing the wireless communication
// This is Ping. This M2 sends the first command

#include "saast.h"
#include "maevarm-rf.h"
#define PACKET_LENGTH 1 //PACKET_LENGTH denotes how many characters are sent in
                                //one message by the wireless card

char count[PACKET_LENGTH] = {0}; //for this program count is an array of length 1 (just a
                                                //single char)
char me[5] = {0xAB, 0xAB, 0xAB, 0xAB, 0xAB};
char you[5] = {0xCD, 0xCD, 0xCD, 0xCD, 0xCD};

void init(void)
{
        //place all init methods here
        m_init();
        m_green(OFF);
        m_red(OFF);

        //this method has this M2 "claim" the address 'me'.
        //Every message this M2 interacts with must be of length PACKET_LENGTH
        //(otherwise adjacent data may get overwritten!)
        RFsetup(me,PACKET_LENGTH);
}
int main(void)
{
        init(); // initialize the system
        while(1)
        {
                for(char i=0;i<*count;i++)
```

```
        {
            //blink the number of times stored within count
            m_green(ON);
            m_wait(500);
            m_green(OFF);
            m_wait(500);
        }
        //add one to the number of times to blink
        *count=*count+1;
        //send the updated count to the other M2 (to pong)
        RFtransmitUntil(count,you,50);
        //wait for a new message
        while(!RFRXdataReady());
        //receive the message
        RFreceive(count);
    }

}
```

F.6 – Wireless Communication Test Program (Pong)

```
// Testing the wireless communication
// This is Pong. This M2 receives

#include "saast.h"
#include "maevarm-rf.h"
#define PACKET_LENGTH 1

char count[PACKET_LENGTH] = {0}; //for this program count is an array of length 1 (just a
                                 //single char)
char you[5] = {0xAB, 0xAB, 0xAB, 0xAB, 0xAB};
char me[5] = {0xCD, 0xCD, 0xCD, 0xCD, 0xCD};

void init(void)
{
    //place all init methods here
    m_init();
    m_green(OFF);
    m_red(OFF);

    //this method has this M2 "claim" the address 'me'.
    //Every message this M2 interacts with must be of length PACKET_LENGTH
    //(otherwise adjacent data may get overwritten!)
    RFsetup(me,PACKET_LENGTH);
}
int main(void)
{
    init(); // initialize the system
```

```
        while(1)
        {
                //wait for a new message
                while(!RFRXdataReady());
                //receive the message
                RFreceive(count);
                for(char i=0;i<*count;i++)
                {
                        //blink the number of times stored within count
                        m_green(ON);
                        m_wait(50);
                        m_green(OFF);
                        m_wait(50);
                }
                //add one to the number of times to blink
                *count=*count+1;
                //send the updated count back to the other M2 (back to ping)
                RFtransmitUntil(count,you,50);
        }
}
```

F.7 – Wireless Communication Test Program (Base Station Commands)

// This program tested the M2's ability to receive commands from the basestation wirelessly.

```
#include "saast.h"
//#include "m_general.h"
#include "maevarm-rf.h"
#define PACKET_LENGTH 5

char message[PACKET_LENGTH] = {0,0,0,0,0}; //for this program count is an array of length 1
                                           //(just a single char)
char me[5] = {0xAC, 0xAC, 0xAC, 0xAC, 0xAC};
char you[5] = {0xCD, 0xCD, 0xCD, 0xCD, 0xCD};

void init(void)
{
        //place all init methods here
        m_init();
        m_green(OFF);
        m_red(OFF);

        //this method has this M2 "claim" the address 'me'. Every message this M2 interacts with
        //must be of length PACKET_LENGTH (otherwise adjacent data may get overwritten!)
        RFsetup(me,PACKET_LENGTH);
}
```

```
int main(void)
{
	init(); // initialize the system
	while(1)
	{
		while(!RFRXdataReady());
		RFreceive(message);
		switch(message[0])
		{
			case 0xA1:// This will eventually represent the Play command
				m_green(ON);
				m_red(OFF);
				break;
			case 0xA4:// This will eventually represent the Pause command
				m_green(OFF);
				m_red(OFF);
				break;
			case 0xA5:// This will eventually represent the Detangle command
				m_green(ON);
				m_red(ON);
				break;
			default:
				m_green(OFF);
				m_red(ON);
				break;
		}
	}
}
```

F.8 – Final Robot Program (Success Bot)

```
//Robockey Senior Design
//3-22-12

//This program allows the robot to receive commands play, pause, and detangle
//from the base station, and upon receiving the play command, find and retrieve the puck and
//score.
//This program is fully interrupt based.

#include "saast.h"
#include "m_general.h"
#include "maevarm-rf.h"
#include <avr/io.h>
#include <avr/interrupt.h>
#define PACKET_LENGTH 5

char message[PACKET_LENGTH] = {0,0,0,0,0}; //For this program count is an array of length
```

```
                                                    //1 (just a single char)
char outmessage[PACKET_LENGTH] = {0xA1,0,0,0,0};
char me[5] = {0xAD, 0xAD, 0xAD, 0xAD, 0xAD};
char you[5] = {0xAA, 0xAA, 0xAA, 0xAA, 0xAA};
volatile int lastpos[2]={300,200};
volatile int pos[2]={330,230};
volatile int dir[2]={-7,13};//Written as a vector to avoid using trig functions on the M2
volatile int posupdated;//Boolean that is only 1 when the position is first updated
int tolerance;
volatile uint16_t adc_results[16];
int channel;
int interruptsAllowed;
int lastCommand;

void init(void);
void init_ADC_inter(void);
void init_wireless_inter(void);
void find_puck(void);
void score(void);
void forward(int speed);
void backward(int speed);
void stop();
void turnleft(int speed,int radius,int fwd);
void turnright(int speed,int radius,int fwd);
void leftwheel(int speed, int fwd);
void rightwheel(int speed, int fwd);

void init(void)
{
        //Place all init methods here
        m_init();
        m_green(ON);
        m_wait(1000);
        m_green(OFF);
        m_wait(1000);
        lastCommand=4;
        //This method has this M2 "claim" the address 'me'.
        //Every message this M2 interacts with must be of length PACKET_LENGTH
        //(otherwise adjacent data may get overwritten!)
        RFsetup(me,PACKET_LENGTH);

        //Tolerance will be the change between left and right that we will
        //Consider to be close enough to equal to not matter
        tolerance=15;
}
```

```
void init_ADC_inter(void)
{
	//Set ADC prescaler to 128 - 125KHz sample rate @ 16MHz
	ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
	//Set ADC reference to AVCC
	ADMUX |= (1 << REFS0);
	//Left adjust ADC result to allow easy 8 bit reading
	ADMUX |= (1 << ADLAR);

	//Set ADC to Free-Running Mode
	ADCSRA |= (1 << ADATE);

	//Enable ADC
	ADCSRA |= (1 << ADEN);

	//Channel is an index for the array adc_results
	channel = 0;

	//Enable ADC Interrupt
	ADCSRA |= (1 << ADIE);
}

void init_wireless_inter(void)
{
	set(PCICR,PCIE0);//Enable pin-change interrupts
	set(PCMSK0,PCINT5);//Demask PCINT5
}

int main(void)
{
	init(); //General initialization of the system
	if(m_pwm_timer(1,3000)==0);
	if(m_pwm_output(1,1,ON)==0);//Pin B6, right wheel
	if(m_pwm_output(1,2,ON)==0);//Pin B7, left wheel


	init_ADC_inter();	//Initialize ADC complete interrupts (used for IR sensors)
	init_wireless_inter();  //Initialize wireless signal interrupts
	sei();   //Enable global interrupts

	ADCSRA |= (1 << ADSC);  //Start A2D Conversions

	int counter=0;
	int counter2=200;
	int init_forward=1;
	int init_goal=1;
```

```
int detangle=0;

while(1)
{
        if(lastCommand==1)  //Play command given
        {
                cli();    //Disable interrupts

                //The robot has detangled during play, but allow this to
                if(detangle==1)
                        detangle=0;
                //Counter allows robot to continue scoring for a small amount of time,
                //even if it loses the puck for a very slight moment.
                if(counter>0)
                {
                        m_green(ON);
                        counter--;
                        //If this is the first attempt, run pre-programmed route to goal
                        //Route is forward, turn left, forward, turn right (to face goal),
                        //then forward to score
                        if(init_goal)
                        {
                                turnleft(58,100,1);
                                m_wait(400);
                                turnright(70,6700,1);//This is STRAIGHT FORWARD
                                m_wait(1150);
                                turnright(58,100,1);
                                m_wait(810);
                                turnright(70,6700,1);//This is STRAIGHT FORWARD
                                m_wait(3000);
                                init_goal=0;
                        }
                        //If not first attempt and counter is triggered, score normally
                        else
                        {
                                score();
                                m_wait(10);    //waiting decreases issues with M2 crashing
                        }
                }
                //Counter has not been activated, therefore puck is not in possession
                else
                {
                        m_green(OFF);
                        //This allows the find_puck/score programming to start after
                        //counter2 gets to 0, even if the robot initially misses the puck
                        if(init_forward)
```

```
                {
                        turnright(70,6700,1);//this is STRAIGHT FORWARD
                        m_wait(10);
                        counter2--;
                        if(counter2==0)
                                init_forward=0;
                }
                else
                        find_puck();
        }
        //If puck is in possession
        if(m_gpio_in(E6))
        {
                counter=300;
                init_forward=0;
        }

        sei();  //Enable interrupts
}

if(lastCommand==4)  //Pause command given
{
        cli();  //Disable interrupts

        //This accounts for the situation where the robot scores a goal in the
        //initial attempt (no detangle has been given)
        //This is useful because it doesn't force the user to manually restart the
        //robot each time in order to perform that first attempt when the play
        //command is given.
        if(detangle==0)
        {
                counter=0;
                counter2=200;
                init_forward=1;
                init_goal=1;
        }
        stop(); //Stop the robot

        sei();  //Enable interrupts
}


if(lastCommand==5)  //Detangle command given
{
        //This disallows the robot from attempting the initial goal after detangling.
        detangle=1;
```

```
                    //Robot has a "clean slate" and will play from its normal programming
                    //(without initial goal attempt).
                    counter=0;
                    counter2=0;
                    init_forward=0;
                    init_goal=0;

                    //Move backwards slowly to "detangle".
                    backward(20);
                }
        }

}

ISR(ADC_vect) // Interrupt for ADC complete which is used to analyze the IR sensors
{
        //disable global interrupts so that all of this code will run
        //and will not be stopped by a different interrupt
        cli();

        // Store results of ADC
        // adc_results is an 8 bit array with the results of each ADC recorded.
        //adc_results => ADC0 ADC1 ADC2 ADC3 ADC4 ADC5 ADC6 ADC7
        adc_results[channel] = ADCH*100/255;

        // round answer to make sure it is now between 0 and 100
        if(adc_results[channel] > 100)
                adc_results[channel] = 100;
        if(adc_results[channel] < 0)
                adc_results[channel] = 0;



        // check which channel you just got results for and perform the correct function for that
        //ADC
        switch(channel)
        {
                case 0:
                        channel = 4;                    //Set channel to 4 so that results are stored in
                                                        //the correct space next time for ADC4
                        ADCSRB &= (0 << MUX5); //Restrict ADC to 0-7
                        ADMUX &= ~(0x0F);       //Reset MUX to XXXX0000 => ADC0
                        ADMUX |= (1 << MUX2);   //Set ADC MUX to 100 which corresponds
                                                        //to ADC4 (By putting a 1 in that location)
                        break;
```

Alex Butler
ENGR 4382 – Senior Design
4/24/2012

Spring Summary of Individual Contribution

I am a member of the Robockey senior design group, which has set up the foundation for an annual robot hockey competition by setting rules and regulations and creating a team of autonomous robots which competed in the Trinity Robockey games against a team fielded by faculty and staff.

The spring semester was devoted to testing the prototype robot which was constructed at the end of the fall semester, building the final robots for our team, and finalizing the programming for the robots as well as the rules and regulations for the competition. My first task was to order a solenoid for our kicking mechanism prototype. When the solenoid arrived I used it to construct a proof of concept prototype of the kicking mechanism and in testing it I decided that it should not be included in the final design because of poor performance and a lack of space on the final robot design.

I was in charge of editing the rules and regulations documents whenever the team decided a change was necessary. I wrote the P/POC assignment, which details exactly what the proof of concept would be, what it can do, and how it will be tested, and conducted the testing that it entailed. I documented the results of this testing in a short memo and also discussed the methods and results of this testing in our P/POC presentation.

Nathan and I did the majority of the soldering which was required for our project. We ordered several more of M2 processors and Wi-Fi chips this semester and Nathan and I split up the task of soldering connections to them. We also split up the task of soldering components to our printed circuit boards when the corrected versions arrived. There were 4 boards total so we each soldered 2 of them. When we figured out that the newest boards also had several mistakes, I helped Nathan to go through and make the necessary corrections. Once the printed circuit boards were correctly soldered and tested, they were attached to the robot chassis which Rachel constructed.

I took it upon myself to research rechargeable battery packs and order a set suitable for our needs. This eliminated the hassle of using several AAs and the need to replace drained batteries.

I also drafted the physical design of our final robot in a computer aided design program called Pro-Engineer. This allowed us to get accurate pictures of the robot from any angle, which was helpful in generating figures for presentations and reports.

I established a coordinate system for the rink by recording the coordinates the overhead camera and tracking algorithm gave when a tracking symbol was placed in several locations on the rink. I determined the scale by which the pixel coordinates the program gives to the actual location on the rink in inches.

My last task was to clean up the lab room where our rink was set up and return anything we borrowed from professors or staff.

For the Final report, I wrote the conclusions/recommendations section and included Pro-E drawings, the rules and regulations documents, and the P/POC testing results in appendices. For the Final Presentation, I talked about the constraints of our project, including the rules and regulations which the group defined.

To: Dr. Diana Glawe

From: Nathaniel Richison

Date: 4/24/2012

Subject: Summary of Spring Contribution

I was part of the Robockey team, whose goal was to set the framework for a robot hockey tournament. This tournament can be used as a learning device for Mechatronics classes, but before any future tournaments can be done a practice competition needed to take place first. For this reason a team of robot hockey players was constructed and programmed.

Since the scope of the project is large it was split into three sections: electronics, programming, and physical construction. I decided to focus on the electronic hardware, and only program enough to test the hardware. This is because I was in charge of most of the electronic prototyping and designing of the printed circuit boards (PCB).

The first major problem my group had is that the boards ordered over winter break were not correct. This meant that the boards had to be redesigned and reordered, but there was another complication that prevented that from happening. The working prototype board from earlier was draining a large amount of power. During testing of this board several solder joints were weakened causing extra failures in the system. Eventually I was able to find the source of the problem (an incorrect frequency used to control the motors), and solder a newer prototype board for the team to practice programming on. After this was completed I was able to go back to designing a new PCB with confidence in the schematic.

I made the printed circuit board design in a program called EAGLE and sent it to a company called BatchPCB. When the boards came in Alex and I soldered all of the components onto the board. There were still some errors with the board, but I was able to design some easy fixes to these new boards. Alex and I made these fixes to the board and attached them to the chassis Rachel made.

Finally I did some writing for the various reports and presentations. Majority of my writing was on sections dealing with the hardware I described above. However, I also handled all of the purchases of the group and created the storyboard for our final closing video.

I believe that my contribution was substantial, because the electronics did require significant time debugging, and was an intricate part of the project. Also I did not get lost in my niche and forget to help the group with the various reports, memos, presentations, and other assignments that popped up throughout the semester. Our project came out successful, and I think that anyone could easily recreate the hardware for the robots thanks to my documentation.

SPRING SUMMARY

TO: DR. GLAWE

FROM: JOHN KERR

SUBJECT: ENGR 4382: DESIGN VIII

DATE: 5/4/12

The purpose of this project is to create a team of autonomous robots which will be capable of playing hockey against a rival team of robots, and will win 70% of the games played. The purpose is also to implement a system of rules and regulations to be used in not only this robot hockey game, but also in future games played at Trinity.

One of my largest contributions to my design group has been to take on a large part of the programming of the microcontroller, known as the M2, which will control each of our robots. Most of my time programming has been spent on creating an improved version of the FaceBot program from last semester. This early version used two of the LTR-4206 infrared photo-transistors to detect the infrared light emitted by the LTE-4206 LEDs on the puck and turned to face it. I had to improve this to use the nine photo-transistors that are used in the final design (placed all along the perimeter of the robot) to detect the puck in 360 degrees. This new program was called FetchBot and also included programming which made the robot move towards and gather the puck in its puckbay.

After the puck was obtained, the robot would attempt to score (ScoreBot) using the location information for where it was on the rink, obtained from the camera and base station computer. This location is read by the camera, and then wirelessly sent from an M2 connected to the computer to the M2 that is connected to the robot. Although I did not program the robot to actually score, I did switch the wireless communication programming from a method called 'polling' to a more efficient programming method known as 'interrupts'.

Polling is an endless loop which continuously asks the microcontroller if the thing that it needs to do is ready to occur yet. Interrupts let the program know when what they need to happen will happen, and allows the microcontroller to be free for other tasks until this event occurs. Once this event occurs, the program will 'interrupt' the current process and perform whatever function is required at that time.

After FetchBot and ScoreBot were working separately, they had to be combined to create a final program which would be capable of both gathering the puck and moving it towards the goal to score. One other group member and I combined these programs. Once the robot was completed, I helped the group perform the competition against Dr. Nickels robot.

The final aspects of the project I was involved in were deliverables. I presented the conclusions in the final presentation. I wrote the majority of the design description

section of the final report (I didn't write the motor, controller board, puck detection, or power sections), as well as writing all of the photo-detector sections (including methods and results). I also am tasked with filming the end of semester video with the help of another group member.

To: Dr. Glawe, dglawe@trinity.edu
From: Matt Galla, mgalla@trinity.edu
Subject: Executive Summary

My group's senior design project was to design, construct, and program a team of autonomous robots that play robot hockey, or Robockey. In order to consider our project successful, our team of robots was required to beat a team of housebots fielded by Trinity faculty and staff with 70%. In addition, our group was required to establish a set of rules and regulation governing a Robockey tournament for use in future years.

My contribution to the project largely dealt with the programming aspect of the project. I wrote up documentation on how to set up a simple C program using the software AVR Studio that can be loaded into the M2 reprogrammable microchip via a USB through the program Flip 3.4.3. My documentation explains how to set up the project configurations to be compatible with the onboard ATMEGA32U4 device and where to find the library of custom functions used to access features on the chip. I also wrote several small programs of increasing complexity both to verify that components are working and to provide samples of how to program the M2.

The programs I wrote included a simple program to blink the on-board LED and an upgraded program that used the built-in timers to produce a PWM to a green LED controlled by a potentiometer. This program implemented the built-in ADC functionality of the M2 chip. I also wrote a program that tested the memory capabilities of the M2 microchip. This program had the M2 memorize one number and then instantiate and manipulate every index of a very large array, checking every time to ensure the manipulation was actually carried out by the M2. When every index of the array was sufficiently manipulated, the M2 was asked to recall the initial memorized variable. This program was run multiple times with increasingly larger arrays until the M2 was unable to correctly recall the initial memorized value. The next program used the USB communication capabilities of the M2 to print some numbers on the computer screen, using the program Kitty. The final testing program that was created purely to learn about and demonstrate a feature we needed later in the project was the wireless transmission capabilities of the rf communication module. This pair of programs were loaded onto two M2 microchips that passed an increasing variable back and forth, blinking the on-board LED the number of times equal to most recent value passed over the wireless transmission.

The final robot was broken into many smaller programs that were each created and verified separately. In particular, I created the framework for wheel control, wrote the non-interrupt version of FaceBot, which turns toward the puck, made the subprogram that receives and handles messages from the basestation, and worked a lot with the both the basestation program and the M2 to create the scoring system and logic. This task required extensive experimentation and interaction with the basestation program, the supplemental M2 program that interacted with the basestation, the robot wheels and power source, the robot glyph, the camera mount and focusing, the interior lights, and even the creation of a new relay program on a third M2 to pass messages received from the robot M2 over the wireless communication network to print on the computer screen via a USB connection for debugging purposes.

In the middle of the project I also worked extensively with the simulator program which gave an outlet to simulate experiments testing the effectiveness of different programming logic to drive the wheels of the robot based on the conditions of the playing field. The purpose of the simulator is to quickly determine whether or not a certain collection of logical calculations results in a behavior that successfully scores goals without the hindrances of imperfection in the data or relying on untested mechanical components of the robot. The simulator required several changes and additions (such as friction and a puck bay) before any scoring logic could be implemented that I also handled.

In addition to what is listed above I helped several group members with other tasks and in some cases, provided small bits of code that could be used to test the operations of other group members.

TRINITY UNIVERSITY SENIOR DESIGN

TO: DR. DIANE GLAWE

FROM: RACHEL ALLEY

SUBJECT: ROBOCKEY SUMMARY OF INDIVIDUAL CONTRIBUTION

DATE: 5/4/12

CC: DR. KEVIN NICKELS

The purpose of this project is to create a foundation for an annual Trinity robot hockey (Robockey) competition based upon a similar project undertaken at the University of Pennsylvania. A team of autonomous Robockey players have been designed, built, programmed, and tested. This team then competed against a team fielded by Trinity faculty and staff in a tournament consisting of 4 games.

This semester I have had several main focuses including: organizational and administrative tasks and chassis construction. Regarding my organizational and administrative tasks, I was responsible for updating the team's task lists. I was responsible for documenting all of the assigned weekly tasks and adding them to the basecamp project management system. I was also been responsible for organizing group meetings for presentations and compiling reports. Since April 2nd I have served a group leader and thus have completed all the responsibilities that accompany this position including: weekly team leader-advisor meetings, monthly management reviews, and making weekly meeting agendas as well as running the weekly meetings.

Regarding the chassis construction, I was responsible for acquiring all the necessary pieces and assembling them in order to provide a fully functioning robot. Completing this task entailed cutting out the Plexiglass pieces for the top and bottom layer of the robot. I then had to cut out a puck bay and slots for the wheels as well as drill all the holes needed to attach all the components to the frame. I made custom brackets in order to attach the motors to the frame and a custom bracket to attach the glyph to the robot. After all these pieces were prepared, I then assembled the frame and attached all the components including: two roller ball casters, a contact switch, two wheels and motors, a printed circuit board, 9 photodetectors with blinders, and a custom mount bracket for the glyph.

Besides these main tasks I have also worked on some smaller tasks. I was responsible for getting the lab setup after the move from Moody, specifically I had to mount the camera to the ceiling and run the appropriate cables to the base station computer. I created the project poster for our group. I along with Alex Butler finalized our rules and regulations, then I created a survey asking for students feedback regarding the clarity of our rules. I also did more research into alternative lighting for our rink and possible kicking mechanisms for our robot. Finally I was responsible for videoing the series of competitions, then along with John Kerr I made a compilation video of the footage for our final presentation. I, along with John Kerr, am responsible for filming and producing the video assignment.

```
case 4:
        channel = 6;                    //Set channel to 4 so that results are stored in
                                        //the correct space next time for ADC4
        ADCSRB &= (0 << MUX5); //Restrict ADC to 0-7
        ADMUX &= ~(0x0F);               //Reset MUX to XXXX0000 => ADC0
        ADMUX |= (1 << MUX1);
        ADMUX |= (1 << MUX2);           //Set ADC MUX to 110 which corresponds
                                        //to ADC6 (By putting a 1 in those two
                                        //locations)
        break;

case 6:
        channel = 8;                    //Set channel to 4 so that results are stored in
                                        //the correct space next time for ADC4
        ADCSRB |= (1 << MUX5);  //Restrict ADC to 8-13
        ADMUX &= ~(0x0F);               //Reset MUX to XXXX0000 => ADC8
                                        //(since it is in the 8-13 range)
        break;

case 8:
        channel = 9;                    //Set channel to 5 so that results are stored in
                                        //the correct space next time for ADC5
        ADCSRB |= (1 << MUX5);  //Restrict ADC to 8-13
        ADMUX &= ~(0x0F);               //Reset MUX to XXXX0000 => ADC8
                                        //(since it is in the 8-13 range)
        ADMUX |= (1 << MUX0);   //Set ADC MUX to 001 which corresponds
                                        //to ADC9 (By putting a 1 in that location)
        break;

case 9:
        channel = 7;                    //Set channel to 4 so that results are stored in
                                        //the correct space next time for ADC4
        ADCSRB &= (0 << MUX5); //Restrict ADC to 0-7
        ADMUX &= ~(0x0F);               //Reset MUX to XXXX0000 => ADC0
        ADMUX |= (1 << MUX1);
        ADMUX |= (1 << MUX2);
        ADMUX |= (1 << MUX3);   //Set ADC MUX to 111 which corresponds
                                        //to ADC7 (By putting a 1 in those three
                                        //locations)
        break;

case 7:
        channel = 10;
        ADCSRB |= (1 << MUX5);  //Restrict ADC to 8-13
        ADMUX &= ~(0x0F);               //Reset MUX to XXXX0000 => ADC8
```

```
                                        //(since it is in the 8-13 range)
                ADMUX |= (1 << MUX1);   //Set ADC MUX to 010 which corresponds
                                        //to ADC10 (By putting a 1 in that location)
                break;

        case 10:
                channel = 5;
                ADCSRB &= (0 << MUX5); //Restrict ADC to 0-7
                ADMUX &= ~(0x0F);       //Reset MUX to XXXX0000 => ADC0
                ADMUX |= (1 << MUX0);
                ADMUX |= (1 << MUX2);   //Set ADC MUX to 101 which corresponds
                                        //to ADC5 (By putting a 1 in those two
                                        //locations)
                break;

        case 5:
                channel = 1;
                ADCSRB &= (0 << MUX5); //Restrict ADC to 0-7
                ADMUX &= ~(0x0F);       //Reset MUX to XXXX0000 => ADC0
                ADMUX |= (1 << MUX0);   //Set ADC MUX to 001 which corresponds
                                        //to ADC1 (By putting a 1 in that location)

        case 1:
                channel = 0;            //Set channel to 0 so that results are stored in
                                        //the correct space next time for ADC0
                ADCSRB &= (0 << MUX5); //Restrict ADC to 0-7
                ADMUX &= ~(0x0F);       //Reset MUX to XXXX0000 => ADC0
                break;

        default:
                channel = 0;            //Set channel to 0 so that results are stored in
                                        //the correct space next time for ADC0
                ADCSRB &= (0 << MUX5); //Restrict ADC to 0-7
                ADMUX &= ~(0x0F);       //Reset MUX to XXXX0000 => ADC0
                break;

    }

    sei();//Re-enable global interrupts for future needs
}

ISR(PCINT0_vect) //Interrupt for receiving the wireless signal from the base station
{
    //Pin B5 is tied to the receiver. Therefore any time a valid packet is received, pin B5 will
    //go low.
    //This line makes sure that the change is from high-to-low rather than low-to-high.
```

```
        if(!check(PINB,5))
        {
                cli();//Disable global interrupts.

                RFreceive(message);
                switch(message[0])
                {
                        case 0xA1:      //Play
                                lastCommand=1;
                                break;
                        case 0xA2:      //Update position
                                if(posupdated==0)
                                {
                                        pos[0]=message[2]*256+message[1];
                                        pos[1]=message[4]*256+message[3];
                                        if(pos[0]!=lastpos[0])
                                                dir[0]=pos[0]-lastpos[0];
                                        if(pos[1]!=lastpos[1])
                                                dir[1]=pos[1]-lastpos[1];
                                        lastpos[0]=pos[0];
                                        lastpos[1]=pos[1];
                                        posupdated=1;
                                }
                                break;
                        case 0xA4:      //Pause
                                lastCommand=4;
                                break;
                        case 0xA5:      //Detangle
                                lastCommand=5;
                                break;
                        default:        //Default is pause
                                lastCommand=4;
                                break;
                }

                sei();//Re-enable global interrupts for future needs
        }
}

void find_puck(void)
{
        int long_wait=0, short_wait=0;

        //If no sensor detects puck
```

```
        if(100-adc_results[0]<45 && 100-adc_results[4]<45 && 100-adc_results[6]<45 &&
100-adc_results[7]<45 && 100-adc_results[10]<45 && 100-adc_results[8]<45 && 100-
adc_results[9]<45)
        {
                turnright(65,400,1); //Turn right at large radius for 20ms
                short_wait=1;
        }
        else
        {
                //If front sensor sees the puck
                if(100-adc_results[4]>67)
                {
                        turnright(70,6900,1);  //This is STRAIGHT FORWARD, for 60ms
                }
                //If front-left sees the puck more than front-right and the difference is greater than
                //tolerence
                else if(100-adc_results[6]>100-adc_results[0] && 100-adc_results[6]-(100-
adc_results[0])>tolerance)
                {
                        turnleft(17,0,1);       //Turn left slowly for 95ms
                        long_wait=1;
                }
                //If front-right sees the puck more than front-left and the difference is greater than
                //tolerence
                else if(100-adc_results[0]>100-adc_results[6] && 100-adc_results[0]-(100-
adc_results[6])>tolerance)
                {
                        turnright(17,0,1);      //Turn right slowly for 95ms
                        long_wait=1;
                }
                //If any of the back half sensors see the puck
                else if(100-adc_results[10]>70 || 100-adc_results[7]>70 || 100-adc_results[9]>70 ||
100-adc_results[8]>70)
                {
                        turnright(45,0,1);      //Turn right for 60ms
                }
                //If somehow it can't validate any other condition
                else
                {
                        turnright(70,6900,1);  //This is STRAIGHT FORWARD, for 60ms
                }
        }

        //Determine time spent executing chosen movement
        if(long_wait)
                m_wait(95);
```

```
        else if(short_wait)
                m_wait(20);
        else
                m_wait(60);
}

void score(void)
{
        if(posupdated==1)
        {
                //The following lines are used for debugging only
                //They command the wireless card on the robot to send a message to another M2
                //connected to the computer
                //via USB. This M2 simply displays the received message on the computer

                //outmessage[1]=dir[0]/256;
                //outmessage[2]=dir[0]%256;
                //outmessage[3]=dir[1]/256;
                //outmessage[4]=dir[1]%256;
                //RFtransmitUntil(outmessage,you,20);


                if(dir[0]<0)//The robot is facing the wrong end of the field
                {
                        if(dir[1]<0)    //The robot is facing more towards the right wall (facing the
                                        //incorrect goal)
                                turnright(60,100,1);
                        else            //The robot is facing more towards the left wall (facing the
                                        //incorrect goal)
                                turnleft(60,100,1);
                        //If either of the above cases occur, the robot carries out the above
                        //command for 0.8 seconds
                        m_wait(800);
                }
                else    //The robot is facing the correct end of the field
                {
                        //The green light turns on for debugging purposes
                        m_green(ON);


                        if(dir[1]<0)    //The robot is facing more towards the left wall (facing the
                                        //incorrect goal)
                        {
                                if(pos[1]<230)//The robot is closer to the left wall (facing the
                                                //incorrect goal)
                                        turnright(40,100,1);
```

```
        //This in turn causes the wheel to begin rotating
        if(m_pwm_duty(1,1,speed)==0);
                m_red(ON);
        //The following if/else statements determine the direction of the wheel which is
controlled
        //through the digital output pin D1.
        if(fwd==1)
                m_gpio_out(D1,ON);
        else
                m_gpio_out(D1,OFF);
}
void rightwheel(int speed, int fwd)
{
 //Counterpart of leftwheel()
        if(m_pwm_duty(1,2,speed/*80/100*/)==0);
                m_red(ON);
        if(fwd==1)
                m_gpio_out(D2,OFF);
        else
                m_gpio_out(D2,ON);
}
```

G Rules and Regulations

Rink

- The rink is a rounded rectangle approximately 240 cm by 120 cm in size (8' x 4')
- There are 50 cm wide goals on each side
- Robots must start outside of the 80 cm mid-rink section

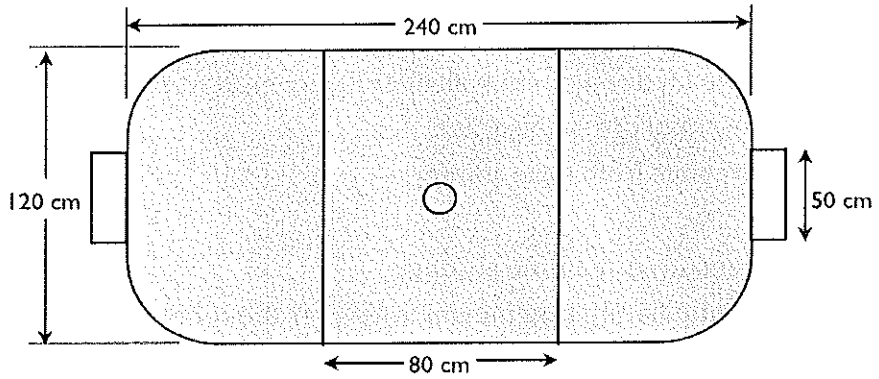


Figure 27: Overhead View of Rink

Rules

- Before the start of the game, a coin is tossed and the winning team chooses which side of the rink they want to start on
    - Teams keep their same side for the entire game
- A game consists of three 2 minute periods
    - There is a 30 second break between periods during which teams may interact with their robots
    - One overtime period if necessary
    - After one overtime period, the game goes into sudden death (First team to score a goal wins)
- Each period starts with the puck in the center and all robots on their respective sides behind a line 40 cm from the center of rink
- After a goal is scored, the robots and puck return to their starting positions
    - Timer is paused until game play resumes

Regulations

Every robot must meet the following requirements in order to compete:

- Robots must be no taller than 13 cm and remain within a 15 cm cylinder at all times
- Robots must be fully autonomous
- Robots must carry their own power source
- Robots must not fully constrain the puck's motion or capture the puck
    - At least half the puck's diameter must be exposed at all times
- Robots must not intentionally damage the rink, the puck, or any other robots
- Robots must not maliciously interfere with the wireless communication system
- Robots must not emit or intentionally reflect infrared light
- Each robot must have a tracking symbol
    - Must have a threaded rod or equivalent at its upper most point to which a 12 cm wide tracking symbol can be attached
- Must respond to commands from base station
    - Commands:
        - Play- Execute game time strategy
        - Stop- Immediately stop moving wherever the robot is
        - Detangle- Reverse direction at a random angle

Tournament Rules

- The Trinity Robockey tournament is single elimination
- Each team is randomly assigned a seed number
    - If there is an uneven number of teams, higher seeded teams may have a first round bye
        - Bye = the practice of allowing a player or team to advance to the next round of a playoff tournament without playing
- There is a five minute break between games for teams to pack up and the next teams to set up

H Pro-E Drawings

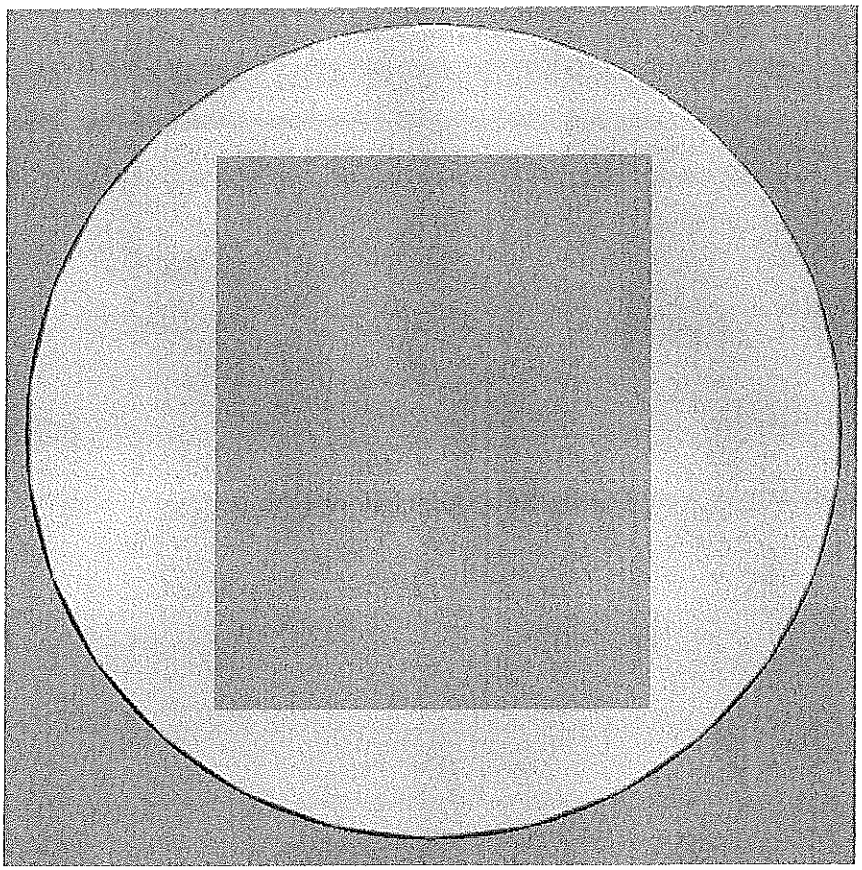


Figure 28: Top View of Robot

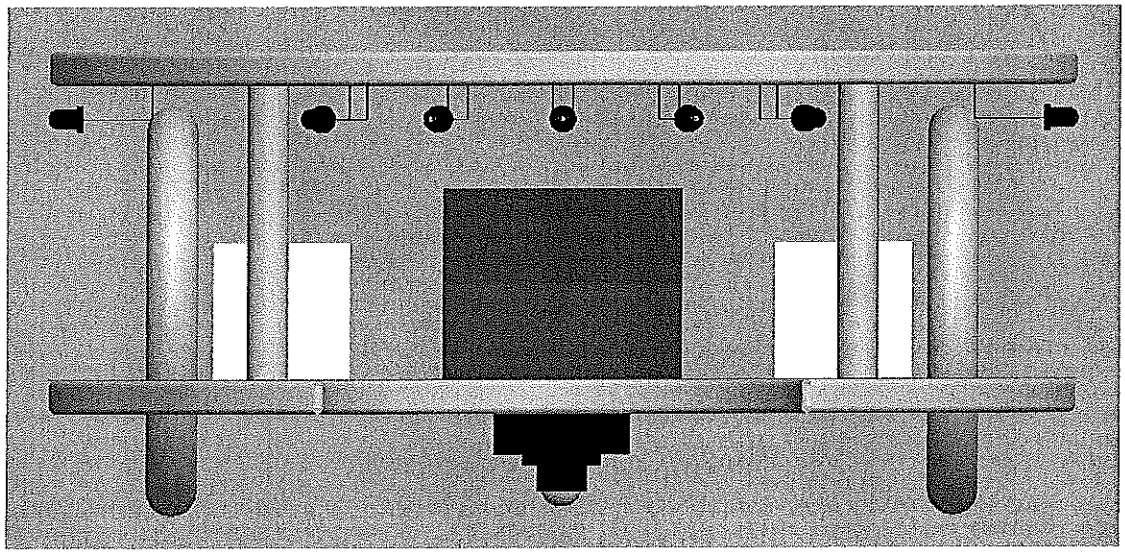


Figure 29: Front View of Robot

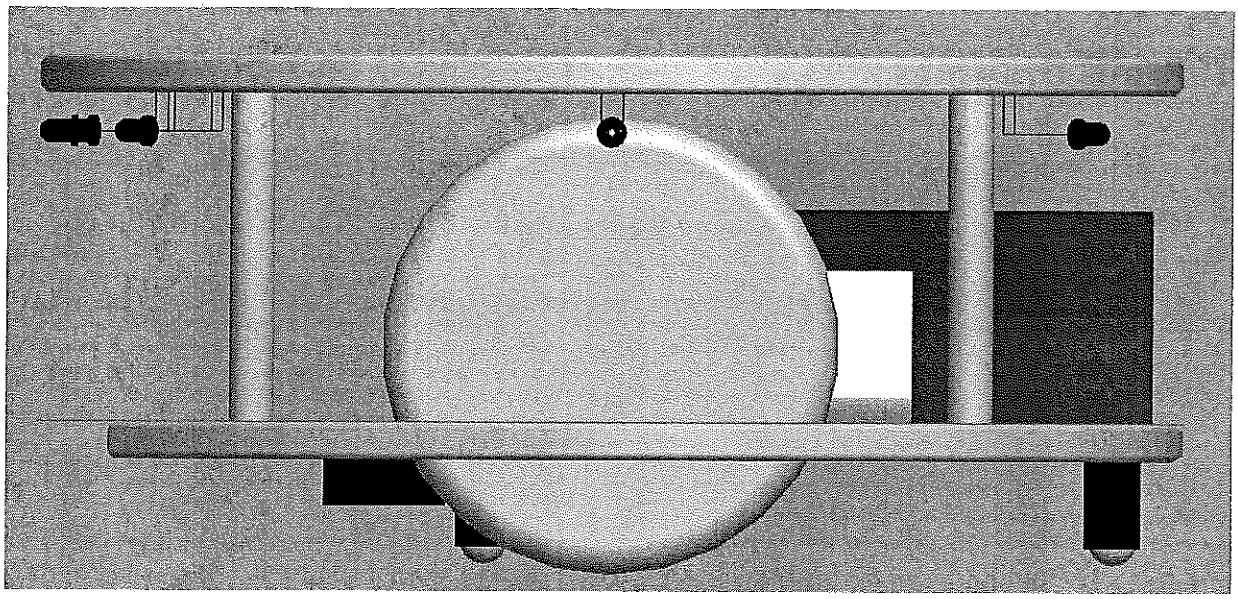


Figure 30: Side view of Robot

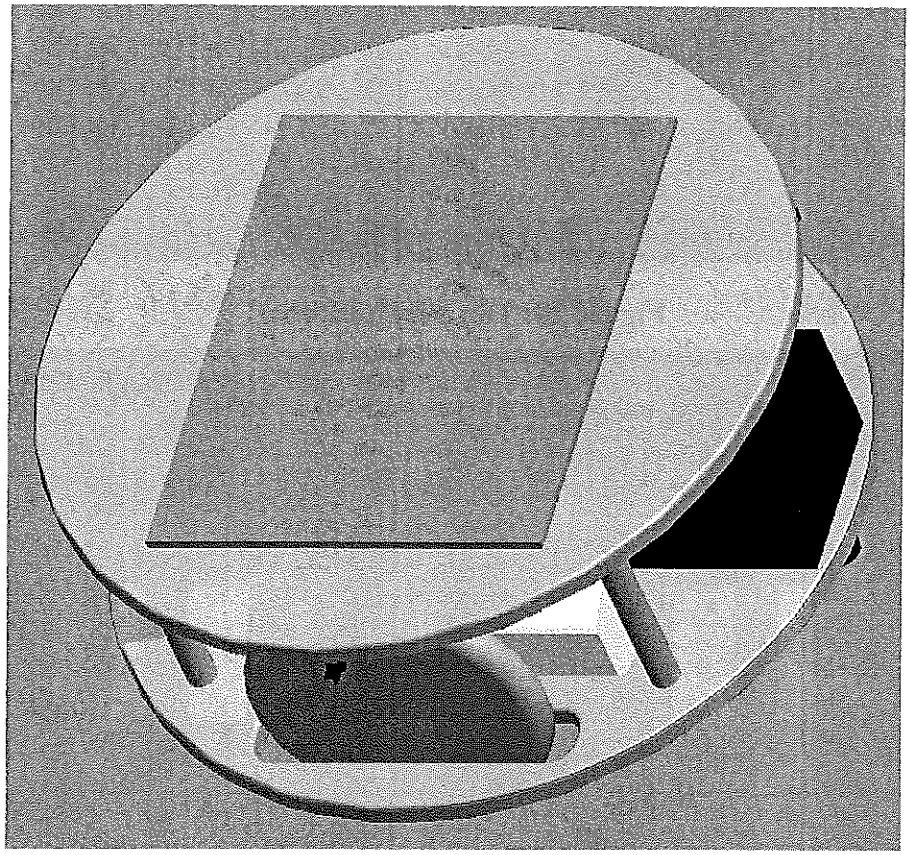


Figure 31: Isometric View of Robot

I Formal POC Testing

Motor Inspection Test

Both wheels spin when commanded to and spin in the correct direction when given a play and detangle command. Both wheels spin forward when given a play command, stop when given a pause command, and spin in the reverse direction when given a detangle command. However it seems that the left wheel is considerably more powerful than the right because the robot curves to the right when it should move forward. This is not a problem with the software as evidenced by the PWM test. It is most likely due to a faulty connection somewhere in the circuitry. The team is working to fix this problem.

PWM Monitoring Test

In this test oscilloscope probes were connected to the PWM output pins on the M2 (Pins B6 & B7). The PWM signals were monitored while the robot was given several commands from the base station M2. The robot is set up on blocks so it will remain stationary during the tests. The first command is a pause command, which tells both wheels to stop. Figure 32 shows that the PWM outputs for both wheels remain at close to 0 V with a small amount of noise.

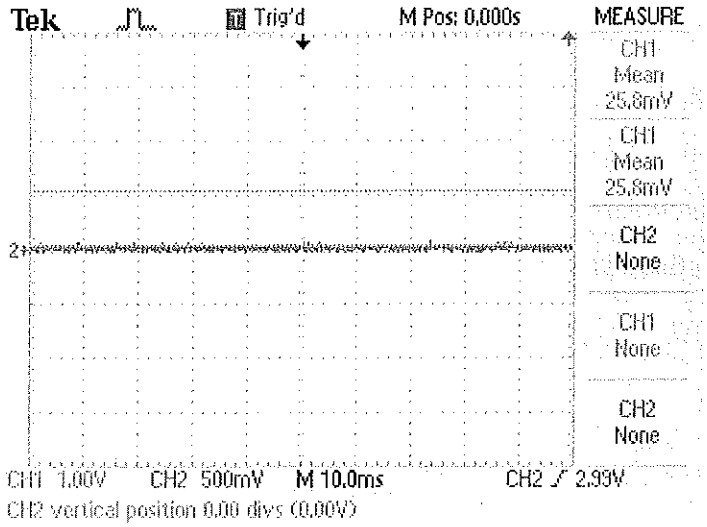


Figure 32: PWM outputs for both motors when stopped.

Next the robot is commanded to play. With the program currently loaded on the robot M2, this command tells both wheels to move forward at full power. Both PWM signals are set to 100%, meaning that the signal is at its highest point at all times. Figure 33 shows that when given a play command, both signals are at a constant 5V as expected, with some noise.

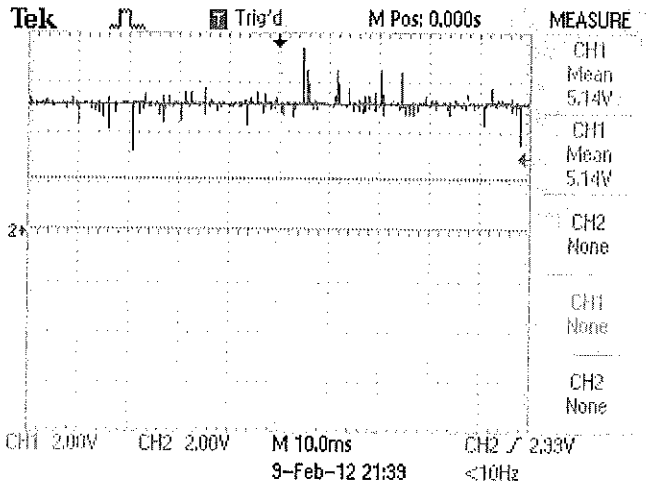


Figure 33: PWM outputs for both wheels with PWM is set to 100%.

In order to test the PWM outputs at a value other than 0 or 100%, the robot is given a detangle command, which currently tells both motors to go backward with a PWM of 55%. Figure 34 shows the output for the left and right wheel. Both signals are square waves which are high approximately half of the time and low the other half, as expected. Figure 35 shows the PWM for just the left wheel with cursors set up to show that the peak to peak amplitude of the square wave is indeed 5V, although there is a small DC offset of 200 mV.

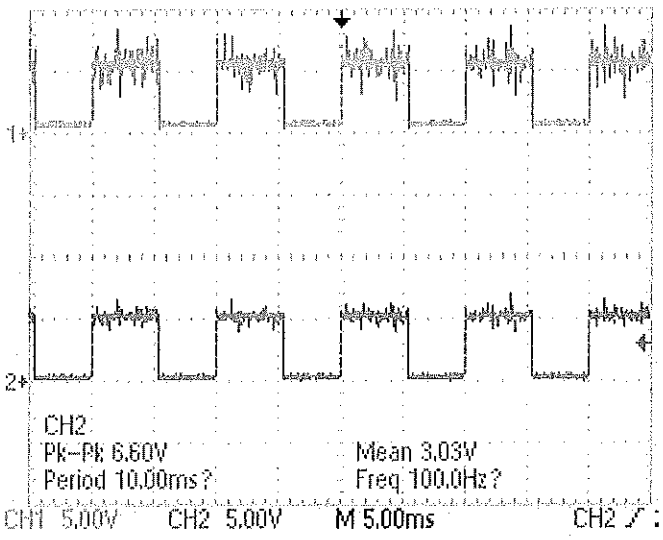


Figure 34: PWM outputs for both wheels when robot is commanded to detangle.

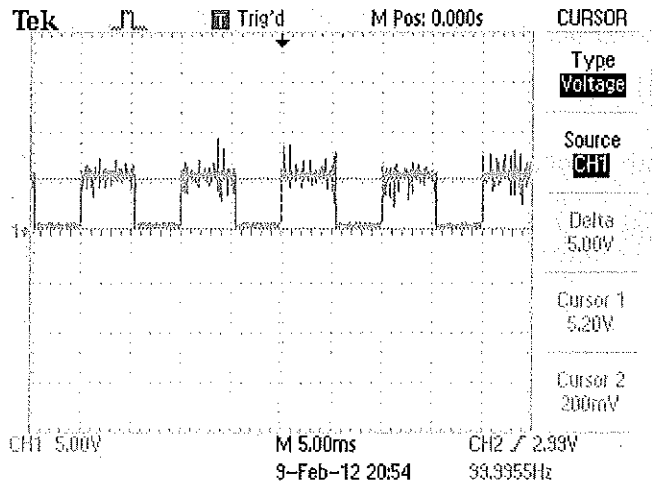


Figure 35: PWM output for the left wheel shows a square wave with duty cycle of 55% that goes between 200 mV and 5.2 V.

Battery Drain Test

In this test, a digital multi-meter is connected across the positive and negative terminals of the battery to connect the circuit. The positive probe is connected to the 10A input terminal to the DMM because it is expected that the current will be too high for the lower current terminal. When the robot is powered on, but the motors are stopped, the circuit draws 59 mA. With both motors running at 100%, the current being drawn from the batteries was measured as 285 mA.

According to the battery manufacturer, the set of 8 AA batteries being used has a capacity of 1800-2600mAh. From this the ideal battery life is estimated at 30.5-44 hours while stopped and 6.3-9.1 hours when the motors are spinning at full power.

Photo-Detector Range Test

The purpose of this test is to attempt to determine the range at which the robot can accurately see the puck. A DMM measures the analog voltage at one of the photo-detector inputs to the M2. The positive probe of the DMM is connected to a wire at the node going to Pin 11 of the M2 and the negative probe is connected to ground. A base reading is taken with the puck turned on to determine the voltage with just the overhead lights on. This voltage is measured as 3.15V. In this circuit a lower voltage means the photo-detector is receiving more IR light. The fact that this voltage is less than 5 V means that he overhead florescent lights are giving off some infrared light. A yardstick is used to measure the distance of the puck away from the robot. The puck is placed 3 inches away and turned on. Voltage measurements are taken as the puck is moved farther away from the robot in 3 inch increments, as seen in Table 5.

Table 5: Photo-Detector Voltage Measurements at three inch increments.

Distance (in)	Voltage (V)	Max -Measured (V)
3	0.013	3.137
6	0.153	2.997
9	0.155	2.995
12	0.170	2.980
15	0.186	2.964
18	0.204	2.946
21	0.208	2.942
24	0.255	2.895
27	0.249	2.901
30	0.350	2.800
33	0.648	2.502
36	0.825	2.325
39	1.270	1.880
42	1.630	1.520
45	1.760	1.390
48	1.990	1.160
51	2.040	1.110
*Voltage = 3.15V with puck off		

When the puck was relatively close to the robot, the voltage readings were fairly steady, but as the distance increased, the readings became much more volatile. The voltage would quickly increase seemingly without bound with the puck in a stationary position. This is why no further data is taken.

The third column in Table 5 shows the measured voltage subtracted from the reading taken with the puck turned off. These values give a better indication of the strength of the reading. Figure 36 shows a plot of this subtracted value as a function of distance. The data shows that the readings stay relatively constant up until the puck is about 30 inches away from the robot then start to fall off at a nearly linear rate as the distance is increased further.

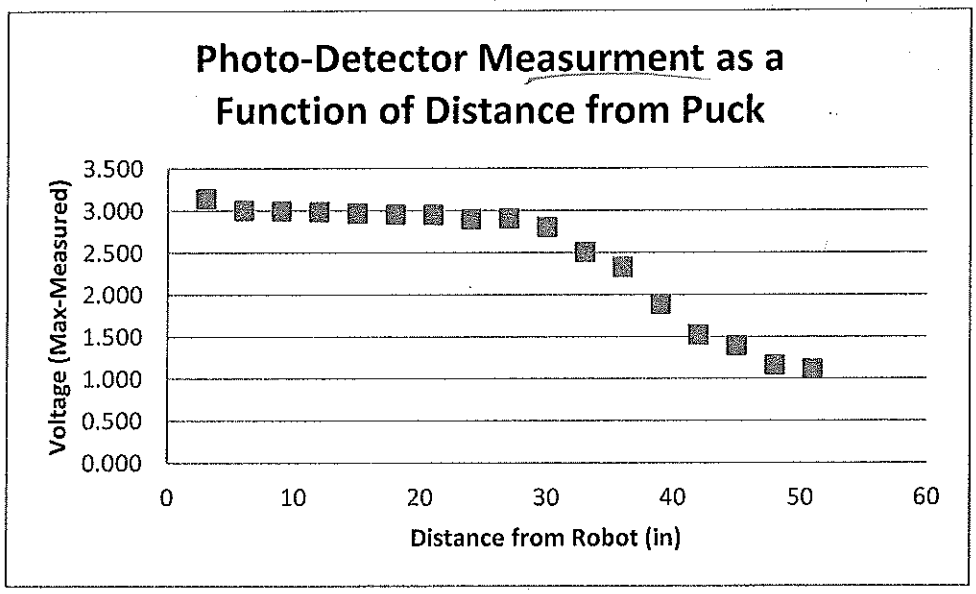


Figure 36: Data taken from Photo-Detector Range Test