Small Size Soccer Robots

A Major Qualifying Project Report submitted to the faculty of

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the degree of Bachelor of Science

Submitted by:

Spencer Belleville
Robotics Engineering

Conner Christensen
Mechanical Engineering

Ashley Espeland
Robotics Engineering & Computer Science

Logan Rinaldi
Robotics Engineering

Nathan Rogers
Robotics Engineering

Benjamin Schwantes
Electrical and Computer Engineering

Evan Vadeboncoeur
Robotics Engineering & Mechanical Engineering

Yifei Zhao
Robotics Engineering & Computer Science

May 3, 2023

Professor Siavash Farzan, Advisor
Robotics Engineering Department

Stephen Bitar, Co-Advisor
Electrical and Computer Engineering Department

Joshua Cuneo, Co-Advisor
Computer Science Department

Alireza Ebadi, Co-Advisor
Mechanical Engineering Department

ABSTRACT

The Small Size Soccer Robots MQP is an interdisciplinary first-year project that aims to design, fabricate, and test a multi-robot system for the international RoboCup Soccer League, targeting the Small Size League competitions. This project unites Robotics Engineering, Computer Science, Mechanical Engineering, and Electrical and Computer Engineering teams to develop a team of small autonomous robots adept at playing soccer with a golf ball. The Small Size League highlights intelligent multi-robot/agent collaboration and control within a dynamic environment, employing a hybrid centralized/distributed system. The project encompasses various tasks, such as designing, fabricating, and integrating the robot's structural and electromechanical components, including the chassis, ball control, and drive systems. The team also designs, assembles, and implements the robot's electrical circuits, featuring the processor, motor controllers, solenoids, and power distribution, while developing corresponding firmware for seamless integration. Additionally, the team crafts software to govern robot movement and execute strategic game tactics, ensuring a competitive performance in the RoboCup Small Size League.

ACKNOWLEDGEMENTS

We would like to thank several people who contributed to the success of this project. First, we'd like to thank our main project advisor, Professor Siavash Farzan, Robotics Engineering, for proposing the project and guiding us throughout the process. We would also like to thank our mechanical engineering advisor, Professor Alizera Ebadi, our electrical and computer engineering advisor, Professor Stephen Bitar, and our computer science advisor, Professor Joshua Cuneo. These advisors were there to support and offer guidance to the individual sub-teams and became a great resource for us. We'd also like to thank two companies who made our mechanical and electrical systems possible by offering to sponsor our project. Nanotech provided 30 drive motors and 7 dribbler motors. PCBWay provided the machined chipper prototype.

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

This MQP is the first year of a highly probable legacy project tasked with forming the groundworks of the WPI RoboCup Small Size League program. To expedite results and to add simplicity to organization, the team split up into three sub-teams: Mechanical Engineering, Electrical and Computer Engineering, and Computer Science. The goal of the project was to create a modular foundation of a prototype system that will be the basis for future testing, validation, and improvement. The prototype consists of a chassis skeleton constructed from custom fabricated aluminum components, a drive system based upon performance oriented brushless DC motors, and a ball control assembly with bespoke components printed using high-grade resin materials. All of which were designed and fabricated from scratch.

The mechanical engineering sub-team was mainly tasked with supplying the team with a functional prototype to use in their physical integration and testing stages. This prototype system would enable future project groups to have a solid foundation of experimental context for the system that would inform their further development and optimization of the system in preparation for international competition. The team found that the development of such a prototype was a task that necessitated a deep understanding of system requirements, previously successful design and fabrication strategies, and expertise in a variety of engineering applications such as designing for manufacturability and CNC milling.

The main goal of the Electrical and Computer Engineering sub-team was to power, control, and communicate with all the necessary components needed to play a match in the Small Size League. A secondary goal was to design and test reliable firmware and hardware that can be modular in the future for easy substitution as more data is collected and components need updating. The team approached these ambitious objectives by planning to design hardware and develop both firmware and software concurrently to maximize the short time spanned by the MQP project.

The main goal of the Computer Science sub-team was to develop and test a software system, built in a modular manner, that was capable of learning, strategizing, and motion planning completely autonomously. A secondary goal was to instantiate a simulated testing environment that included real-world physics. To accomplish this, the team chose to divide and conquer by breaking up into two smaller teams: Strategy and Navigation, that would both develop and test their results in a nearly parallel way.

To test that the project objectives were satisfied, the prototype underwent: a stress analysis in simulation, electromechanical iterations, firmware fine tuning, and strategizing in a simulated environment. The tests verified that the prototype was functioning properly and fully autonomous with a master to agent relationship. At the end of this project's first year, the team was able to create a robot in a modular setup as well as a developed codebase with the necessary class structure to set the foundation of a highly potential WPI RoboCup Small Size League program.

CHAPTER 1 INTRODUCTION

RoboCup was founded with the goal of advancing the state of the art of autonomous intelligent robots. The first official RoboCup soccer games were played in 1997, with over 40 teams and 5,000 spectators in attendance. Since its strong start, the league has grown into an international event where top teams of six autonomous robots compete against others in their size class. This academic year, AY2022-23, is the inaugural year of WPI's RoboCup Small Size League (SSL) Soccer Robotics team. The team of Robotics Engineering, Electrical and Computer Engineering, Computer Science, and Mechanical Engineering students developed a prototype autonomous soccer robot as part of their Major Qualifying Project (MQP), laying the groundwork for future MQP teams to continue these efforts in hopes of competing on the international RoboCup stage. The goal for the first year of the program is to design, build, and test a prototype robot for demonstration during Project Presentation Day in April.

The MQP group was first divided into sub-teams, Mechanical Engineering, Electrical and Computer Engineering, and Computer Science, that were to be responsible for certain aspects of the overall system. The Mechanical Engineering (ME) sub-team is responsible for the development of the physical structure and assemblies within the robot including chassis, drive system, and ball control. The Electrical and Computer Engineering (ECE) sub-team is responsible for the development of the electronic components of the robot including the PCBs and their associated software and firmware. The Computer Science (CS) sub-team is responsible for the programming of the robots themselves as well as the overall strategy system. This includes motion planning, game state analysis, and strategy.

We split our team into three sub-teams: mechanical engineering, electrical and computer engineering, and computer science. Sub-teams conducted research of Team Description Papers (TDPs) of previously successful programs to build a fundamental knowledge base of designs, strategies, and solutions to common issues. From this review, sub-teams decided on the best options to pursue for this project and set the development of the system in motion.

The ME sub-team determined the chief objectives of the project to be:

1. Design or identify suitable solutions for:
    a. Chassis- base plates, standoffs, mounting interfaces
    b. Drive System- omni-wheels, drive motors, motor mounts
    c. Ball Control- solenoids, dribbler motor, dribbler, kicker, chipper, ball control superstructure
2. Fabricate or otherwise acquire the above components.
3. Assemble above systems and integrate with ECE components.
4. Test prototype and refine design.

By using the manufacturing facilities in Washburn shops, Higgins Rapid Prototyping Laboratory, and Innovation Studio's Makerspace in conjunction with resources such as SolidWorks 3D CAD and Esprit CAM software at our disposal as WPI students and securing partnerships and sponsorships with companies and organizations such as PCB, the ME sub-team made progress towards fulfilling these objectives.

The ECE sub-team identified the following objectives:

1. Design and implement circuits capable of:
    a. Powering circuits and electrical hardware
    b. Operating robot hardware
    c. Communicating with master controller
2. Develop firmware capable of interpreting and executing tasks from master controller

The CS sub-team determined the main objectives to be:

1. Install and be able to use a simulation program for testing purposes
2. Determine and use a path planning algorithm to traverse the field
3. Develop a game-play strategy, capable of:
    a. Analyzing the game state of the board
    b. Assigning roles to robots
    c. Creating plays and be able to select from them
    d. Calculating the probability of successful shots and passes
4. Support ECE with firmware development and communication protocols

With a large team, split into sub-teams, collaboration can sometimes be challenging. This team excelled at collaboration and made the necessary integrations of the physical robot, the electromechanical system, and the system architecture. The integration of each sub-team's developments enabled the combined testing of each sub-team's contributions. Whether it was testing that the circuit boards could power and control the mechanical system or that the on-board communication protocol could receive the transmission from the software architecture, or even that the coded robot kinematics matched with the manufactured robot. All together the team designed, tested, and validated in unison while developing in parallel.

The following chapters will outline our complete background research, design process, testing, results, and conclusions that capture the overall progress of this project.

CHAPTER 2 BACKGROUND

Each sub-team read a multitude of Team Description Papers (TDPs) and existing solutions available by SSL teams over a wide range of years. From these TDPs, as well as some supplementary material, we were able to come up with a starting point for our designs. The following sections cover our research in each aspect of the project.

2.1 MECHANICAL BACKGROUND

The Mechanical sub-team was tasked with providing the physical platform with which the ECE and CS sub-teams were to use to accomplish the common tasks within the game, including moving, turning, shooting, passing, and dribbling. The sections that follow introduce the fundamental mechanical subsystems that the sub-team identified as essential and reports the decision-making process for design and component related matters related to each subsystem.

2.1.1 CHASSIS

The design of the chassis of a Small Sized Soccer Robot is integral to its success. The chassis provides a solid and reliable base from which the rest of the robot can perform. Without a well designed and constructed chassis, the robot can hit many roadblocks and problems that will halt its success within the competition. In the RoboCup rules, there are aspects of the chassis or size of the robot that fall into pre-made design constraints, one example of this is the robot’s size. The dimensions of the robots allowed to compete are already set out for us but what is left for us to decide is things such as material and construction among other things. After looking at many previous competing robots, we have narrowed it down to a general shape and look but we can achieve that look different in many ways.

2.1.1.1 CHASSIS MATERIAL

The design of the chassis of a Small Sized Soccer Robot is integral to its success. The chassis provides a solid and reliable base from which the rest of the robot can perform. Without a well designed and constructed chassis, the robot can hit many roadblocks and problems that will halt its success within the competition. In the RoboCup rules, there are aspects of the chassis or size of the robot that fall into pre-made design constraints, one example of this is the robot’s size. The dimensions of the robots allowed to compete are already set out for us but what is left for us to decide is things such as material and construction among other things. After looking at many previous competing robots, we have narrowed it down to a general shape and look but we can achieve that look different in many ways.

2.1.1.1.1 PLASTIC

Plastic is the first option we looked at for the chassis of our robot. Plastic is a very interesting material and offers quite a few benefits. Such things as its extremely high manufacturability and low weight were very appealing in the decision process. Along with these two attributes, there was also the very low predicted cost which was also appealing to the team. Although these three criteria were happily met, they were weighed out by the cons. These cons were mainly the extremely poor durability of 3D printed plastic which could not withstand any sort of contact or heavy weight the robot would experience. The second con was its poor recoverability. The problem of not easily switching out any critically broken plastic would cause any success of the robot to falter.

2.1.1.1.2 STEEL

The second material we looked at was steel. Steel was almost the exact opposite of plastic in the way it met the defined criteria. While the durability of the plastic was low, steel was the exact opposite and has the highest durability of all the materials looked at. Along with high durability, it also has a relatively low cost and much better recoverability. Unfortunately, because of its high durability, it does suffer in manufacturability, which is important for us as we are a first-year team with little experience and equipment available. This lack of manufacturability along with low adaptability were the factors that drove us to our decision.

2.1.1.1.3 ALUMINUM

The final material we investigated is aluminum which is also the material we have decided to go with. There are many reasons for this decision, but the biggest factors were its very high durability which will allow the robot to be heavier and survive any contact made, as well as its manufacturability. The combination of these two criteria is a very big reason for our team's decision because we can now make a robot that is durable but also can be made with the tools that are available to us.

2.1.1.1.4 CHOOSING A MATERIAL

With our criteria met we were happy to see our literature review aligned with our choice. Aluminum was a recurring choice with most RoboCup teams like the University of Adelaide for example (UAdelaide, 2005). Having chosen a material, we then moved on to the next aspects of our robot chassis we needed to choose.

TABLE 2.1 CHASSIS MATERIAL DECISION MATRIX

Criteria	Weight	Plastic		Aluminum		Steel	
	1-10	Score	Weighted	Score	Weighted	Score	Weighted
Cost	7	9	63	9	63	8	56
Weight	8	8	64	7	56	5	40
Manufacturability	7	9	63	8	56	6	42
Adaptability	5	7	35	7	35	6	30
Durability	9	2	18	9	81	10	90
Recoverability	6	2	12	8	48	7	42
Total Score			255		325		300

2.1.1.2 CHASSIS LAYERS

All robot designs in TDPs change from year to year and look a little different as you look from team to team. With the material of the chassis chosen we now had to evaluate how we wanted our chassis to look. From our literature review, we found three options that have been used over the years which were different ways to stack the components of your robot. These included having the robots consist of one layer with all components on one baseplate, two layers that would allow a place for the PCB and other electronics to rest, or three layers that would give the robot plenty of space for everything. After creating the decision matrix in Table 2.1, we decided to go with the double-layer design.

2.1.1.2.1 SINGLE BASE

The single base plate layer design was immediately seen as the most complicated of the three. The intention of the design is to have a singular base plate that will encompass all the internals of the robot. This would include the drive system and ball control mechanisms along with all electronics required to power and control the robot. The benefits of this design are its low cost and weight due to it being a single layer. These, however, are the only benefits of this design. The negatives greatly outweigh the positives such as terrible recoverability since if something breaks in the chassis it can't be replaced since it is all one unit. Additionally, it is a huge design challenge for us as a first-year team to make a single-layer design that can house everything that needs to be within the robot.

This design has been used many times in the past by multiple different teams. Below is a figure from an old 2014 SKUBA design which was a single-layer robot (Sukvichai & Panyapiang, 2014). As seen in the photo they have found a way to fit everything they need for the robot within a single layer of the robot. This design is quite old as they have changed their chassis as time has gone on.

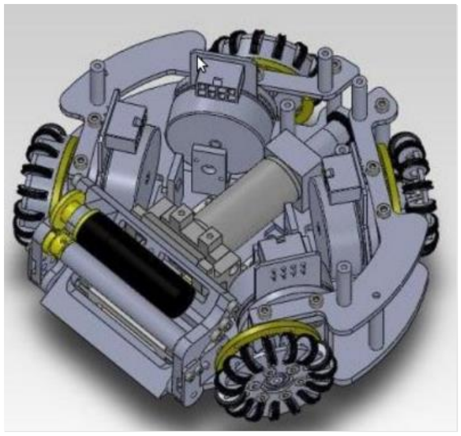


FIGURE 2.1 3D MODEL OF 2014 SKUBA ROBOT

2.1.1.2.2 TRIPLE LAYER

The next design is the triple-layer design of the chassis. The triple-layer design was found to be the least desirable option for the chassis after using the decision matrix and looking at examples of its use. The triple-layer design offers lots of space for all the internals of the robot but that seems to be the only positive. It is much heavier than the other two designs and lacks the other two's stability. For these reasons we have decided that the positives of allowing us to have more freedom of where to place things in the robot are heavily outweighed by the overall complications it would cause.

The triple-layer design, however, has been used in the past. Below is a figure from the Georgia Tech RoboJackets team in the year 2007 which is the first TDP of theirs we have available (Bardagjy et al., 2007). While their first design was triple-layered, the next year they changed their design to a more compact double-layer design.

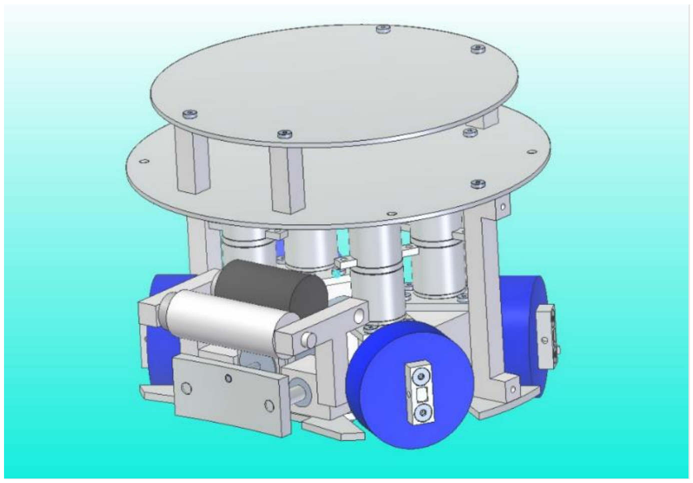


FIGURE 2.2 3D MODEL OF 2007 ROBOJACKETS DESIGN

2.1.1.2.3 DOUBLE LAYER

The final layer design we looked at is the double-layer design. This design was quickly agreed upon as the best design for our first-year robot. From the decision matrix, you can see it has a relatively high value for all the criteria with an emphasis on stability and weight. These two criteria, however, were not the deciding factors. The reason we chose this over the other two was that it offered the most stability with relatively easy manufacturability along with enough room for us to have all the mechatronics housed on the first layer of the robot with a given space on the second layer for the PCB and any other electronics.

The double-layer design is a widely used design with multiple different RoboCup teams including the two figures below. Figure 2.3 shows the RoboJackets change from their first-year triple-layer design to their later double-layer design (Bardagjy et al., 2007), (Bardagjy et al., 2008). Figure 2.4 shows the 2014 CMDragons (Biswas et al., 2014) two-layer design they used. With these two examples and many other team examples, we plan to use these two-layer designs which give one space for mechatronics and another for the PCB.

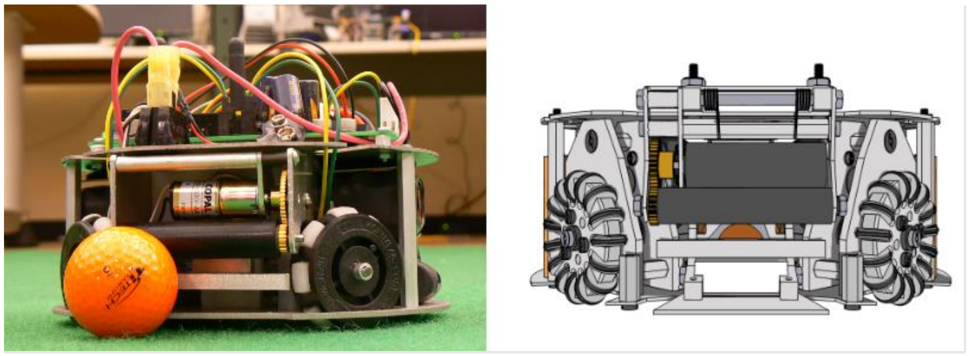


FIGURE 2.3 GEORGIA TECH ROBOJACKETS 2007 ROBOT (LEFT) AND 2008 ROBOT (RIGHT) 3D MODEL

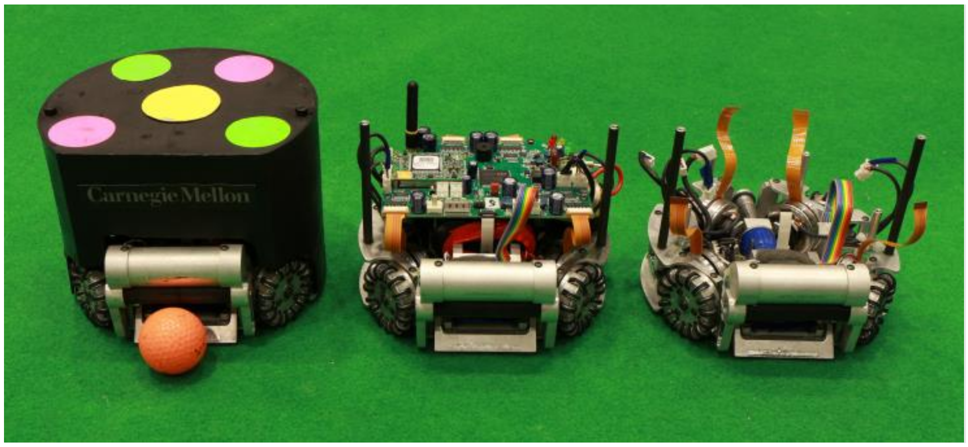


FIGURE 2.4 2014 CMDRAGONS DOUBLE LAYER ROBOT

TABLE 2.2 CHASSIS LAYERS DECISION MATRIX

Criteria	Weight	Base Plate		Double Plate		Triple Plate	
	1-10	Score	Weighted	Score	Weighted	Score	Weighted
Recoverability	6	4	24	8	48	7	42
Stability	6	7	42	9	54	6	36
Manufacturability	8	6	48	6	48	5	40
Cost	7	8	56	7	49	6	42
Weight	8	8	56	7	56	6	48
Total Score			234		255		208

2.1.1.3 CONNECTION POINTS

The final physical aspect of the chassis design is the connection points of the two layers of the robot. Since we have decided upon using a double-layer design for our robot, we must now find a way to functionally connect these two plates. After extensive literary research, we have come up with two options to connect these two plates. The option consists of standoffs and plates. After creating the decision matrix below we decided to go with the standoffs.

2.1.1.3.1 PLATES

The first connection design we looked at was connection plates. Connection plates are specially designed and manufactured plates that are connected to the two plates with nuts and screws to create a strong connection. These plates offer the best stability between layers and are extremely durable. These plates also offer a good amount of recoverability as they can be easily changed out, but their poor manufacturability makes it hard to have replacements on hand. Another big problem with these plates is the weight they will add to the robot which is unnecessary to our design.

The plate connection design has been used before as shown in the figure below (UAdelaide, 2005). From what you can see, the plates offer a solid connection point that can be easily replaced if need be. Also, from the image, you can see an exact "black box" space where all the internals will fit.

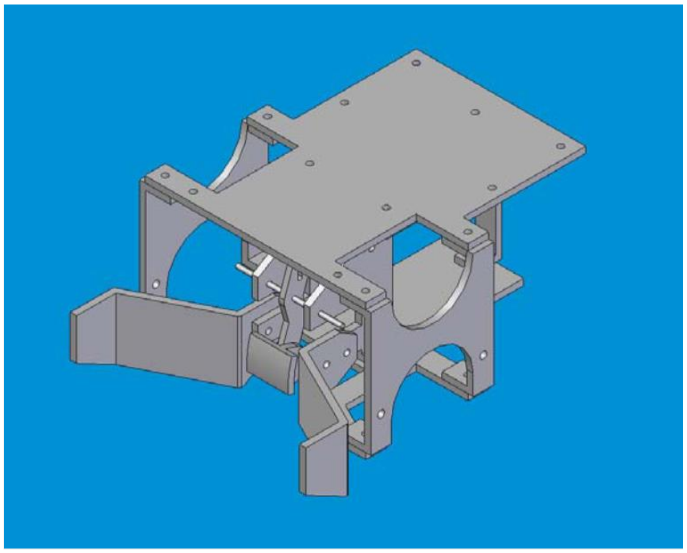


FIGURE 2.5 ADELAIDE PLATED CHASSIS DESIGN

2.1.1.3.2 STANDOFFS

The second option for connection points was metal standoffs. These standoffs would be female to male standoffs which would be secured by nuts and bolts. These standoffs are a great alternative to the plates as they offer relatively the same support and stability but cut down on the overall weight considerably. Along with less weight, they are also cheaper and are readily available to purchase whereas we would need to manufacture custom plates. These standoffs also take up less room on the robot itself which leaves more room for the internals to be worked in.

These standoffs have been researched and are highly spoken about by other RoboCup teams. Below is an image from the CMDragons team which shows one of their robots without a cover and standoffs can be seen on the bot (CMDragons, 2014).

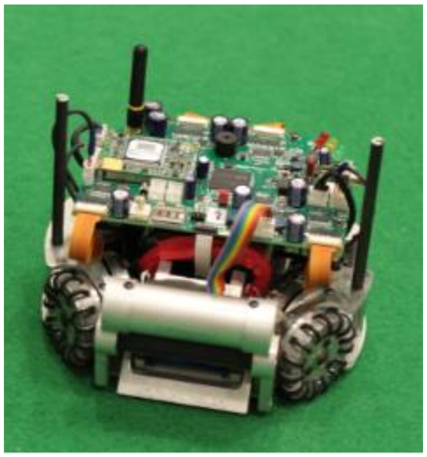


FIGURE 2.6 CMDRAGONS CHASSIS STANDOFFS

TABLE 2.3 CHASSIS CONNECTION POINTS DECISION MATRIX

Criteria	Weight	Standoffs		Plates	
	1-10	Score	Weight	Score	Weight

Recoverability	5	9	45	8	40
Manufacturability	6	8	48	6	36
Stability	9	8	72	9	81
Cost	7	9	63	7	49
Weight	8	8	64	6	48
Durability	8	6	48	9	72
Total Score			340		255

2.1.2 Ball Control

In the small size soccer robot league, the ball control is reliant on complex mechanisms in small packaging. There are three ways the robot interacts with the ball. The kicker mechanism is used for direct kicks along the ground using a punching action. The kicker is often employed for passes, free kicks, and shots on goal. The chipper mechanism is used for parabolic kicks of the ball. The chipper is advantageous to have for kicking the ball over a blocking opponent robot or out of a crowded part of the field. The final mechanism is the dribbler which uses a roller to contact the ball while the robot is in motion. The dribbler is used to maintain contact and control of the ball while the robot is in motion on the field. The league functions off iteration so as a team just starting, we are relying on existing and former teams' TDPs.

While this means the selection process could have included countless designs and possible decision matrices it is important to note our limitations. We are a first-year team and have no first-hand experience with these robots or designs. Additionally, our ability to manufacture components ourselves is limited by our facility's capabilities. Finally, our budget limits what vendors are possible for stock or outsource manufacturing.

To alleviate some of these limitations we will be testing and experimenting with ball-controlling mechanisms designs. This will be important for us to get an understanding of how the ball and robot interact physically in front of us. We plan to start with 3D-printed prototypes of each component and will then work to replace that with newly machined components. It may be shown that the 3D print will be a sufficient material choice, but we will not know until later. Changes in the chosen material and design may then change to reflect any new information we learn along the way.

2.1.2.1 KICKER

There are two main kicker designs teams use, an angle kicker and a straight kicker. When deciding between kickers there were two main criteria manufacturability and adaptability. As a first-year team, we are looking to limit the amount of added complexity of each mechanism. The thought process behind it is as a first-year team with limited manufacturing capabilities we want to be able to produce a functional robot. We also are prioritizing adaptability for a similar reason. Due to our team's limited first-hand experience, we are making the most educated decisions we can, but we want room to improve our designs if necessary.

2.1.2.1.1 ANGLE KICKER

An angle kicker rotates along a central axis changing the distribution angle of the ball. The advantage of this is that a robot can distribute the ball at multiple angles without moving the physical robot. For this to be possible the front of the robot needs to be open or have a wide slot so the kicker can swing left to right to change the distribution point on the face of the robot. Out of the TDPs evaluated only one team had a confirmed angle kicker. Aside from the added complexity to the actual kicker, the other mechanisms of the dribbler and chipper would also need modified designs to work with the moving kicker. It also means that rather than trying to perfect one kick, multiple kicks at different angles would need to be programmed and simulated.

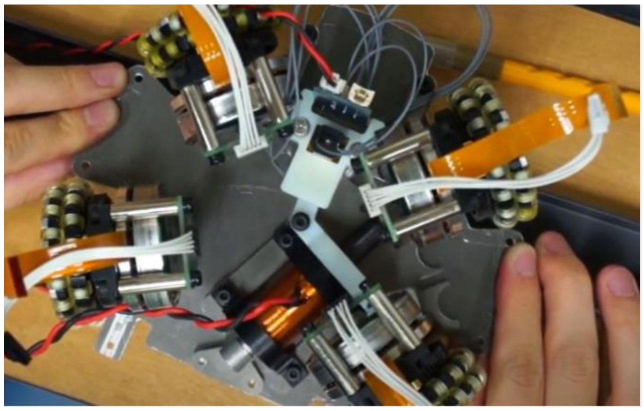


FIGURE 2.7 MULTI-ANGLE KICKER FROM OP-AMP 2017

2.1.2.1.2 STRAIGHT KICKER

The other design is that of a straight kicker which is mounted to hit the ball in the center of the face of the robot. One of the largest factors that led to choosing the straight kicker was the simplicity of both the design and manufacture. The straight kicker being mounted in one position allows there to be more room surrounding the mechanism. This will help with the adaptability of the robot. The saved space can then be used by the ECE sub-team or allow for design revisions later down the line. The straight kicker also means that each of the ball-controlling mechanisms is not as constrained by each other's design as with an angle kicker. For these reasons, we chose the straight kicker based on the criteria in a decision matrix found in Table 2.4.

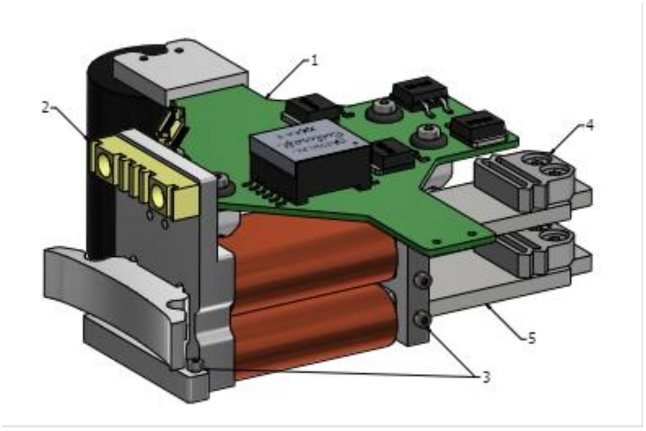


FIGURE 2.8 STRAIGHT KICKER TIGERS 2020

TABLE 2.4 KICKER DESIGN DECISION MATRIX

Criteria	Weight	Multi-Angle Kick (OP-AmP)		Straight Kick (TIGERs, KIKS, SRC, CMDragons)	
	1-5	Score	Weighted	Score	Weighted
Cost	3	2	6	3	9
Weight	3	2	6	4	12
Manufacturability	5	2	10	5	25
Adaptability	4	5	20	2	8
Fixability	3	3	9	3	9
Popularity	2	3	6	4	8
Total Score			57		71

2.1.2.1.3 PLUNGER

An additional decision matrix was created for the shape of the plunger used in the kicker. There are two commonly seen types of plungers, parabolic, and flat plate. Our robots are using the flat plate plunger. We chose this one because it would likely be easier to manufacture than a parabolic plate. Additionally, while some teams have experienced success using a parabolic plunger others noted that it led to inconsistencies in the kicks. This same thought process carried through to the fixability criteria. Since the flat plunger is easier to manufacture it should be easier for us to either fix or replace that component should anything happen while in use. The plunger will likely be made from aluminum, but further evaluation will be carried out.

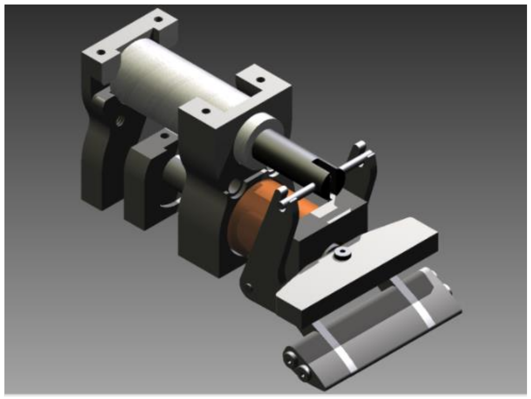


FIGURE 2.9 FLAT PLATE KICKER ROBOJACKETS 2013

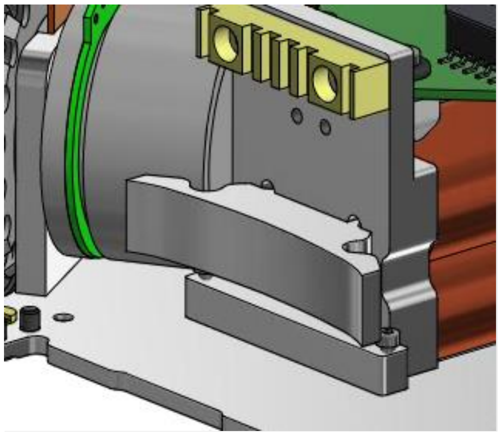


FIGURE 2.10 PARABOLIC PLATE KICKER TIGERS 2020

TABLE 2.5 KICKER PLUNGER DECISION MATRIX

Criteria	Weight	Parabolic Plate (RoboJackets, TIGERS, RoboDragons, SRC, ITAndroids, SKUBA)		Flat Plate (RoboJackets, RoboTeam Twente, Immortals, OP-AmP, CMDragon)	
	1-5	Score	Weighted	Score	Weighted
Cost	3	3	9	3	15
Weight	3	3	9	4	15
Manufacturability	5	3	15	5	25
Adaptability	4	3	12	2	12
Fixability	3	4	12	3	15
Popularity	2	5	10	4	10
Total Score			67		92

2.1.2.2 DRIBBLER

The dribbler is crucial to the robot's ability to control the ball throughout the game. The dribbler allows the robot to have possession of the ball between passing or kicking the ball. It consists of a roller that maintains contact with the ball while the robot is in motion. Importantly the material of the roller needs to have high enough friction with the ball that there is no slippage. To design a dribbler one of the first decisions to be made is what kind of roller will be used. For this, a decision matrix was used shown in Table 2.6. There are a variety of shapes a roller design can have, however, due to the limitations of both machining and experience we chose to focus our efforts on understanding the more common designs from TDPs. An evaluation of five different roller shapes was carried out.

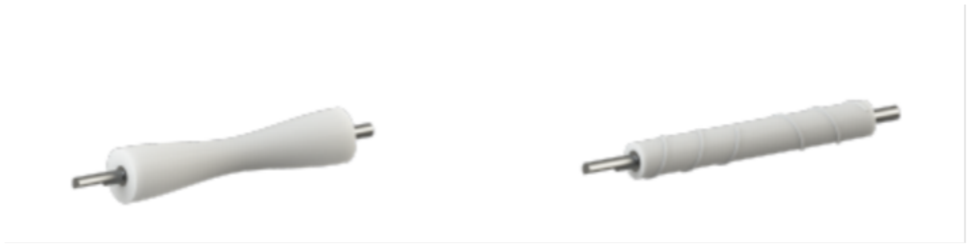


FIGURE 2.11 HOURGLASS (LEFT) AND THREADED STRAIGHT (RIGHT) ROLLERS ROBOTEAM TWENTE 2022

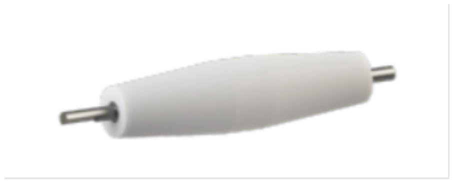


FIGURE 2.12 CROWNED PULLEY ROLLER ROBOTEAM TWENTE 2022

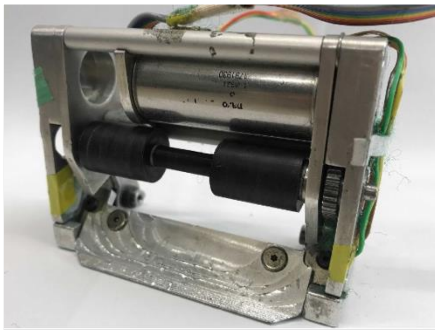


FIGURE 2.13 SEGMENTED ROLLER KIKS 2022

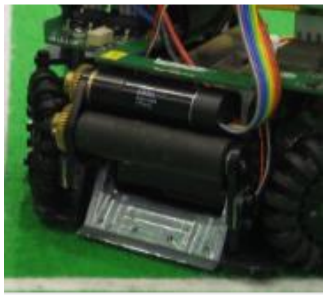


FIGURE 2.14 STRAIGHT ROLLER ROBODRAGONS 2022

The five different designs are hourglass, straight, threaded straight, crowned pully, and segmented. Each of them has its benefits and disadvantages both on and off the field. Of these five the hourglass and crowned pulley have the most complex fabrication process followed by the threaded straight. This is due to their unique shapes and the need for their material to have a grip. If these were made from aluminum these would be suitable parts for a lathe. However, because we want a material with higher friction like silicon or rubber, they are a challenge for clean turning on the lathe. These materials also take longer to turn, decreasing the efficiency of manufacturing. Notably, both the straight and segmented designs offer more flexibility with the fabrication process. Finally, additional criteria were placed on finding a design that was reliable and had models already out there to guide our design choices. All these factors lead to the selection of a straight bar which can be aluminum tubing with a rubber coating. The exact high friction outside of the roller will be tested as we move into manufacturing.

TABLE 2.6 DRIBBLER ROLLER DECISION MATRIX

Criteria	Weight	Hourglass		Straight		Threaded Straight		Crowned Pulley		Segmented	
	1-5	Score	Weighted	Score	Weighted	Score	Weighted	Score	Weighted	Score	Weighted
Cost	3	3	9	5	15	4	12	3	9	5	15
Weight	3	3	12	4	12	4	12	2	6	5	15
Manufacturability	5	2	10	5	25	4	20	2	10	4	20
Adaptability	4	2	8	3	12	3	12	2	12	2	12
Fixability	3	5	15	5	15	5	15	5	15	4	12
Popularity	2	3	6	4	8	5	10	1	2	1	2
Total Score			62		87		81		54		76

2.1.2.3 CHIPPER

The chipper gives the robot a tactical advantage by getting the ball out of a congested field of play. Whereas the kicker is a direct pass forward along the ground the chipper allows for the ball to take parabolic flight. The ball can be chipped over an opponent or used to pass the ball out of the opponent's reach. The chip kicker works by contacting the underside of the ball and lifting it into the air. The chip kicker can either be a solid piece of metal like aluminum machined into a wedge shape or it can be a modular design that is assembled. The chipper can then be mounted onto either the dribbler or kicker mechanisms. These four design choices were evaluated through the criteria in the design matrix presented in Table 2.7.

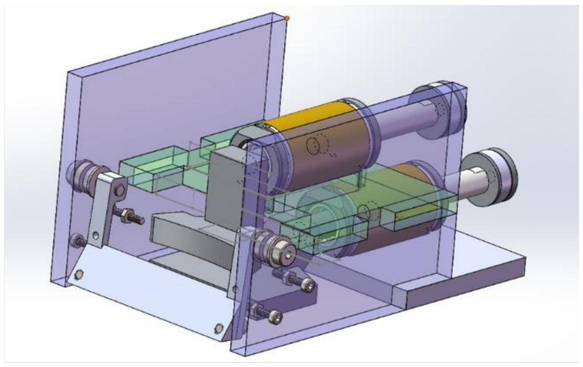


FIGURE 2.15 MODULAR PACKAGED WITH KICKER SRC 2020

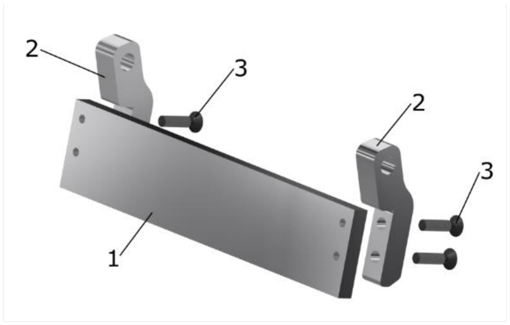


FIGURE 2.16 MODULAR PACKAGED WITH DRIBBLER ER-FORCE 2022

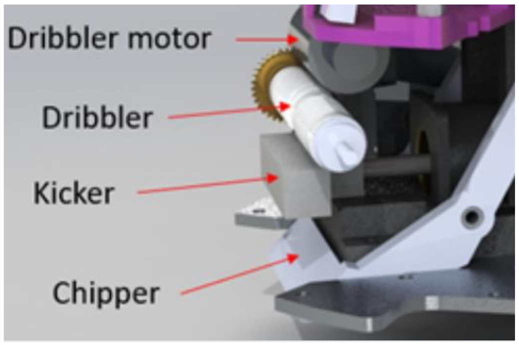


FIGURE 2.17 SOLID PACKAGED WITH KICKER ROBOTEAM TWENTE 2022

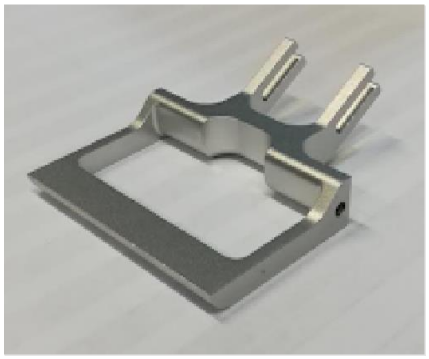


FIGURE 2.18 SOLID PACKAGED WITH DRIBBLER ROBOJACKETS 2019

TABLE 2.7 CHIP KICKER DESIGN DECISION MATRIX

Criteria	Weight	Hourglass	Straight	Threaded Straight	Crowned Pulley

	1-5	Score	Weighted	Score	Weighted	Score	Weighted	Score	Weighted
Cost	3	5	15	5	15	1	3	1	3
Weight	3	3	9	5	15	2	6	3	9
Manufacturability	5	5	25	5	25	2	10	4	20
Adaptability	4	4	16	3	12	3	12	2	8
Fixability	3	3	9	4	12	2	6	2	6
Popularity	2	2	4	4	8	2	4	4	8
Total Score			78		87		41		54

For our team, we chose to go with a modular chip kicker design that will be mounted as part of the dribbler mechanism. This choice made sense for our team from both a manufacturing standpoint as well as a cost one. To design and fabricate a solid chipper wedge, we would need to use a larger amount of stock material as well as follow a more complex machining process. The chip kicker is a small component in an already small robot so any milling into a shape would be on a small scale. Given our manufacturing limitations and lack of experience in making, we opted for the seemingly easier design for manufacturing. The chipper will likely be made from aluminum, however, if the 3D-printed parts prove reliable, we hope to move forward with those.

2.1.3 Drive System

The drive system is the collection of mechanical and electromechanical components that provide the robot with the means to maneuver the playing field. This includes drive motors, wheels, and suspension, which facilitate the robots' motion and turning as informed by their programming. While these components are secondary to the robots' code in terms of game strategy, a robot's performance in executing that strategy can only be as effective as the design of its drive system. It is imperative to the success of a team to design and implement a sufficiently sophisticated and robust subsystem.

2.1.3.1 Wheels

Wheels are the primary contact with the playing surface of our robot. Wheels can make or break the effectiveness of the overall drive system. Careful selection of wheel design is paramount to designing a high-functioning team of robots whose movements are consistent and sharp.

There are a variety of wheel types to choose from. Some designs are more widely used than others and are therefore being considered more heavily for a team in its infancy. The goal for an effective wheel design is one that translates the input from hardware and software into physical movement the most efficiently. Frequent choices for this include Mecanum and Omni-Directional wheels. Traditional wheels are included in this discussion for comparison.

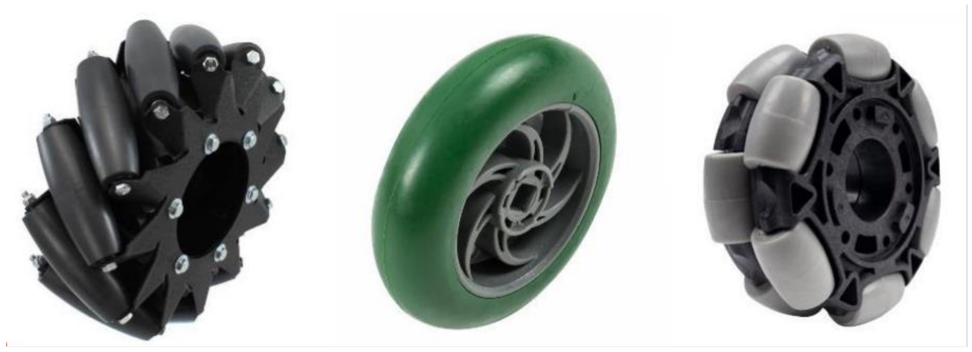


FIGURE 2.19 WHEEL TYPES WE CHOSE FROM MECANUM (LEFT), TRADITIONAL (MIDDLE), OMNI-DIRECTIONAL (RIGHT)

The criteria used to select between these three designs are accessibility, cost, durability, weight, adaptability, turf performance, and agility. Accessibility encompasses the ease of acquisition with regards to purchasing or sourcing through sponsorship as well as access to fabrication machines and materials. Time is factored into the accessibility criteria. Cost is the rating of a design based on how low the price is when considering the purchase of completed parts or raw materials and processing costs. Durability speaks to the resilience of a design to degradation or damage that will adversely impact performance. Weight is the lightness of a design. Adaptability is the ability to modify or optimize the design for the system's needs. Turf performance factors in the ability of a design to resist negative impacts due to the playing surface such as grip or turf getting stuck in rollers. Agility describes the ability of a design to facilitate rapid movements and changes in direction.

TABLE 2.8 WHEEL DECISION MATRIX

Criteria	Weight	Omni-Directional		Mecanum		Traditional	
	1-10	Score	Weighted	Score	Weighted	Score	Weighted
Accessibility	7	9	81	9	81	9	81
Cost	8	8	48	6	48	8	48
Durability	9	7	56	6	48	8	64
Weight	8	5	40	6	48	7	56
Adaptability	7	5	35	5	35	4	28
Turf Performance	8	6	48	5	40	8	64
Agility	9	9	81	8	72	2	18
Total Score			394		328		295

Omni-Directional wheels are the standard for high-performing teams in the Small Sized Soccer Robot League. They provide the same functionality as Mecanum wheels while being significantly simpler to adapt or manufacture. There is also a wide existing selection of these parts from a variety of suppliers to choose from.

2.1.3.2 WHEEL CONFIGURATION

Wheel configuration influences the maneuverability of a robot. Some configurations are more articulate and conducive to precise movement than others. Wheel layout must be compatible with wheel selection as well as hardware and software design.

The predominant designs for wheel layout are three-wheel Omni-Directional and four-wheel Omni-Directional. There is room for adaptations in wheel offset and angle within these designs, which is factored into the decision matrix in Table 2.9. An obsolete configuration is included in this design matrix for comparison, as differential-based rear-wheel drive and front wheel steering was only briefly used early in the Small Sized Soccer Robot League.

The selection criteria for configuration include cost, weight, adaptability, simplicity, and agility. Adaptability is where the offset and angle optimizations are factored in. Simplicity describes the resistance of the design to issues related to unnecessary complexity.

TABLE 2.9 WHEEL CONFIGURATION DECISION MATRIX

Criteria	Weight	Front Steer/RWD (obsolete in Small Soccer Robot League)		Three Wheel O-D (UAdelaide 2005, Tech United Eindhoven 2017)		Four Wheel O-D (Robojackets 2007-2019, SKUBA 2010-2014, CMDragons 2009-2016, EagleKnights 2006)	
	1-10	Score	Weighted	Score	Weighted	Score	Weighted
Accessibility	8	6	48	7	56	8	64
Cost	8	6	48	6	48	7	56
Durability	7	4	28	7	49	7	49
Weight	6	8	48	7	42	8	48
Adaptability	9	3	27	7	63	8	56
Total Score			199		183		259

Like wheel selection, the four squarely opposed wheel configuration is the norm for use with Omni-Directional wheels in successful teams. This design offers the same maneuverability as other Omni-Directional configurations while eliminating complexity caused by asymmetry associated with triangular three-wheel designs. Four squarely opposed wheels also unlock speed capabilities with an additional drive motor. There is also potential for optimization with angular and positional offsetting of wheels.

2.1.3.3 SUSPENSION

Suspension systems in Small Sized Soccer Robots are an uncommon choice. Teams rarely implement them due to their relatively small impact on their envelope in the internals of the robot. However, if implemented effectively, their benefits can outweigh their drawbacks:

TABLE 2.10 PROS VS. CONS OF USING SUSPENSION

Pros	Cons
• Reduces bouncing due to uneven wheels	• Can introduce instability if not calibrated properly

• Keeps wheels in solid contact with turf for maximum grip • Can be adjusted to preference depending on the system • Reduce rigidity/stress on chassis and adjacent components	• Unpredictable variance in kicking/dribbling due to rocking of chassis • Introduces potentially unnecessary complexity to design (most teams have wheels and motors hard bracketed to chassis)

Options for suspension systems include coil overs, sway bars, and double wishbone. Suspension systems are more frequent in the middle-sized soccer robot league. There is little precedent for suspension in Small Size Soccer Robots, so a 'no suspension' option was included in the decision matrix (Table 2.11) for comparison.

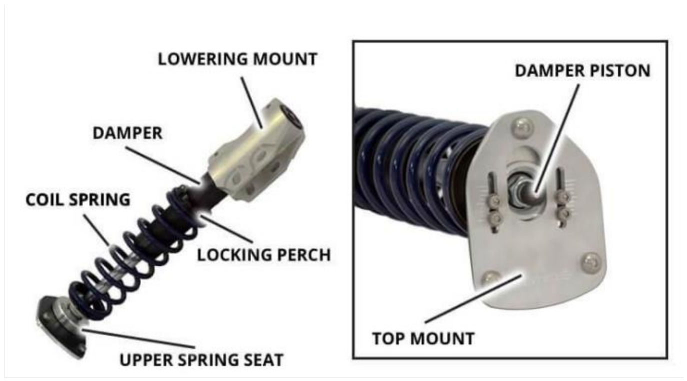


FIGURE 2.20 COILOVER SUSPENSION [20]

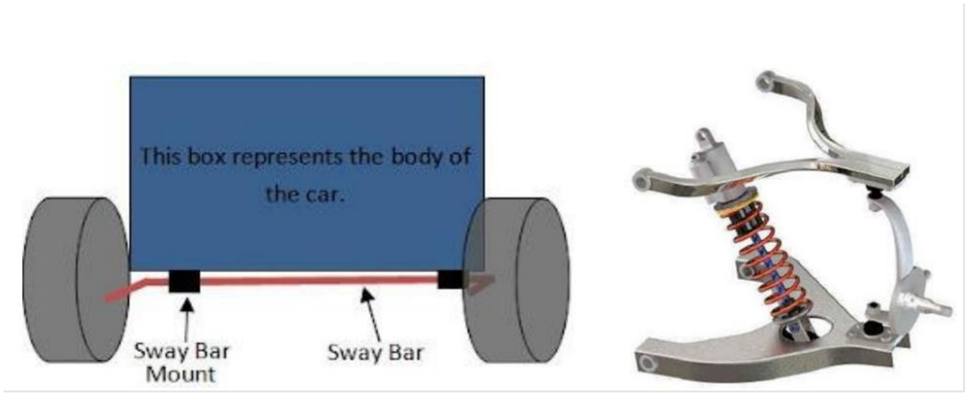


FIGURE 2.21 SWAY BAR SUSPENSION (LEFT) AND A DOUBLE WISHBONE SUSPENSION (RIGHT)

The selection criteria for a suspension system include accessibility, cost, durability, weight, adaptability, size, and simplicity. The size criteria refer to the spatial envelope requirement of the design, which speaks to the design's potential to avoid obstruction by other internal subsystems or from obstructing other internal subsystems.

TABLE 2.11 SUSPENSION DECISION MATRIX

Criteria	Weight	Coilovers		Sway Bars		Wishbone (Hybrid)		None	
	1-10	Score	Weight	Score	Weight	Score	Weight	Score	Weight

Accessibility	7	7	49	6	42	6	42	10	70
Cost	8	6	48	6	48	5	40	9	72
Durability	7	6	42	7	49	5	35	7	49
Weight	6	7	42	8	48	5	30	10	60
Adaptability	6	7	42	7	42	8	48	6	36
Size	8	6	48	7	56	5	40	9	72
Simplicity	8	6	48	7	56	5	40	10	80
Total Score			319		309		267		333

The potential benefits of a suspension system are slightly outweighed by their introduction of unnecessary complexities. As a point of improvement down the line for a more mature team, a suspension system can have subtle benefits. For a team in its infancy such as ours, the implementation of more advanced mechanical subsystems such as double wishbone designs can do more harm than good by taking focus away from establishing a strong fundamental mechanical base. Most successful teams also opt to not include suspension systems in their designs.

2.1.3.4 MOTORS

Motor selection for Small Size Soccer Robots is very important as the motors provide the actuation and allow the robot to move and perform higher-level functions. DC brushless motors are the optimal choice for this application because they provide a reasonable amount of torque, have long lifespans (no brush friction/wear), and require less maintenance than traditional DC motors.

The selection criteria for motors are more empirical than the rest of the design, which include motor cost, voltage (24V), power rating (between 50 and 70 watts), motor weight, rated angular velocity (ideally between 4500 and 6000 RPM), and rated nominal torque (around 60 to 130mNm). The selection criteria were formed largely around the existing motor specifications in the league. The Maxon EC-45 is common within the league due to its flat, compact design and reliability, so this was used as the general standard. Furthermore, teams use different power ratings of the EC 45, so we chose 50W as the middle ground (teams range from 30-70W). The three main candidates for motor selection are the Maxon EC 45 (50W), the DF 45 (65W), and the EC24527 (62.2W).

TABLE 2.12 MOTOR DECISION MATRIX

Criteria	Weight	DF-45		Maxxon EC-45		EC24527	
	1-10	Score	Weighted	Score	Weighted	Score	Weighted
Cost	9	8	72	6	54	TBD	TBD
Weight	6	8	48	7	42	9	54
Torque	8	9	72	10	80	9	72
RPM	8	9	72	8	64	9	72
Voltage	7	10	70	10	70	10	70
Power	8	7	56	8	64	7	56
Total Score			390		374		TBD

The motor selection for our drive system is the DF 45 due to it meeting all design criteria and being cost-effective compared to other similarly specked motors, for $148.70 per unit. The EC 45 is priced at $218.09 for a single unit, meaning the DF 45 is $69.39 cheaper per unit. The DF 45 also has a slightly lower nominal torque rating than the EC 45 (130mNm vs 144mNm) which substitutes for a higher nominal angular velocity (4840 RPM vs 4500 RPM). It is also 110g lighter than the EC 45, a saving of 440g per robot.

If the EC2457 can beat out the DF 45 in price, this is our optimal choice, because it matches or is slightly better than both the DF 45 and the EC 45 in nearly every category and could potentially save even more money.

2.2 ELECTRICAL BACKGROUND

To complete a literature review for this project, we researched existing Team Description Papers (TDPs). A TDP is an official paper each SSL team must submit to be eligible to compete in the league. It describes the design of the team's system, like a concise version of the MQP report. These papers are publicly available and describe general system architecture that is an excellent starting point for research. These TDPs have been published once a year, dating back to 1996, so there was a vast amount of knowledge to pull from. Analyzing a single team and following its progress throughout the years was a very effective mechanism for determining what design choices went well and what didn't. Also, as the years go on, the TDPs shorten in length and description as they tend to remark more deeply on the novel portions of their systems.

The kicking system of the soccer robots have been described frequently in the TDPs. To power the kicking mechanism, teams typically use capacitor banks of 200V capable of kicking the ball between 10 and 20 m/s. Teams began to add chipping mechanisms to their kicking systems after several years of the league's existence. These feature wedges at an angle capable of lifting the ball in the air and shooting it over robots. [4] Earlier in competitions, "it was common for teams to encounter voltage drops when their kicker solenoid was activated. This plagued teams who would sometimes face brownouts because of kicking the ball. The solenoid current draw could cause a voltage drop low enough that would cause the processor to fall into sleep mode or, in the worst cases, turn off completely." [5] Teams solved this problem by either isolating the kicking circuit or partitioning the power supply into a kicking battery and a non-kicking battery.

The dribbling module is less sophisticated electrically than the kicking module, and remains similar between each team, even over time. Teams use a single BLDC motor to control the dribbler, which is a rotating shaft that comes in various shapes (see the mechanical engineering MQP report for more information). The only main difference electrically is the choice of using closed-loop encoder feedback for precise prescribed angular velocities, or open-loop control systems calculated based on the rated load for the motor. The main factors at play for this decision are cost and complexity of design.

The most variation in sub-system design for electrical modules is found in the power distribution module of the robot: the onboard battery charging circuitry is often different. "Some teams had battery cells designed to be easily removable from the robot chassis and opted to charge the batteries outside the robot on a dedicated battery charger. Others integrated a basic charging circuit onto their robot, allowing the robot to be charged over a simple DC voltage input." [5]

This decision is one rooted in preference and cost, rather than performance of the robots, as the method of charging the cells will not change the way the power distribution occurs.

In terms of the drive system, throughout the years there were numerous mechanical configurations that were tried, such as wheel type, orientation, number of wheels etc. On the electrical side of things, the system is controlled by brushless direct current motors (BLDC) for their long lifespan and ease of control in custom applications. The main differences in drive systems occurs in the implementation of the motor control: teams either using electronic speed controllers, integrated circuits, or custom BLDC control circuits designed by the team. Teams that created custom BLDC drive circuits often used FPGAs, which are expensive, and were used on teams that had the budget to implement them.

In conclusion of the literature review, we became equipped with the electrical module and component options and benefitted from knowing how they performed. The remaining key factor for the choice of components was cost, as the budget was quite limited for this project.

2.2.1 Robot Requirements

The next step for us was to lay out all the basic desired requirements necessary to accomplish the goal of the project. These goals were kept broad so that the team could change course when needed. The goals contain smaller sub-goals that are described in the System Requirements, section 3.1.1.1.

2.2.1.1 POWER BOARD

The robot requires a power board capable of supplying power to all components on the robot, even when operating under strenuous conditions, and must have the ability to do all this and last for an entire robocop game. The drive motors typically draw a large amount of current when operating under load, and sometimes have peak current draw. However, peak currents are unlikely to happen on all motors simultaneously, but the power board should be capable of providing the necessary current to all motors under load while dealing with one motor drawing a peak current.

The board should also be capable of providing appropriate voltages for the motors as well as several other components, particularly 5V and 3.3V to logic driven devices such as processors since these are common operating voltages. Motor voltage ratings do not need to be met as stringently as any current ratings, so the voltage for the selected motors should be approximately met.

Robocup matches in the small size league last ten minutes standard, and fifteen minutes if the game goes into overtime. The robots should always be prepared to go into overtime, so the power board should be designed to operate the robots for a minimum of fifteen minutes.

2.2.1.2 KICKER BOARD

To operate the solenoids for chipping and kicking the ball, a kicker board needs to be designed. Solenoids for this application need to draw a lot of current, and that is only possible if the resistance of the solenoid is overcome by a large voltage. This voltage is typically greater than what the voltage on the power board would provide, so the kicker board needs to be capable of converting the voltage from the power board to a much higher voltage. It must be able to operate the solenoids at this voltage and at high currents, as well as operate the solenoids in a relatively frequent manner.

2.2.1.3 PROCESSOR BOARD

A processor board is necessary to turn decisions made by software into actions utilizing hardware. This board requires several aspects of functionality, which can be summarized into executing firmware quickly, wireless communicating with a master controller, controlling the motion of the robot, and controlling the hardware for ball manipulation. This board was the focus of Noah Page and was completed by him early in the project. Refer to his report for greater detail regarding its design and component selection.

2.2.2 COMPONENT SELECTION

The following sections describe the process and decisions for selecting specific components for the power and kicker boards, as well as additional components to operate the three phase motors.

2.2.2.0 BATTERY CELLS

It was important to have battery cells capable of high continuous discharge current and a 3-amp power storage capacity, all at a moderate price point. The Molicel P28B was selected due to its unique material properties that allow it to meet the requirements, with a large thermal safety net. Cells with a similar discharge rate to the Molicel P28B do not satisfy the requirements for storage capacity, which is why all other options were eliminated.

2.2.2.1 VOLTAGE REGULATORS

Voltage regulators are essential for the power board to provide voltages lower than the voltage of the battery pack. Three voltages are necessary for boards and components to function: 12V, 5V, and 3.3V. Other considerations were price and availability, mainly to preserve the budget for more expensive items. The LT2575-5 LDO was selected for 12V and 5V, while the LD1117V33 was selected for 3.3V. These regulators were selected because they were the cheapest and most available compared to other models.

2.2.2.2 BATTERY MANAGEMENT

A battery management chip is an important component used to monitor the voltage and current of the battery cells. The BQ76925 front end multiplexer was chosen because of its low cost compared to other models and ability to work with our selected cell type.

2.2.2.3 SOLENOIDS

It was found during research that many competing robots use solenoids to propel the ball with the forces required to be able to play the game. These teams however made custom solenoids. This option was not an option for the team due to the lack of manpower and knowledge for that that, so solenoids available on the market were researched. The solenoids from the magnetic sensor systems series S-20-100-H push type were selected. In the purchasing options, there are a few strands, so there are four options in relation to duty cycle. These change the number of turns in the solenoid when selecting. For the desired configuration, the pulse solenoid was chosen. The plunger for the solenoid weighs about 0.34kg, which is nearly negligible in this case with a change to 0.35 kg.

2.2.2.4 CAPACITORS

Large capacitors are necessary for the kicker board to operate the solenoids efficiently. The capacitors need to satisfy the following conditions: a voltage rating of 250V, a discharge current of 15 amps, and operate at a temperature greater than 75 Celsius. After researching capacitors, the Nichicon LGG2D152 was chosen because it met the voltage rating and discharge current requirements and operates at a temperature of 85 degrees Celsius, while having the best purchase price compared to other models.

2.2.2.5 ENCODERS

The encoders need to be able to attach to the motor axles being used and fit within a compact area, while having good resolution and reasonable price. After extensive research, only two encoders were found that fit the dimension of requirements of 6mm or less in thickness between the motor bracket and wheel and can fit on a 4mm diameter shaft. The first option was the E4T from USDigital, which could have a pulse resolution between 256 and 2048. The second option was the CUI Devices AMT-112Q-V, which features a programmable PPR between 128 to 4096, with an interchangeable grip for differing shaft sizes between 3 and 7 mm.

TABLE 2.13 ENCODER DECISION MATRIX

Criteria	Weight	E4T		AMT-112Q-V	
	1-5	Score (1-5)	Weighted	Score (1-5)	Weighted
Shaft Diameter	5	4	20	5	25
Thickness	5	5	25	20	20
Resolution	4	4	16	5	20
Cost	3	2	6	3	9
Total Score			67		74

The AMT-112Q-V was chosen over the E4T because the AMT-112Q-V fits the criteria better. The AMT-112Q-V is $34.95 per unit, while the E4T is $37.00 per unit. The AMT-112Q-V has better resolution and is more flexible regarding what shaft it can fit on, providing greater flexibility for development. The one benefit the E4T has over the AMT-112Q-V was its smaller thickness. However, this difference in thickness is small and was not a significant factor in the decision matrix.

2.2.2.6 CHARGING IC

To charge the capacitors to high voltages, an IC designed for this task is needed. The LT3751 was arbitrarily chosen to be the intended IC, but that chip had supply shortages, so its simpler counterpart was chosen, the LT3750. The LT3750 is an isolated capacitor charger and is used to boost the voltage from the VCC input.

2.2.2.7 HIGH POWER TRANSISTOR

There are three main options available for high power transistors: power BJTs, IGBTs, and power mosfets. Each class of transistor fills a similar role but has different properties to fill that role. Three properties had to be met: high switching frequency, voltage dependent output, and consistent switching delays. Out of each three types of transistors the best one was IGBT as its output is dependent upon input collector voltage and it is not dependent upon its input current.

The requirements for IGBTs were broken down into properties such as voltage and continuous output current. After further researching and eliminating several other options, the IKW7560T was selected. It has a max voltage rating of 600 volts and a current amperage of 75 watts.

2.2.2.8 MOTOR CONTROLLERS

In order to operate the three-phase drive and dribbler motors, motor controllers and electronic speed controllers (ESCs) were researched. These devices needed to meet the design criteria, which included voltage (24V), current(3A-4A), cost, and simplicity. Using these specifications, the search was narrowed down to two devices: the Allegro A3930/3931 motor controller, and the Cyclone CY-35A ESC.

TABLE 2.14 MOTOR CONTROLLER DECISION MATRIX

Criteria	Weight	A3930/3931		CY-35A	
▨	1-5	Score (1-5)	Weighted	Score (1-5)	Weighted
Voltage	4	5	20	5	20
Current	4	4	16	5	20
Cost	3	2	6	4	12
Simplicity	5	1	5	3	15
Total Score	▨	▨	47	▨	67

The device selected to operate the three phase motors is the CY-35A due to it meeting the design criteria better than the A3930/3931. Both devices meet voltage and current requirements adequately, but the CY-35A is a better choice for cost and complexity. At $10.25 per unit, the ESCs are relatively cheap, even with an additional investment cost of $14.39 for a device to flash the firmware on the ESCs. This is considerably less costly than the A3930/3931. While the IC costs $6.50 per unit, other components would have to be purchased per unit in order to assemble a circuit and make the chip functional. Additionally, a custom PCB would have to be designed in order to implement the circuit, which is much more expensive compared to the ESCs.

2.3 SOFTWARE BACKGROUND

For the software research, we identified the following game aspects:

- Vision Module
- Strategy
- Controls
- Communications
- Testing and Simulation

2.3.1 VISION MODULE

The Vision Module takes camera data from SSL Vision to tell the master where the robots and ball are located. This section will cover our research into SSL Vision.

2.3.1.1 SSL VISION

Small Size League Vision (SSL Vision) is the shared vision system software developed by RoboCup, and now the standard in competitions (Zickler, Bruce, Biswas, Licitra, & Veloso, 2009). It is an open-source software developed in C++ that utilizes the Qt toolkit, which is a toolkit specializing in creating graphical user interfaces and cross-platform applications (Zickler, Laue, Birbach, Wongphati, & Veloso, 2009). According to RoboCup SSL's vision system document published in 2009, they use multiple plugins to process the camera data. It begins by taking the camera image and using a lookup table to label different objects on the field based on color. It goes through each pixel to create a threshold image. Then, similar colors are grouped into regions and the plugin calculates the bounding boxes and centroids of each region. A pattern detection plugin is used to identify the location and orientation of the robots using the colored dots on top of the robots (Zickler, Laue, Birbach, Wongphati, & Veloso, 2009). Finally, the information is shared with the teams via network. Teams can choose how to use this information. For example, the 2009 CMDragons team uses the information in one of their two clients – the client that deals with graphical interfaces for controlling and monitoring. In 2010, Skuba used SSL Vision in one of their three servers, the vision servers which feed information into their controller. In general, having this SSL Vision shared system allows for easy calibration and interchangeability with different image processing plugins (Zickler, Bruce, Biswas, Licitra, & Veloso, 2009).

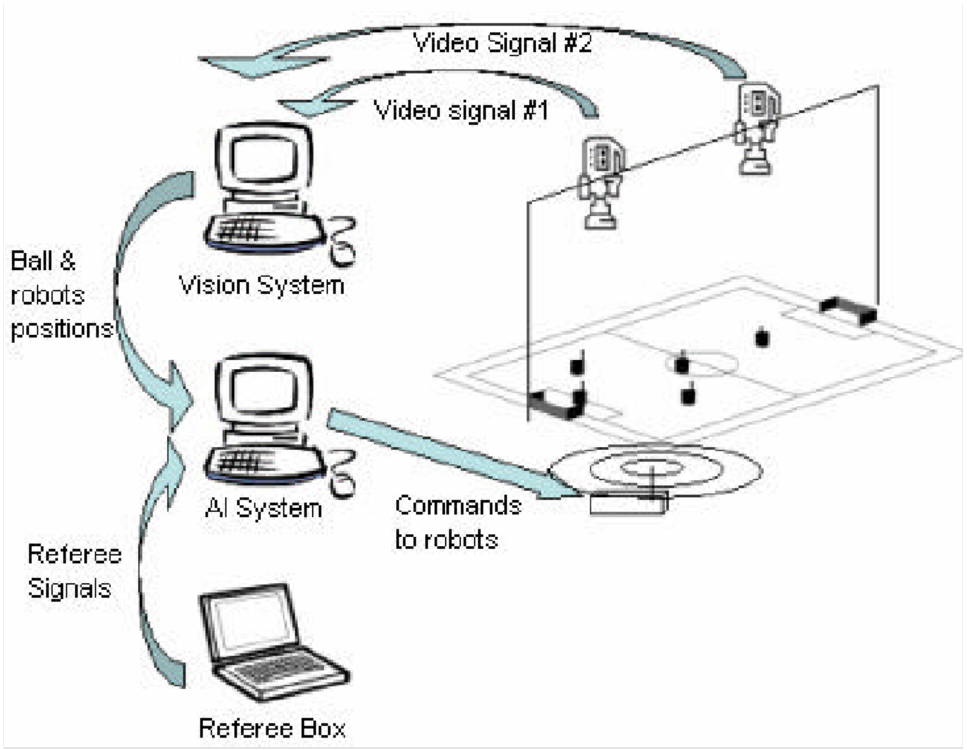


FIGURE 2.22 ROBOCUP SSL VISION SYSTEM (MARTINEZ-GOMEZ, L ET AL., 2005).

The SSL Vision System is responsible for collecting field data such as position of the ball, positions of the soccer robots, and velocities of them. These data will be sent to the robots and used as inputs to our AI system.

2.3.2 STRATEGY MODULE

The strategy module contains all the decision making and planning needed to determine what each of the individual agents should do. There are three main sections in most SSL teams' code structures. First comes the building blocks: Tactics and Skills. These are the base functionalities of the agents that can be combined to form more complex roles and strategies. Next comes the methods of determining which of these tactics and skills should be used in the Heuristics section. Finally comes the Role Assignment section which chooses which agent will take on each of the designated roles built from Tactics and Skills, then selected by the Heuristics section.

2.3.2.1 TACTICS AND SKILLS

The CMDragons created the STP Architecture, or Skills, Tactics, and Plays Architecture that modern teams use to implement their strategies. Tactics and Skills are used for implementing individual robot behavior to achieve a role. Multiple robots performing roles in parallel creates a play. Tactics and Skills are low-level primitive behaviors of the robots that are carried out in parallel by each robot to achieve the selected play. There can be multiple Skills in a Tactic, and multiple Tactics in a role, depending on the play's complexity.

Skills are the lowest-level control behaviors implemented in the robots, such as navigating to a point safely (implemented using the RRT algorithm), shooting a ball with a specified velocity, dribbling the ball up the field, or chip-kicking a ball to a certain location. Skills may be performed in series to implement a Tactic. Skills make basic use of the hardware and are the building blocks for more complex behaviors.

A Tactic helps a robot fulfill a certain role, such as being an attacker, defender, or goalkeeper (Bowling, 2016). It is a macro of skills that one robot carries out to achieve a plan or play. Examples from the CMDragons and the Robojackets include a goalkeeper blocking shots (combination of moving to point multiple times), two robots passing the ball to each other (moving to points, using shooting mechanism to pass), defenders preventing passing/shooting (moving to points), a robot retrieving a loose ball (moving to point and dribbling), and attacking robots moving the ball to the goal to shoot and score goals (moving, dribbling, kicking) (Zickler, Bruce, Biswas, Licitra, & Veloso, 2009; Bardagjy et al., 2008). From their 2019 TDP, ETDP defines a "harass" tactic, which determines the opposing robot to harass to attempt to intercept a pass or force a turnover (Bootsma et al., 2019).

2.3.2.2 HEURISTICS

Like determining roles for individual robots, the overall optimal play that is chosen also needs some form of heuristic(s) to be based on. A heuristic is a technique designed to solve a specific problem faster. Many variables in the world state are used as heuristics to determine which play the strategy controller converges on. For example, in 2016, the CMDragons used an aggressiveness level, fulfilled by several "applicability conditions" as their heuristic to determine their play in any situation. These conditions were ball possession, field region, and a calculated opponent's level of aggression.

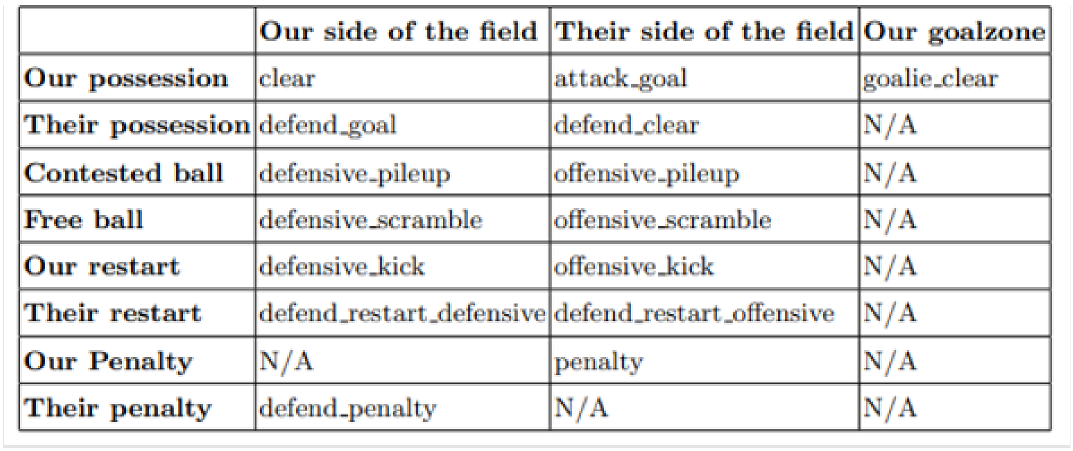
$$
\mathbf{ballP}_t = \begin{cases} \text{ourB} & \text{if } (t^{\text{us}}_{\text{near}} > t^{\text{thresh}}_{\text{near}}) \wedge (t^{\text{them}}_{\text{near}} < t^{\text{thresh}}_{\text{near}}) \\ \text{theirB} & \text{if } (t^{\text{us}}_{\text{near}} < t^{\text{thresh}}_{\text{near}}) \wedge (t^{\text{them}}_{\text{near}} > t^{\text{thresh}}_{\text{near}}) \\ \text{contendedB} & \text{if } (t^{\text{us}}_{\text{near}} > t^{\text{thresh}}_{\text{near}}) \wedge (t^{\text{them}}_{\text{near}} > t^{\text{thresh}}_{\text{near}}) \\ \text{looseB} & \text{if } (t^{\text{us}}_{\text{far}} > t^{\text{thresh}}_{\text{far}}) \wedge (t^{\text{them}}_{\text{far}} < t^{\text{thresh}}_{\text{far}}) \\ \mathbf{ballP}_{t-1} & \text{otherwise} \end{cases},
$$

where $t^{\text{thresh}}_{\text{near}}$ and $t^{\text{thresh}}_{\text{far}}$ are time thresholds.

FIGURE 2.23 BALL POSSESSION CALCULATION BY CMDRAGONS

Referring to Figure 2.24, ball possession is broken up into a piecewise function of time. To determine which state the ball possession is in, CMDragons keeps track of the amount of time the ball is near either team (taken from an arbitrary reference time), defined as "t us near" and "t them near", where "near" is an experimentally determined distance between the ball and either team. They then compare "t us near" and "t them near" to a predetermined constant time threshold and use a combination of these comparisons to determine the ball possession. The "ourB" ball possession state means that the CMDragons have possession of the ball, since the ball has been close to them longer than the time threshold, and close to their opponents less than the time threshold. "theirB" means their opponents have the ball, "contendedB" means it is undecided who has the ball and likely will be decided soon. "looseB" refers to a ball that is not near either team and therefore the ball is up for grabs. Finally, "ballP(t-1)" occurs when none of the above conditions are satisfied, which means the current ball possession is set as the same as the previous ball possession.

The field region used to calculate the aggressiveness level is simply which half of the field is the ball on: CMDragon's half, or their opponent's half. The opponent's level of aggression is determined by the number of their robots in their defensive half field. If they have all robots in their half, aggressiveness is set to defense, otherwise it is set to 0. Furthermore, these conditions break down into detailed functions. For example, ball possession is calculated based on the time the ball spends near either team in conjunction with its distance relative to each team (each variable has their own adjustable threshold). The combination of these many variables makeup the heuristic that then allows the team to determine which play to use, and subsequently which roles to assign to each robot (Mendoza et al., 2016).



	Our side of the field	Their side of the field	Our goalzone
Our possession	clear	attack_goal	goalie_clear
Their possession	defend_goal	defend_clear	N/A
Contested ball	defensive_pileup	offensive_pileup	N/A
Free ball	defensive_scramble	offensive_scramble	N/A
Our restart	defensive_kick	offensive_kick	N/A
Their restart	defend_restart_defensive	defend_restart_offensive	N/A
Our Penalty	N/A	penalty	N/A
Their penalty	defend_penalty	N/A	N/A

FIGURE 2.24 GAME CONDITIONS THAT TRIGGER ROBOJACKET'S ACTIONS

Another way to decide how to determine what play to implement is the division of potential circumstances as demonstrated by the RoboJackets. Their team breaks down the game into a set of game conditions based on factors like ball possession, field side, and referee input. These instances have a set strategy to go with them and are designed to optimize the outcome and lead to more goals scored. Referring to the table in Figure 2.25, we can see what game conditions (heuristics) choose the corresponding actions for the RoboJackets. For example, if the ball is in the RoboJacket's possession, and is on their side of the field, their action should be to clear the zone. If it is their opponent's possession and is on the RoboJacket's side of the field, the play is to defend the goal. Setting up a matrix like this is very simple and can help discretize a very complex game.

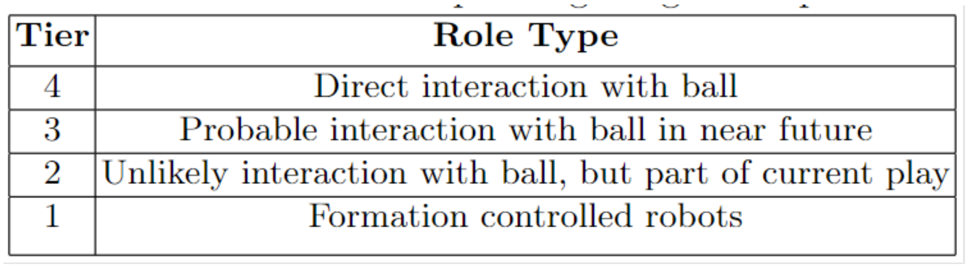
Another important and powerful play selection heuristic that can be implemented is reinforcement learning. When used in the context of the RoboCup, this would factor in previous plays within a game to learn the opponent's strategy and to adapt to it as the game moves on. A popular reinforcement learning algorithm is Q-learning, which is a model-free reinforcement algorithm that learns the value of an action in a particular state. Q-learning has been used in other Robocup leagues to further certain abilities of participating robots such as "shooting the ball while walking." Q-learning can also be helpful to filter out noise by fine-tuning control parameters after several iterations (Fadelli & Xplore, 2021).

2.3.2.3 ROLE ASSIGNMENT

In RoboCup, a robot may need to fulfill different roles, or positions, throughout the game depending on who has possession or the current play. These roles can look different depending on the team's playbook. For example, in 2010, Skuba had predetermined plays with specific positions (Wasuntapichaikul et al., 2010). Each role contains the set of tactics and skills necessary to execute the role. After the roles that need to be fulfilled have been determined, the next step is to assign a role to each of the agents on the field. Efficiency is a priority in this stage as it cuts down on the time necessary for all agents to move to their assigned positions and therefore makes it more likely to be successful against the opposing team.

Most teams use some variation of the Hungarian Algorithm. The Hungarian Algorithm calculates the cost of a robot fulfilling a role, and then when the costs of all robots are calculated, it chooses the minimum cost solution. The cost determination is where the teams seem to differ the most.

A simple way of calculating the cost is by using distance. This is what MRL did in 2019; they had a role assigner module that calculated the cost based on the distance between the robots' current position, and target location of the role being assigned. One issue with this is sometimes a robot may be close to a position, but since it was previously a defender, it may not be the best fit to leave the defense area and become an “active” role (Ganjali et al., 2019). For this reason, they sorted all their possible positions into two groups: one for defenders and regional roles, second for markers, stop cover, active, and positioner roles. Now, after they calculate the cost, they make sure that a robot is only switched into either the same role or a role in that group.



Tier	Role Type
4	Direct interaction with ball
3	Probable interaction with ball in near future
2	Unlikely interaction with ball, but part of current play
1	Formation controlled robots

FIGURE 2.25 THE TIERING SYSTEM USED BY GEOGIA TECH'S ROBOJACKETS

RoboJackets also used a Hungarian algorithm in 2020, however they used a tiered method of assigning that made sure the most critical roles were optimally assigned (Almagro et al., 2020). They made tiers based on what the role required: direct interaction, probable interaction, unlikely interaction but part of the play, and formation controlled. Formation controlled agents are not likely to interact with the ball in the immediate future, but instead the robots set up to be in default positions waiting until they become pertinent to the play, though they have default behavior also associated with the formation position. This could be staying in a certain zone and blocking the most likely shot on goal, for example.

Using the tiers, they assign all the roles in the highest tier first using distance from the robot to the desired location. Once all tier one roles are assigned, they move on to the next tier, and this process continues until all the roles have been fulfilled.

The RoboTeam Twente, however, ran into the issue that they used different types of robots (Bos et al., 2020). Since there was no longer a homogenous pool to choose from, they had to take into account the skill level of each robot. This led them to come up with a bit more complicated of a cost equation that calculated the cost of the robot traveling to a location and the reward for the robot performing that action. Ultimately, the cost considers a specific robot's distance to the ball, proficiency in completing that task, and the importance of that task. The importance of the task is predetermined and then used in the equation to make sure it’s completed by the best suited robot. They created an equation that takes all these factors into account to assign the cost of different roles for different robots and calculated it by simulating multiple practice scenarios and seeing how different robots act in different positions.

2.3.3 Navigation Module

The navigation module contains all the planning and calculations needed to establish each agent's obstacle-avoidance path as well as the set of wheel speeds required to follow their respective path. There are two main sections that make up the navigation module: Path Planning and Motion Control.

2.3.3.1 PATH PLANNING

Once a destination has been determined by the Strategy Module, a path must be planned, starting from where an agent currently is and ending at the specified destination. To figure this path out, many teams use the Rapid-exploring Random Tree (RRT) algorithm on their master for their real-time path planning. RRT is a tree search algorithm. A Tree is classified as a starting point, an end point, and its branches. For our use, we called points, Nodes, and the branch that connects two Nodes, we called Links. Therefore, our Tree is a starting Node, an end Node, and a list of Links. RRT uses randomization to generate Nodes and then attempts to Link the Nodes to the preexisting Tree. By solely relying on randomization, one can generate a random path from the starting Node to the end Node.

However, a random path is not necessarily the best path. For now, the best path is the shortest path. To make the path shorter, we used a similar cost evaluation as the A-Star algorithm (A*). The cost is the distance from the current Node of the Tree to the end Node. By updating the cost, after Linking a Node to the Tree, we can choose to only add a new Node if its cost was less than the last Node. This would mean the new Node is moving closer to the end Node. By using this cost evaluation with the previous functionality, we create a new algorithm called RRT-Star (RRT*). With this, one can produce a shorter and more efficient path.

Though the path is shorter and more efficient, it is yet to be optimal. An optimal path is an efficient path that avoids obstacles. An obstacle can be defined as both a teammate, an opposing team bot, or even the boundary lines. If the generated nodes, within the path, fulfilled the proper criteria (ie. closer to the goal and did not cross obstacles) then the outputted path would be the quickest path between the starting point and the goal all while avoiding obstacles. This optimized path would look similar to Figure 2.27 taken from an article written by Tim Chinenov, a Software Engineer for SpaceX.

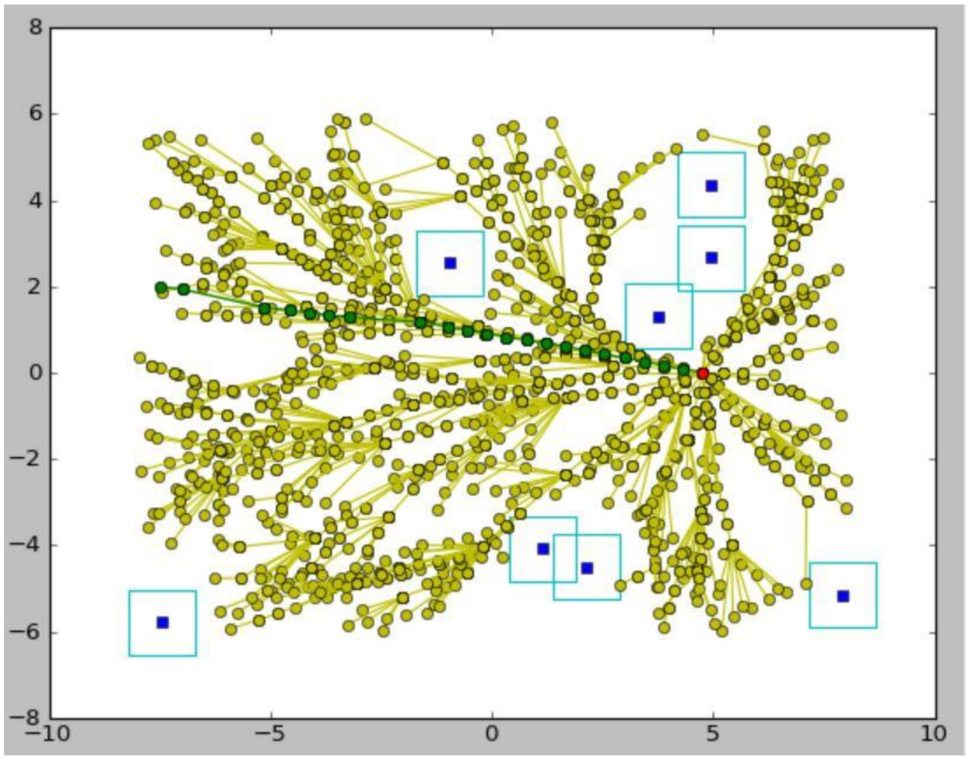


FIGURE 2.26 VISUALIZATION OF RRT* (CHINENOV, 2019)

2.3.3.2 MOTION CONTROL

Generating a path is just the first challenge of the Navigation Module. In order to follow the generated path, the wheel speeds of the agents must be calculated and maintained using kinematics and control theory.

Figure 2.28 below shows the kinematic transformation matrix used by Kasetsart University's team, SKUBA, to go from a desired linear velocity (X velocity, Y velocity, Rotational Z velocity) to the four-wheel velocity (wheels 1-4) that would enable the robot to travel at the desired linear velocity.

$$\psi = \begin{bmatrix} \cos\theta\cdot\sin\alpha_1 + \cos\alpha_1\cdot\sin\theta & \sin\theta\cdot\sin\alpha_1 - \cos\alpha_1\cdot\cos\theta & -d \\ \cos\theta\cdot\sin\alpha_2 + \cos\alpha_2\cdot\sin\theta & \sin\theta\cdot\sin\alpha_2 - \cos\alpha_2\cdot\cos\theta & -d \\ \cos\theta\cdot\sin\alpha_3 + \cos\alpha_3\cdot\sin\theta & \sin\theta\cdot\sin\alpha_3 - \cos\alpha_3\cdot\cos\theta & -d \\ \cos\theta\cdot\sin\alpha_4 + \cos\alpha_4\cdot\sin\theta & \sin\theta\cdot\sin\alpha_4 - \cos\alpha_4\cdot\cos\theta & -d \end{bmatrix}$$

$\psi^\dagger$ is the pseudo inverse of the kinematic equation

α_i is the angle between wheel i and the robot x-axis

$\ddot{\theta}$ is the robot angular acceleration about the z-axis of the global reference frame

$\dot{\phi}_i$ is the angular velocity of wheel i

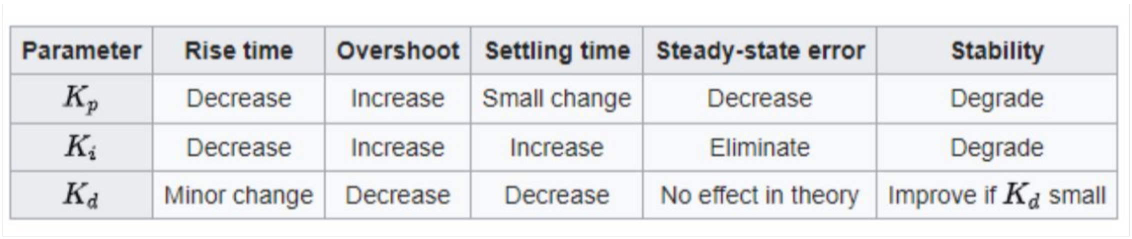
d is the distance between wheels and the robot center

FIGURE 2.27 THE KINEMATIC TRANSFORMATION MATRIX USED BY KASETSTART UNIVERSITY'S SKUBA

Control Theory refers to the use of a control system on a dynamic system. A control system uses controllers within a control loop to manage, command, direct, or regulate the behavior of the system. A controller monitors the controlled variable and compares it with the setpoint. The difference between the actual and the desired value of the variable is called the error. This error is applied as feedback to generate an action or an adjustment that brings the controlled variable to the same value as the setpoint. This described control loop is called feedback control. A feedback control loop involves taking measurements using a sensor and making calculated adjustments to keep the measured variable within a set range or as close to the setpoint as possible. An example of a controller would be the thermostat controlling the temperature of one's home (Simrock 2019).

In this project's case, the agents are the dynamic systems, the controlled variable is their velocity, and the controller used to control their velocities is called a proportional-integral-derivative (PID) controller. A PID controller is a compensator that uses error, the difference between the desired output and the actual output, while taking into account the past dynamics (through integration) of the system as well as anticipates the future dynamics (through derivatives) of the system. The controller then uses the proportional aspect to give weight to the feedback of the compensations the controller is applying. One may use a PID controller on the motors to regulate the velocity output depending on the error of destination of the robot. By controlling the motors, we will be able to turn tightly, move fluently, and stop accurately at high speeds to reach our destination.

Teams in the SSL can choose to use any or all components of PID (ie. P, I, D, PI, PD, ID, PID) to control their motors. Using more of a proportional gain increases the control signal for the same level of error. This causes the system to react more quickly resulting in a faster time to reach the desired output (rise time) but also tends to overshoot the desired output. The addition of an integral gain eliminates the overall error over time. Resulting in a faster rise time, however, causes the system to react slower, causing a slower settling time as well as a longer time to react to an overshoot. Lastly, adding a derivative gain manages the effects of large errors on the system causing a lower overshoot and a faster adjustment time from overshoot to the desired output (settling time) all while having little to no effect on the rise time and overall error.



Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability
K_p	Decrease	Increase	Small change	Decrease	Degrade
K_i	Decrease	Increase	Increase	Eliminate	Degrade
K_d	Minor change	Decrease	Decrease	No effect in theory	Improve if K_d small

FIGURE 2.28 PARAMETERS OF PROPORTIONAL, INTEGRAL, AND DERIVATIVE GAINS (DRIVER PID SETTINGS, 2012)

2.3.3.3 EMBEDDED FIRMWARE

The robots use firmware to enable the hardware to interface with the onboard software. When it comes to controlling the robot, the data gathered from the hardware through the firmware will mainly consist of voltage readings from our drive train as well as data readings from the gyroscope of the inertial measurement unit (IMU). The voltage readings will be used in our above-mentioned PID controller. As for the gyroscope data, it will be used to better estimate the robot's orientation angle. This estimation will be an average between the angle reading from the field camera and the angle reading from the IMU.

2.3.3.4 POSITION ESTIMATION

One problem with the current SSL vision system is the latency. It takes time to process the video and to send the data to the master in charge of the six agents. In this time, the agents are still moving, meaning the Strategy and Controls modules would be operating on positions that are out of date, thereby any decisions made would not be optimal to the current situation.

One way to combat this is to predict the current positions and orientations of the agents based on previous vision data such as positions and velocities. The Kalman filter is commonly used by multiple teams to estimate the exact and current positions of the agents. State space variables, such as position and velocity, are tracked and estimated using weighted averages. Noise is addressed by the weights of the measurements to account for the differences in certainty in measurements. This filter can run in real-time, which is an important quality for any algorithm in this area.

Some teams, like SKUBA and RoboJackets have gone further and used a modified Multi-Hypothesis Extended Kalman Filter (MHEKF) to more accurately predict the current positions of the agents by running multiple extended Kalman Filters in order to increase the probability that one of them converges on the correct positions. Multi-Hypothesis Extended Kalman Filter is a variant of the Extended Kalman Filter that calculates the chances of a robot's real-time location at a certain position using a multi-model probability distribution. It could be very effective when predicting the positions of soccer robots and the field, especially when the SSL vision data is lagging behind.

2.3.4 COMMUNICATIONS

Communications between each device on the field are also crucial to teamwork and data processing. There are a few ways for robots to communicate with robots. No rule restricts teams from using other kinds of communications. However, ball and robot movement detection, which are performed by an SSL-vision system, require signal transmission based on a UDP network. In addition, other components of the SSL software such as AutoRef are using TCP network parameters. Hence, TCP and UDP wireless communications are both required by the competition.

TCP and UDP refer to two different packet formats used in network communication. A data packet in network communication has two major parts: headers and payloads. TCP/IP headers contain the destination and intermediate IP or MAC addresses. UDP headers contain source and destination port numbers. Apart from that, they both contain a payload where all information is stored. When transmitting TCP or UDP packets, the receiver should parse the headers and extract data from the payloads. In RoboCup SSL vision, information such as the position and velocity of the ball is packed up in a TCP/IP packet and sent to the robots through wireless network communication.

Network-based communications using TCP or UDP are much more reliable and faster. The only difference between TCP and UDP is that TCP sends data and simultaneously checks errors for the signal but UDP has limited error checking functions. Furthermore, TCP guarantees the delivery of the packets, but UDP does not. TCP is better for sending signals that contain detailed information at the cost of a slower speed while UDP is better for sending signals that are less detailed but needs to be sent urgently.

2.3.5 Testing and Simulation

Robot software development usually requires a full functional real robot for testing. However, developers may suffer from all kinds of hardware problems which would cause the testing to become inaccurate and time consuming. A handy real-world simulator will make software development for robots a lot easier because we no longer need to worry about hardware failure and testing can be done without the need for an actual field which we don't have yet.

A good simulator for RoboCup Small-Size Soccer League must have great solutions to the omni-directional movement structure and ball kicking or dribbling mechanism, especially when the opponent's robots are trying to contest for the ball. These two features are rather important because they are very unusual compared to other moving mechanisms or structures in many real-world physics simulators, which means it could be very hard or impossible to simulate. In addition, incorrect simulation means meaningless testing without the actual robots and fields, which would ultimately affect our deliverables.

grSim and ER-Force are designed specifically for the RoboCup SSL and maintained by the teams. They are automatically subject to SSL-simulation-protocol. Non-SSL simulators such as NVIDIA PhysX 4 need modifications to implement SSL-simulation-protocol. In the literature review, we will focus on grSim, ER-force, and PhysX. Other open-source simulators such as UberSim, SimRobot, and Gazebo are also available on the internet. It is certainly not impossible for us to use them, but they have no significant advantages over either grSim, ER-force, or PhysX in terms of robot soccer simulation and therefore not wasting time introducing them individually.

grSim is a 3D simulator mainly written in C++. It is developed by Amirkabir university's small size soccer robot team named "Parsian". What makes this simulator brilliant is the careful treatment of omni-directional drive and kicking mechanism. To reduce the deficiencies of modeling omni-directional movement, wheels are constructed as solid cylinders attaching to the robot's chassis with an ODE's (Open Dynamic Engine) Hinge joint in grSim. An angular motor with configurable limited torque is attached to each joint (Monajjemi, Koochakzadeh, & Ghidary, 2012). To simulate the sub-wheels' movement, an additional type of friction other than tangential and perpendicular friction is added to the wheels. By configuring the size of the additional friction, omni-directional movement can be achieved. In terms of the kicking mechanism, a cylinder spinning backward is added to simulate the actuator on the robot. In reality, the mechanics are designed in a special way to prevent the soccer ball from spinning sideways. It would be rather deficient to create an exact model in the simulator and apply physics to it. However, grSim allows the users to configure frictional force on each direction to make sure the ball is not spinning sideways similar to the omni-directional drive. Furthermore, the grSim is supported on all platforms and the GUI interface looks very straight forward.

ER-force is a simulator developed by the ER-force team in 2019 and released as an open-source simulator for RoboCup SSL recently. The developers of the simulator had experience using grSim and found it unable to meet their requirements. They suggested this simulator is better than grSim in most aspects. One of the key improvements ER-force simulator has over grSim is the ability to simulate vision noise (Bergmann et al., n.d.). In reality, it is very common that part of the ball detection data package could consist of noise. The noise affects the detections for position, rotation, and area. By modeling the noise with Gaussian Distribution and also accounting for the latency of the vision system, the movement of the ball and robots can be reflected on the simulator concisely. In addition, developers can also implement their high-level features such as robots that can be dragged around on the field on the top of the SSL-simulation-protocol. Trajectory planning for the ER-force is also well-developed, making it accurate and realistic. Like grSim, ER-force is also supported on all platforms. Apart from all of the nice things that the ER-force has, the modeling of the dribbling or kicking mechanism is not as good as grSim. According to the ER-force developers, modeling the mechanism as a rigid body and 'gluing' the ball to it meets their requirements. This is far worse than grSim's method of adding a frictional force that does not exist in reality to simulate the mechanism that prevents the ball from moving sideways.

NVIDIA PhysX 4 is a powerful open-source industrial-level real-world engine. The key advantage of using PhysX as a simulator is that we do not have to worry about the inaccuracy of simulating omni-drive movements and dribbling mechanisms. The engine itself can guarantee a very realistic simulated environment that is far beyond the requirements of the RoboCup SSL competition. This would boost our software development speed by a lot if we know how to use it properly. Hence, the only problem this simulator has is the steep learning curve. We not only have to model everything from scratch, but also have to think of a way of making it subject to the SSL-Simulation-protocol. Currently, only the CMDragons team is using PhysX as their simulator. If we were to do the same, it would take either too much time or too much workload for team members to code it regarding the scope of this project, even if we only use existing APIs. In addition, we are completely on our own if the CMDragons team does not wish to help us.

CHAPTER 3 DESIGN AND DEVELOPMENT

We conducted the design and development of the mechanical, electrical, and software systems in our sub-teams. The following sections provide a description of the overall system, design concepts and prototypes, and details of the development of each subsystem.

3.1 SYSTEM OVERVIEW

3.1.1 MECHANICAL OVERVIEW

The system is designed to be capable of linear translation as well as rotation around a carpeted playing surface as per RoboCup regulation. This is achieved via four brushless DC motors paired with omnidirectional wheels. The robot is also equipped with a ball control system designed to interact with and manipulate the ball. The core functionalities of this module are to dribble the ball by rotating a cylinder that keeps the rotation of the ball in phase with the robot's motion, chip the ball by sweeping a plate at an angle under the ball, and kick the ball by thrusting a bar into it.

Per the official RoboCup Small Size League Rules, the robot must:

- Fit inside a 0.18 meters wide and 0.15 meters high cylinder at any point in time.
- Be fully autonomous
- Not pose danger to itself, another robot, or humans.
- Not damage or modify the ball or the field.

The dribbling device must:

- Not elevate the ball from the ground.
- Not take full control of the ball by removing all of its degrees of freedom.
- Not cover more than 20% of the ball surface

3.1.2 ECE System Overview

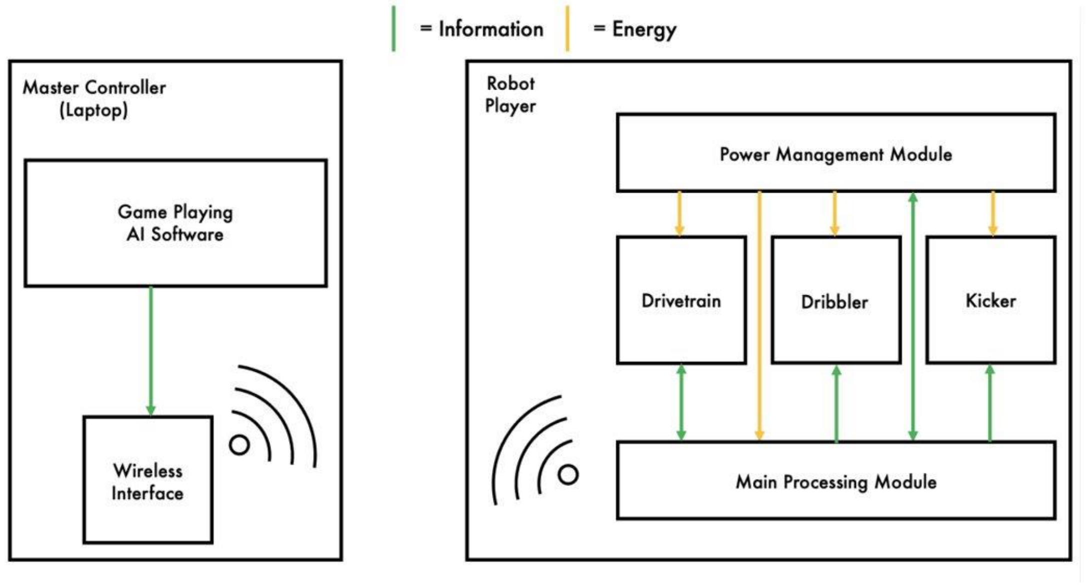


FIGURE 3.1 SYSTEM ARCHITECTURE BLOCK DIAGRAM

The system in Figure 3.1 is the architecture diagram used when we initially started to break our work into a bunch of different components on and off robot. Outside of the AI system, we have the router and wireless interface that allows us to communicate with all our robots. On the robot, we broke it down into modules of power management, kicker, dribbler, drivetrain and the main processing module. Next, we broke down each module into their individual requirements to serve their purpose. Drivetrain must be to able drive the robot at the desired high speeds, precisely. The dribbler module’s purpose is to essentially develop the electronics to interface and be able to drive the ball, by keeping it within the chassis as the soccer robot moves. The kicker is the module with the drivers needed to actuate the kicker and chipper mechanisms. Finally, the main processing module contains the wireless communication, robot identification, debug circuitry, clocks, and the main processor.

3.1.2.1 SYSTEM REQUIREMENTS

The system requirements were broken down by the power module above which would be done by the tasks based on our extensive research of the TDP's and other circuitry of similar robots. To be more thorough than the previous section, first, the master controller module consists of the

AI. The master control is handled purely by the software. Next, the physical connection between the hardware and the software is handled by a router system which consists of PCB design on the circuit boards, and essentially the requirements to follow in this section were the drivers necessary to communicate with multiple robots concurrently, and to be able to accurately communicate messages to and from the AI system, so that we can properly manage the robots on the field.

The power management module was broken down into the power systems and the battery management sub-system. The power system was required to power an assortment of voltage levels ranging from 3.3 volts to 5 volts, and 22.2 volts. This needed to be done in a manner that reduces noise to a minimum, and to be able to prevent burnouts from high voltage changes. The battery management module performs the function of communicating with the AI to provide insights on the state of the robots internally.

The goal of the drivetrain was to develop a motor driver that can control the robot and work with a PID controller to have precise and fast movement that works well under load and is responsive to step inputs. The dribbler builds on top of the drivetrain as much of the circuitry is reused. The main difference in circuitry between the drivetrain and dribbler module is that we do not need a closed feedback loop to control the ball well. However, there is still some outside math that needs to be done for the robot to maintain possession of the golf ball while driving.

The kicker is the final portion of the system of the robot. Its system requirements are first, fine-tuned control over operating a solenoid at the speeds required that needed by the software to move the golf ball. It also requires being monitorable during operation. To perform such a task, we need to be able to have control over the amount the capacitors are charged and the rate at which they can discharge at the voltage and current requirements needed. This must occur without causing too much thermal interruption to the other components that exist within the kicking system (both electrical and mechanical), and outside of the kicking system.

3.1.3 System Overview of Software Architecture

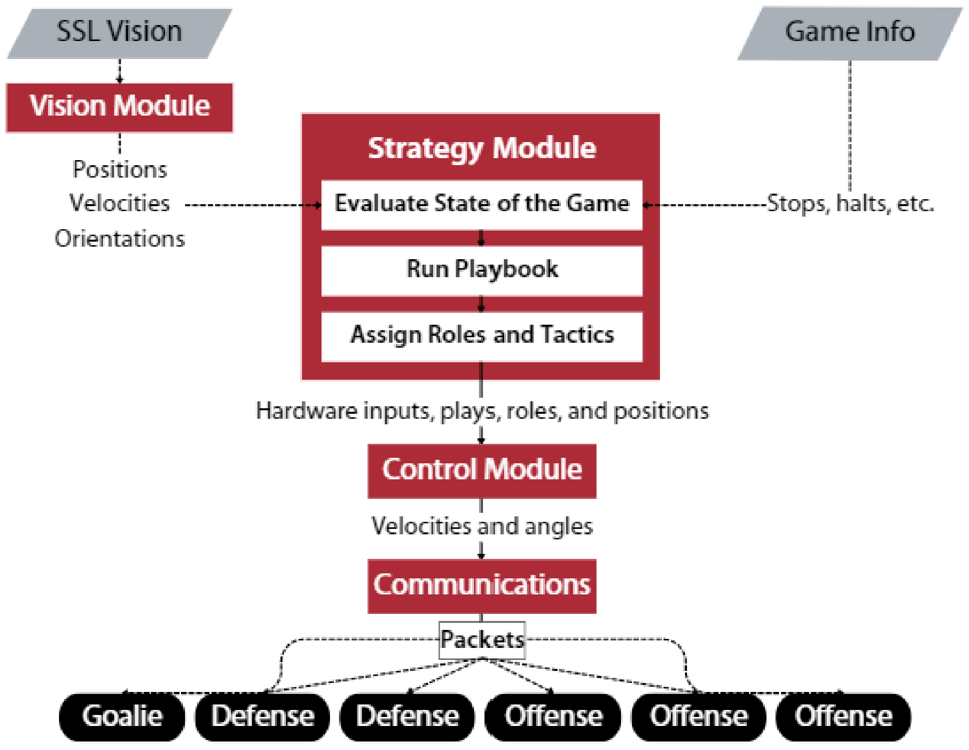


FIGURE 3.2 SOFTWARE ARCHITECTURE OVERVIEW

As seen in Figure 3.2 above, we designed an architecture that takes SSL Vision and the Game Information as inputs. SSL Vision is fed into our vision module, which outputs positions, velocities, and orientations of both teams as well as the ball. Our strategy module handles the evaluation of the game state, selecting a play from our playbook, and assigning roles and tactics. Our control module turns this information into velocities and angles to send to the robots through our communications system.

To design our code structure, we split up the work into two main modules: navigation and strategy. Navigation had to do with moving a robot to a point, and motion planning given two points and a set of obstacles. The strategy module aims to tackle decision making in gameplay, such as which robot should be assigned which position, analyzing the field, calculating probabilities, and selecting plays. Our overall UML diagram is shown in Figure 3.3 below.

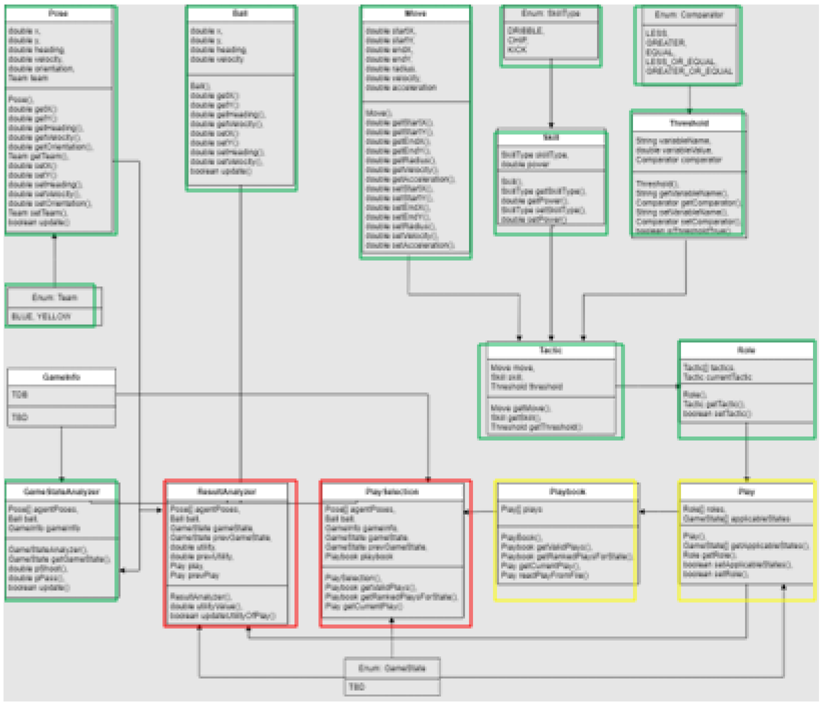


FIGURE 3.3 THE UML DIAGRAM OF ALL CLASSES IMPLEMENTED IN OUR FINAL DESIGN.

3.2 CONCEPTUAL DESIGNS AND PROTOTYPING/MODELING FEASIBILITY

The following sections outline the sub-teams' conceptual designs. We begin with a progression of our mechanical CAD designs as well as the modeled feasibility of the design.

3.2.1 CAD V1.0

To start off we used our initial design for the CAD as shown in the figure below. This design was created from all of the efforts of our research in A term and was the proposed product we planned to produce. From the figure you can see the top and bottom plate which were planned to be manufactures using a mill to cut from solid aluminum stock but also four aluminum L brackets which were to be secured to the top plate of the robot and would hang down. These L brackets were to be cut from aluminum blocks using a mill just like the top and bottom plates. Along with these we had also discussed the kicking mechanisms which included a main kicker base and within it was the dribbler, kicker, and chipper. These can be seen at the front of the design and were to be manufactured in Washburn as well and cut from aluminum stock or 3D printed.

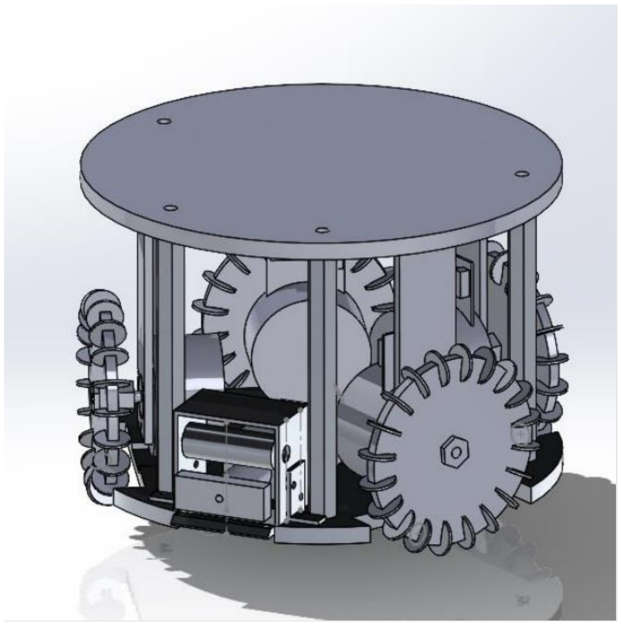


FIGURE 3.1 SOLIDWORKS CAD MODEL ONE

3.2.2 CAD V2.0

After multiple weeks in phase two the team realized that the design, we made needed to be tweaked so that manufacturability could be easier and plausible for the team. Looking below at Figure 25 you can see the changes between this new updated CAD and our previous version. To start off the base change that can be seen is the obvious color changes on the model itself. We color coded the different parts of the robot so that they could easily be differentiated. To start the green box on the top of the robot is a “black box” of the PCB meaning it is just a place holder for the actual PCB that had yet to be designed at this point in time. Under the top plate you can see four bronze cylinders which are the motors which connect to the obvious omnidirectional wheels present. Along with those the last color change is with the kicker mechanism. The dribbler is yellow while the kicker is red so that you can easily see the distinction between the two.

The real design change here that is to be pointed out is the wheel mounts. As seen in the figure below, the hanging L-bracket wheel mounts that were fastened to the top plate have been changed to smaller L-brackets which are now fastened to the base plate rather than the top. These brackets were changed from four-inch-tall brackets to two-inch-tall brackets because manufacturability became a large roadblock here. With the previous design we either had to manufacture the brackets or buy them and then drill the holes into them. After looking online, they did not sell four-inch tall L-brackets which left us with the need to manufacture them. After talking with Washburn supervisors, we decided that manufacturing these parts would entail purchasing very expensive aluminum stock blocks along with an arduous milling process. With this knowledge we decided to change them to smaller brackets that would be fastened onto the baseplate and ones that could be purchased online.

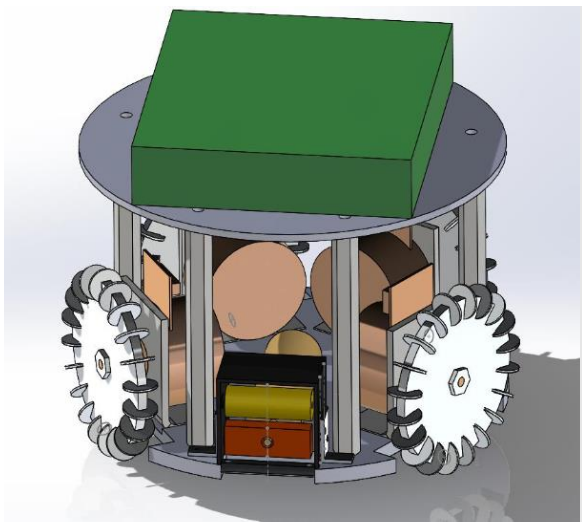


FIGURE 3.4 SOLIDWORKS CAD MODEL TWO

3.2.3 CAD V3.0

The third CAD iteration which is shown in the figures below, has the most changes made from the previous two iterations. These changes can be broken down into multiple different parts with the first being the top and base plates of the robot. The entire shape of the baseplate has been changed. The major change from the original design to the current design is the chape in which the design now holds. Before the changes the baseplate was very sharp and jagged around the edges and stock was only taken off where clearance needed to be for the robot. This left the base plate to be thicker than needed and weigh much more than it needed to. In this third CAD iteration the baseplate was changed to a much smoother and streamlined shape. This was for two different reasons with the first being all of the sharp edges smoothed out so that the mill could cut the plate much easier. The second being to save space and cut off access stock so that the baseplate would be lighter. Another change to the baseplate is a fifth hold was added to that extra support could be added to the design to make it much more rigid and durable.

The next change in this CAD iteration is the top plate of the robot. The reasons for the top plate's changes are much simpler than the change for the base plate. The top plate now has holes cut into it in the middle of the plate, the reason for this was we wanted to cut down the overall weight of the top plate along with allow more air flow to come through the plate and cool down the PCB and batteries. Heat management measures were also implemented in the form of heat sinks mounted on the baseplate near high thermal components like the two solenoids.

The standoffs used to mount the baseplates on are two inches shorter than in previous iterations. This was made possible by redesigning the ball control module that aligned the solenoids side-by-side rather than on top of one another. Combined with the cutouts on the top plate, this greatly reduced the overall weight and lowered the center of gravity of the design, leading to a faster and more agile robot.

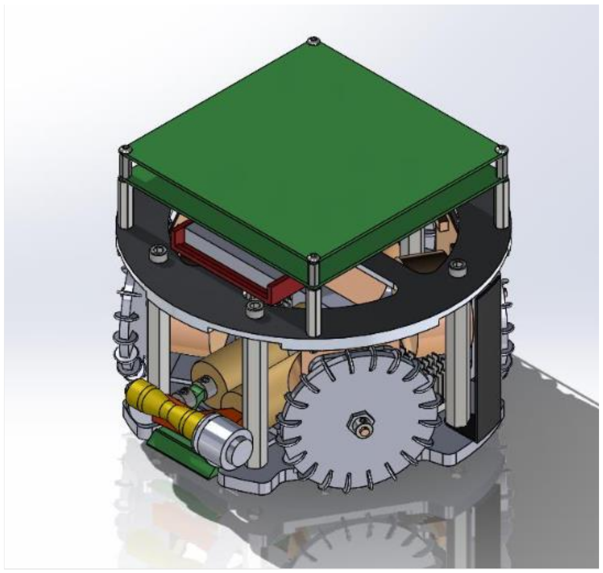


FIGURE 3.5 SOLIDWORKS CAD MODEL THREE ISO VIEW

Below you can see a top-down view of the CAD in the figure below. This top-down view truly shows the shape of the bottom plate along with the locations of all the inner working parts. You can see at the top of the figure is the kicking mechanism and you can also see that at the center of the robot are the kicking mechanism solenoids which will sit in the robot side by side to each other.

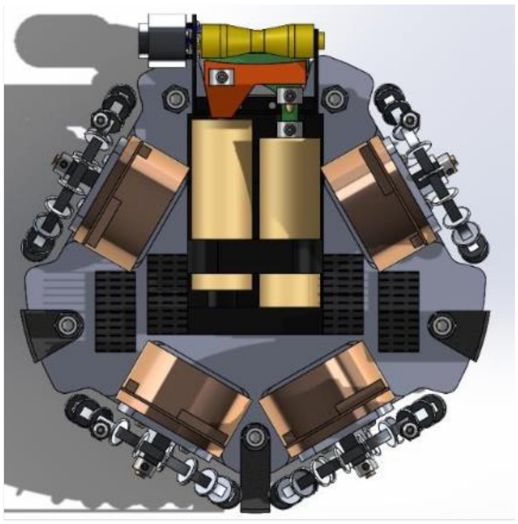


FIGURE 3.6 SOLIDWORKS CAD MODEL THREE TOP VIEW

While in the above figure you are able to get a look at where the solenoids sit in the robot, you are unable to see how they are mounted. In the figure below, you can see a close up look at the solenoid holder. This piece has been especially designed so that the solenoids can be placed directly in them and held perfectly in place on the baseplate of the robot.

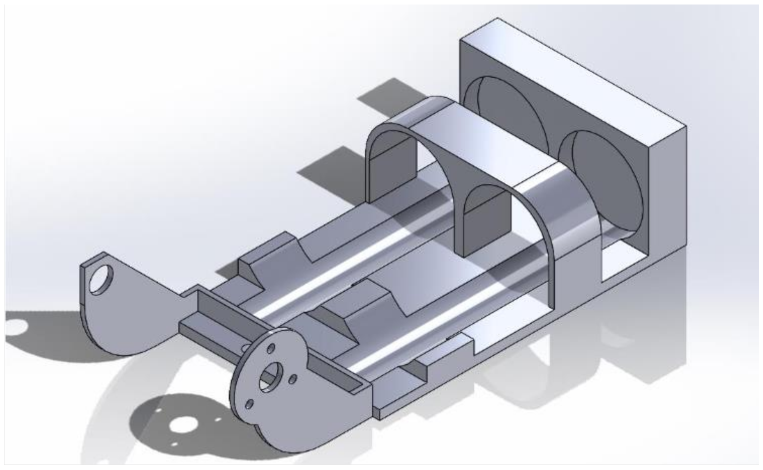


FIGURE 3.7 SOLIDWORKS CAD MODEL THREE SOLENOID HOLDER

The kicker mechanism which is in the front of the robot was redesigned from the original design to better fit the space constraints and manufacturability along with how it will perform. The first part of this mechanism is the dribbler which is shown below in the figure below and is colored yellow in the CAD design. This dribbler was specially designed with the hourglass shape so that the ball we be held directly in the middle of it and be centered perfectly for the kicker and chipper.

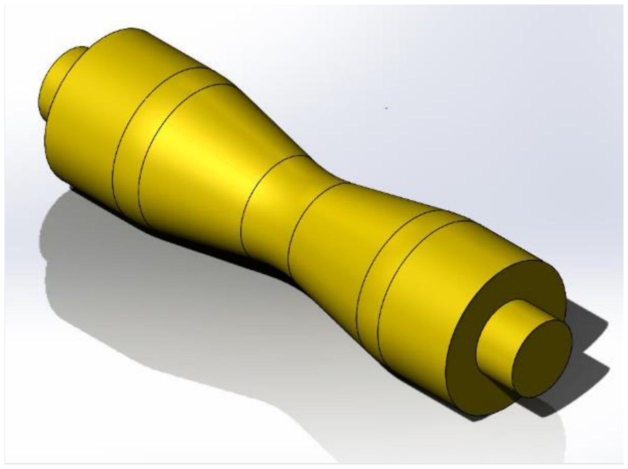


FIGURE 3.8 DRIBBLER MODEL

The next mechanism is the custom kicker in the figure below which is colored and orange-red. This kicker is different from the original flat plate because it is now curved so that the ball can find itself placed directly in the middle of the surface. When placed directly in the middle of the kickers surface we can optimize the accuracy of the kicker and in the program for more accurate results in testing and competition.

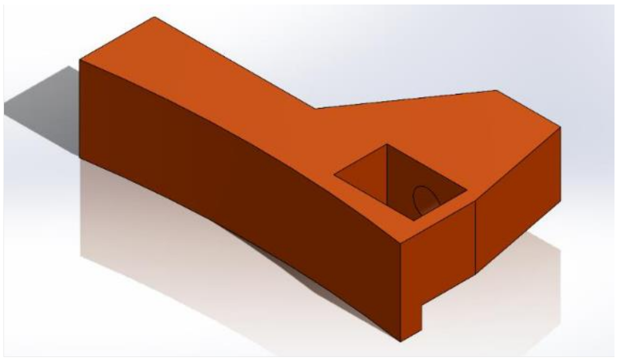


FIGURE 3.9 KICKER MODEL

The final mechanism of the kicker module is the chipper which is shown in the figure below and is colored green in the CAD. This design is the most dramatically different from the original. We changed the motion of the chipper from a swinging motion to a now horizontal translation. Using a push type solenoid, the chipper will be pushed forwards and the angled plate on the bottom that will strike the ball will hit the ball at an angle and send it flying into the air rather than on the ground. This was the decision because of its ease creation in terms of solenoids and connection pieces.

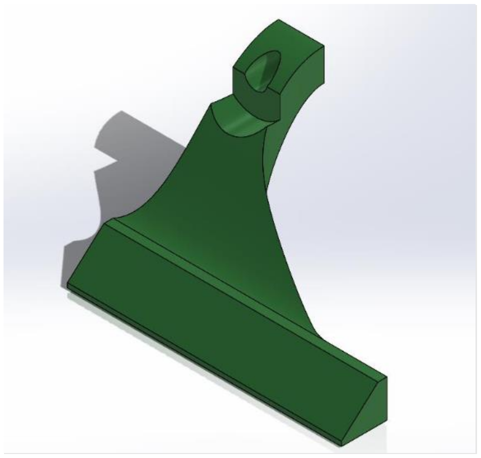


FIGURE 3.10 CHIPPER MODEL

3.2.4 CAD V3.1

The updates to the CAD in this iteration mostly came in the form of ball control module improvements. These modifications to the design predominantly came from structural integrity issues (documented in section 3.2.5 Collision Analysis) in the motor and solenoid mount superstructure stemming from material and spatial constraints. The dribbler assembly mounting structure was particularly flimsy and would frequently deform plastically along the intersection between it and the solenoid housing structure, resulting in excess friction caused by the dribbler bar rubbing against the opposing mounting hole. The drive motor adjacent to the ball control assembly would also rub against one of the solenoids due to poor tolerancing of mounting holes in the baseplate.

With the updated superstructure came slight modifications to the dribbler, kicker, and chipper ball control components, which were also simplified for ease of manufacturing purposes. The dribbler is smaller in diameter in this iteration to account for the new placement of the dribbler motor gear design, and now features a knob at one end to prevent it from sliding out of the mount during operation. The kicker is also thinner to make room for the new dribbler assembly and has integrated mounting screw holes. The chipper is significantly less complex in geometry, with a less rounded, sweeping design in favor of a simpler to manufacture block configuration. The stem of the chipper that attaches to the solenoid shaft was extended to make the shipping face as close to the ground as possible to create better angled contact with the ball and was similarly fitted with set screw holes.

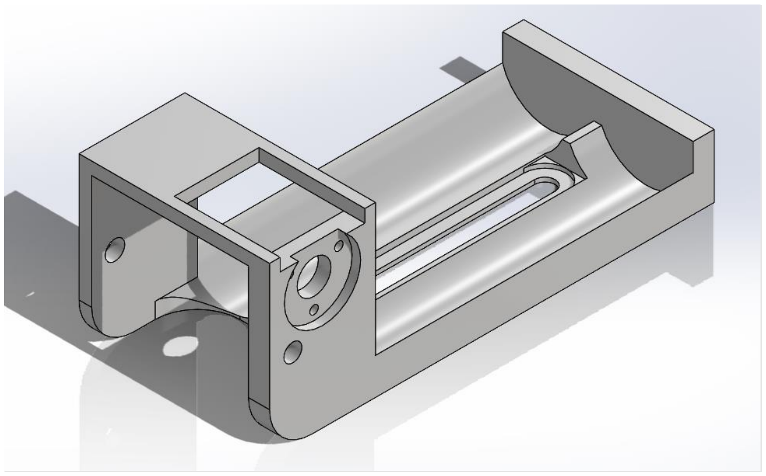


FIGURE 3.11 UPDATED BALL CONTROL MOUNT

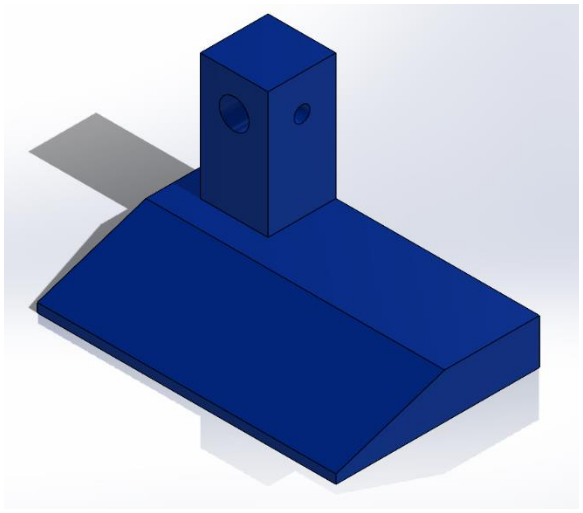


FIGURE 3.12 SIMPLIFIED CHIPPER

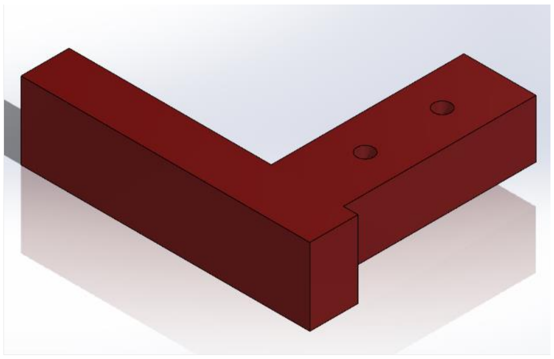


FIGURE 3.13 SIMPLIFIED KICKER

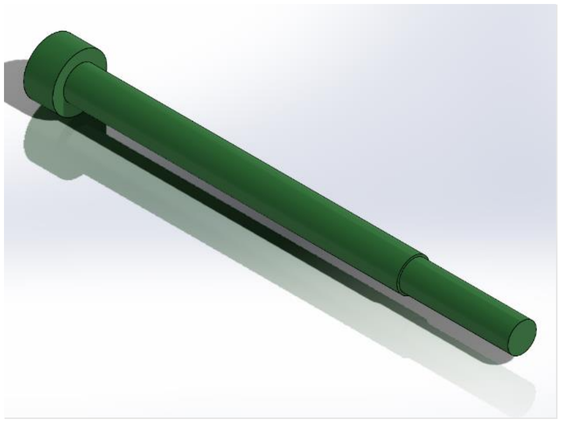


FIGURE 3.14 SIMPLIFIED DRIBBLER

Furthermore, the team experimented with alternative rapid prototyping methods and different materials for these components, particularly the new mounting superstructure. Early prototypes were printed in the Innovation Studio's 3D Printing Lab in PLA. Later iterations of the design were printed using the Higgins Rapid Prototyping Lab in rigid resin materials. This proved to significantly increase the strength of the structure and resulted in better fitment of components such as the dribbler motor and solenoids due to the higher precision tolerance capabilities of the more advanced printers.

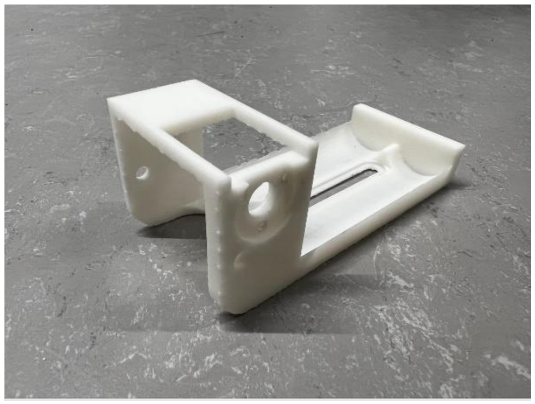


FIGURE 3.15 RIGID RESIN BALL CONTROL MOUNT

The dribbler bar was determined to be acceptable when printed using consumer grade 3D printers in PLA plastic. Further improvements of the dribbler can be made in the form of a rubber sheath or added ribbing based on future testing. The kicker was intended to be machined in aluminum, however, due to personnel complications, it was determined that an acceptable stop-gap solution would be to print it out of PLA as well and attach a small plate to the kicking surface to avoid chipping. These improvements will develop as testing dictates and permits. Finally, the chipper was determined to require the strength of a metal material, and thus needed to be machined. This fabrication was outsourced to PCBWay and the part was received at the end of C-Term.

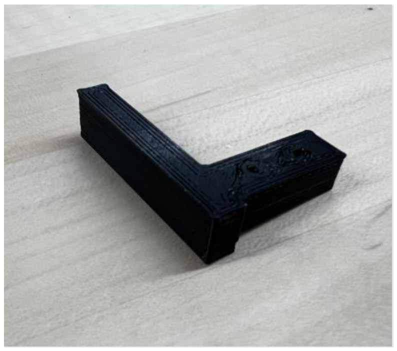


FIGURE 3.16 NEW KICKER

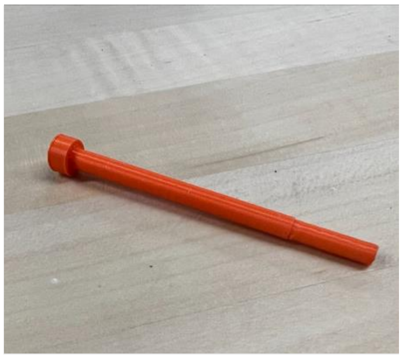


FIGURE 3.17 NEW DRIBBLER

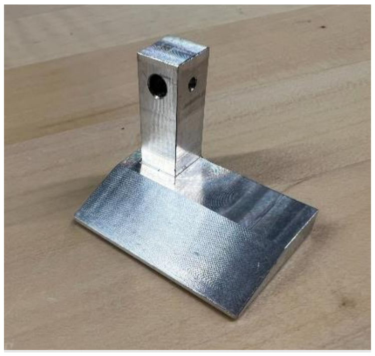


FIGURE 3.18 NEW CHIPPER

3.2.5 Collision Analysis

Collision load analysis was conducted in this version of the robot on newly developed and previously unanalyzed components such as the ball control mount for verification purposes to demonstrate that the robot would not fail structurally in the event of a collision with a robot of 40kg (medium sized league upper limit) at increasing speeds up to a maximum of 15 m/s. In these simulations, the structure is fixed in all six degrees of freedom at its base where it would be mounted to the baseplate and simulated impacts are applied normal to the side face of the mount at its exposed corner and normal to the front face of the mount to simulate a direct frontal and side impact. This analysis was conducted to observe the structure's potential behavior under a worst-case impact condition in which a competitor collides with the robot at a theoretical weak point. The result of a minimum factor of safety of 1.4 and 1.9 for the resin material indicates that this redesigned component is acceptably prepared for such an unrealistic loading scenario.

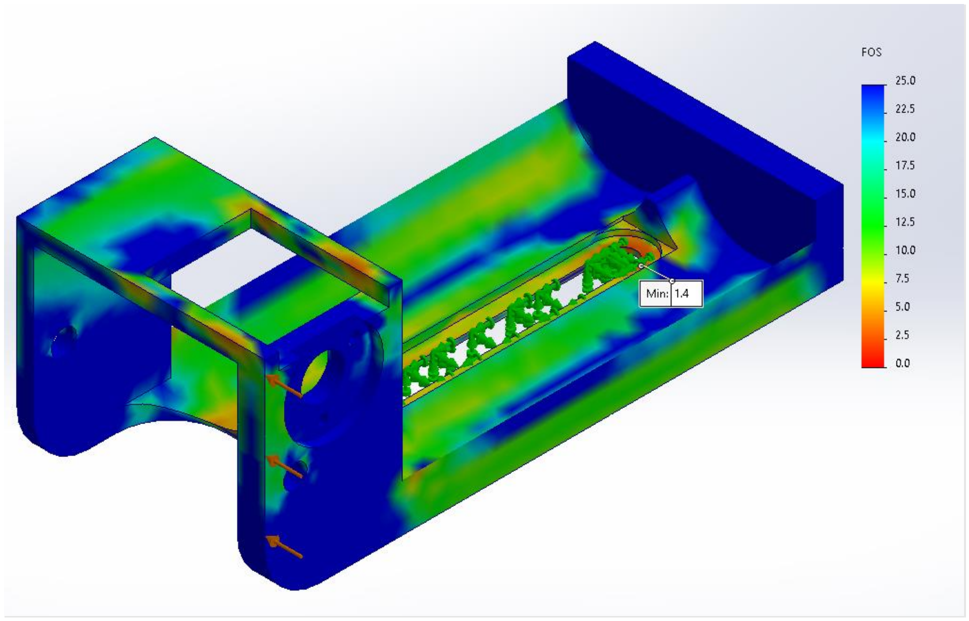


Figure 3.19 Side Collision Simulation - Factor of Safety

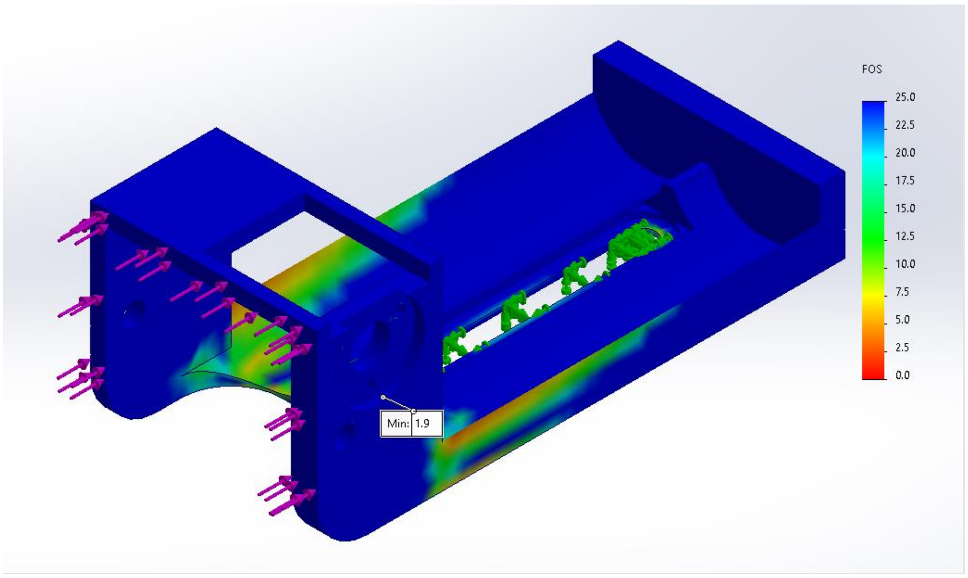


FIGURE 3.20 FRONTAL COLLISION SIMULATION - FACTOR OF SAFETY

3.2.6 ECE CONCEPTUAL DESIGN

Our conceptual design involved taking each module pictured in the system architecture in figure 3.1 and then using the development board to prototype the circuits we needed to prototype each module or sub-module. We implemented the system architecture into its pieces by first bread-boarding them, where we would also start with the initial design of drivers. From there, we worked on the schematic design and the PCB layout design.

3.2.6.1 DEVELOPMENT BOARD

To validate circuits and components that would later be embedded in the custom PCBs, the team would need a development board. The development board needed to be equipped with a plethora of IO, the peripherals that would be used on the processor board, and an identical processor to the ones that would be placed in the processor board, as there would be no need to change firmware. Due to supply issues, we couldn’t always obtain the exact variant of a chip and had to settle for variants that were almost identical, which was reflected in the firmware.

The board we settled on working with was the SAMV71 Xplained Ultra Evaluation Board. It features the 32-bit ARM Cortex-M7 Processor, an on-board embedded debugger, and the additional peripherals to extend the board’s features. The Xplained Ultra evaluation board is compatible with Microchip Studio’s Atmel Start API, which makes creating drivers much simpler. Atmel Start allows the user to select the board, processor, desired driver/middleware type, and configuration related criteria, and gives a baseline for the requested driver code.

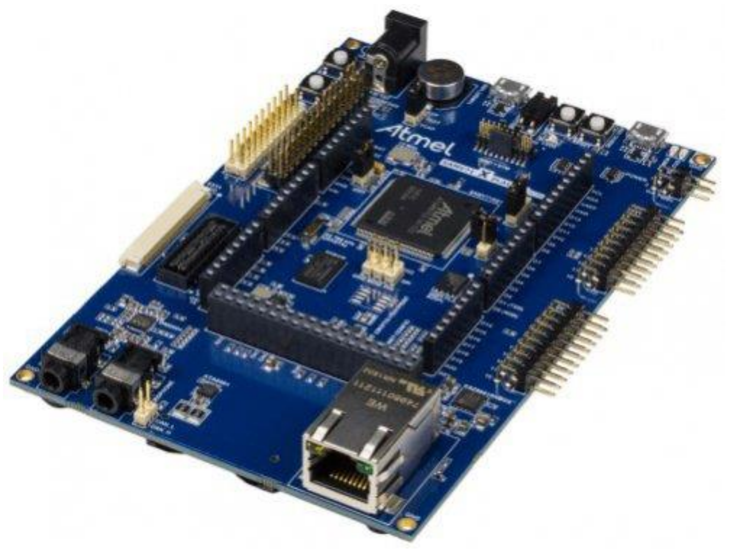


FIGURE 3.21 SAMV71 XPLAINED ULTRA EVALUATION BOARD

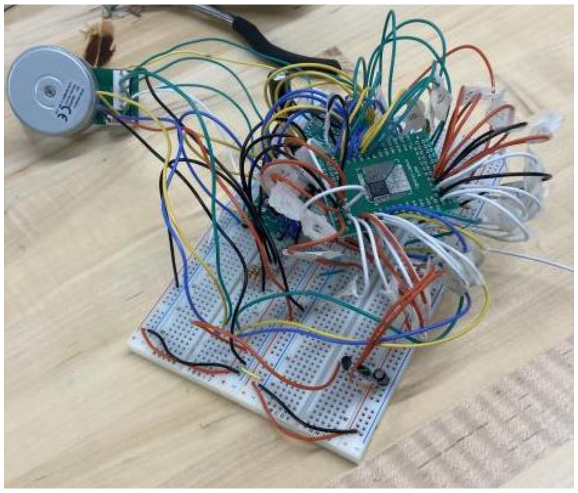


FIGURE 3.22 MOTOR CONTROLLER PROTOTYPE CIRCUIT

3.2.7 ECE HARDWARE FUNCTIONALITY

The following sections describe calculations and designs made to create circuity necessary for robot functionality.

3.2.7.1 SOLENOIDS

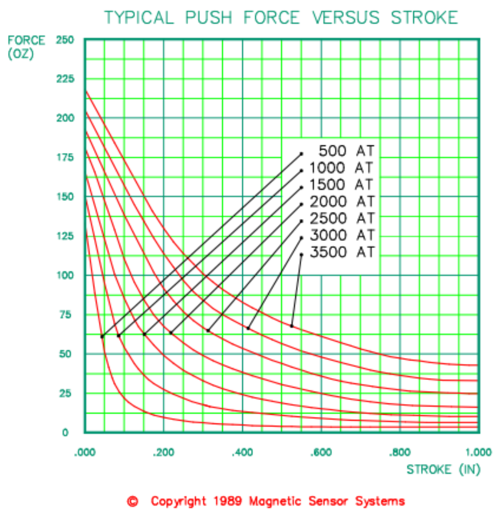


FIGURE 3.23 FORCE VS STROKE DISTANCE OF SOLENOID

Figure 3.23 shows the force curves of each of the constraints are based on the pulse. Calculations are based off the curve for the 3500 ampere turns, and as this is run at 20.0 volts, which gives an estimated current draw of 1.6 Amps after scaling. With a force of 20.85 newtons launching the golf ball at approximately 0.45 meters per second, the time of contact was calculated to be 0.0010865 seconds, which is equivalent to 1/936 seconds. Then, with the 20 newtons, a predictable model can be derived to give forces for all possible amperages. These can be characterized by the following equations seen in solenoid section of the appendix.

3.2.7.2 PROCESSOR BOARD

The Processor Board was designed by Noah Page. It handles all functionality of all boards in the PCB stack. Please refer to Noah's published report for an in-depth analysis. We tested and verified its design in the term following his departure from the team.

3.2.7.3 POWER BOARD

The power board contains all voltage supplies and OBO to get the power requirements needed as it streams off 22.2 volts to the motors, 12 volts to the kicking board, 5 volts to all ICs that require it and 3.3 volts to the processors. Next there are fuses on board to limit the amount of current drawn by systems which allows 30 amps to the motors and 10 amps to the kicker board. This limits overcurrent draw and offers overcurrent protection on the system such as the wheels and the capacitor charger, like in cases when it possibly short circuits or malfunctions. This protects against overcurrent draw and damaging the ICs in the process.

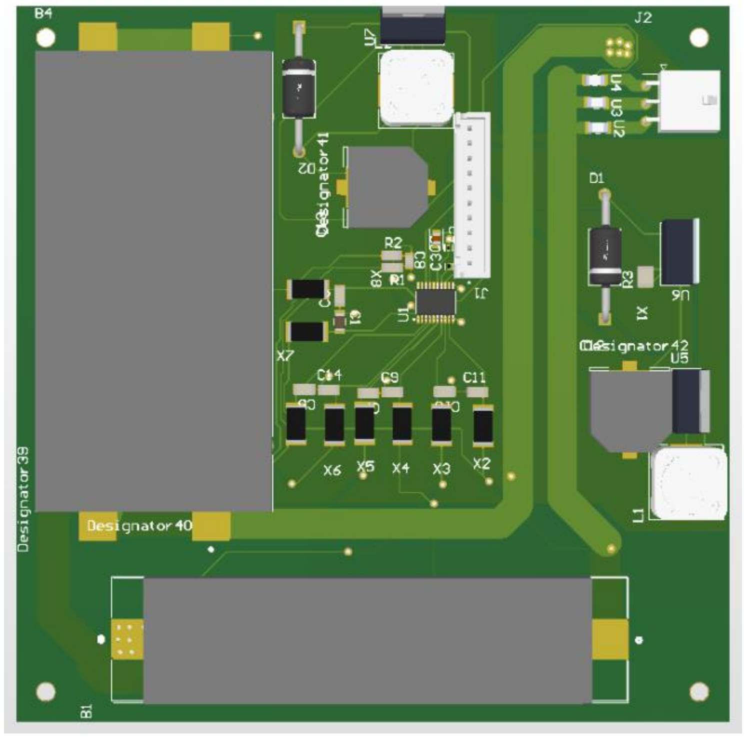


FIGURE 3.24 ALTIUM GENERATED PNG OF POWER BOARD

3.2.7.4 KICKER BOARD

The charging circuitry pictured in Figure 3.25 is used to run the LT3750, a chip specked out due to functional requirements. In the current configuration there are a few equations that justify the alternative behavior which comes with the RV out and RCM resistors. The current sensing resistors between the RV out and RCM resistors source and the leave of the transistors are 0.08 ohms. This alters the minimum current, whereas the 60.4K ohms in the RV out alters the total voltage out of the chip to 250V. The 40K32 changes the minimum pulse duration. From that, the efficiency is derived based on the capacitor sizes. The math for the equations characterizing this directly can be found in the appendix. This configuration runs off 7 amps and a maximum current charge time of 9 seconds to reach 250 volts. The transformer allows for a maximum of 10 amps to be pulsed through it, which is not reached. The S3J directs the current towards the capacitors allowing the configuration in the solenoid below in figure Zeta. The capacitor charging chip charges by setting kick charge high. On the processor it requires about 0.8 volts to be turned on, so using the normal 3.3 Volt DIO of the SamS70 chip works. The chip can be directly driven as its impedance is isolated from the rest of the chip and power amplification.

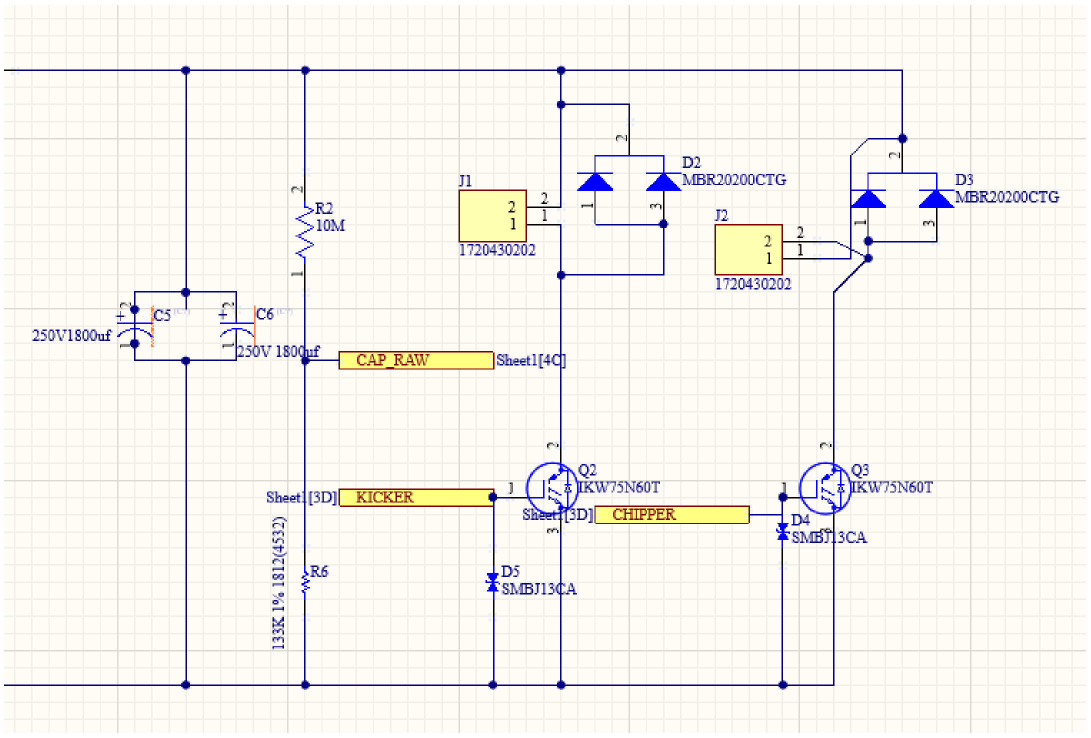


FIGURE 3.25 ALTIUM SCHEMATIC OF SOLENOID RELEASE CIRCUITRY

In the capacitor charger, the most interesting part is the IGBTs. The IGBT is essentially a cross between a power BJT and a power MOSFET that has the responsiveness of a BJT, but also has the drive properties of a MOSFET, essentially allowing to control the amperage across by changing the voltage. It has the switching speed of the BJT which is better than what would normally be considered of a power MOSFET of equal ratings. One of the drawbacks of these chips is that it's recommended to use an IGBT driver as it has special properties such as if the voltage is ever within range there's a dead zone that's quite large. Driving this with essentially anything under 5 volts would draw half an amp of current at most and so it is recommended to use a PWM signal to drive the transistor. This can be done by routing the PWM signal into a IGBT driver that raises the voltage to 12 volts allowing us to ignore the low voltage behavior of the IGBT. Then after analyzing the data sheet and through some testing we came to find out that due to the unique properties of the IGBT we needed an antiparallel diode.

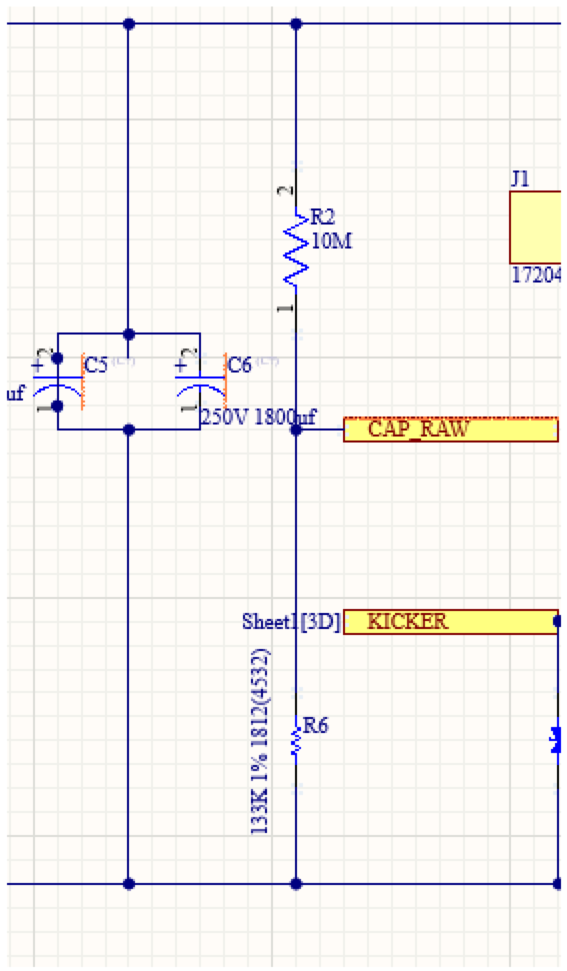


FIGURE 3.26 VOLTAGE DIVIDER FOR CURRENT MEASUREMENT

After reading the data sheet and computing values, which can be found in the appendix, and selecting a few options, the best performance was achieved with the SMBJ13CA diode. Not having the diode led to inconsistent behaviors and the damaging of a transistor. The diode needs to be tied to the emitter and gate to have any influence and generate the proper behavior from the IGBT. To choose the PWM period it is recommended to look at the image above for all possible characterizations of the transistor, as well as the time to discharge the capacitor. The solenoid has a resistance of 4 ohms meaning that it allows up to 62.5 amps to be driven through it. This empties the capacitor in .016 seconds. Combining that with the response rate of the FAN3229TMX IGBT driver, a period of 10KHz was decided upon. For reference, in a closed circuit the transistor has a turn on delay of 32 ns and turn off delay of 363 ns. As this turn on delay is 0.3 percent of the period and the turn off delay 363ns, these are not negligible amounts but modellable in the firmware. The diode is in the place of an isolator to allow for any charge on the solenoid activation that ends up being stored in it to be emptied safely, as an inductors charge tend to be stored. This prevents drawbacks to the system and acts as a safety barrier. Please find the relevant calculation for the kicker circuit in the Appendix.

Figures 3.26 and 3.27 describe the voltage divider in op-amp RC filter used to monitor the current on the kicker board as this provides an inference for the current voltage rating and the amount of charge after a kick has been fired. This is hooked up to an ADC on the processor board. This brings the 250-volt range down to 3.3 volts, where 3.3 represents a maximum of 251 volts. Then, we use an RC filter to smooth out the signal. This reduces the noise coming from the capacitors, meaning when the solenoid fires it slowly decreases to not get an immediate jump. Another portion is used to prevent the large amount of noise that happens when firing.

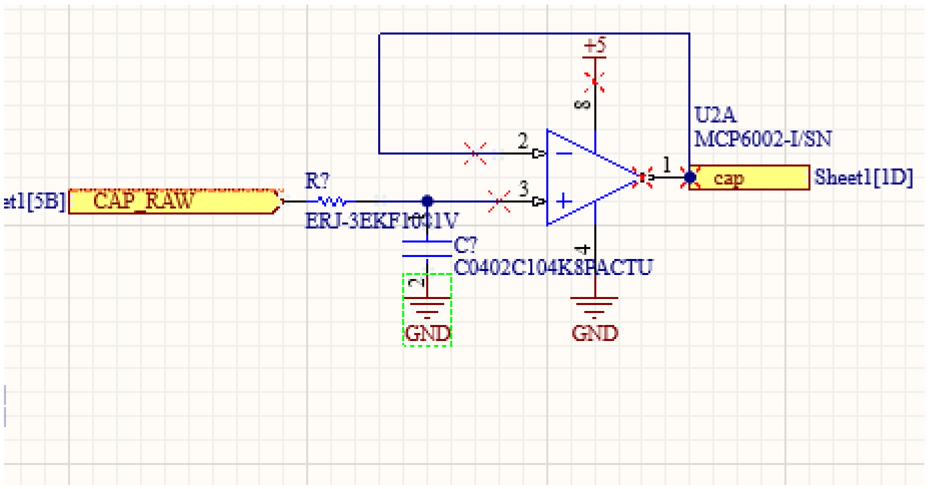


FIGURE 3.27 OP-AMP WITH RC FILTER FOR MEASURING CHARGE OF KICKER BOARD

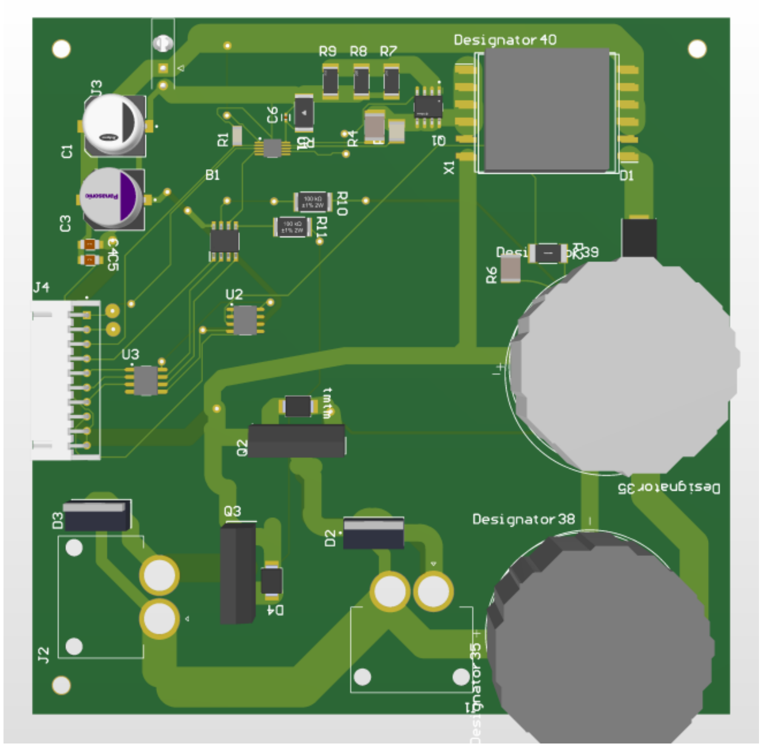


FIGURE 3.28 ALTIUM GENERATED PNG OF KICKER BOARD

3.3 FIRMWARE AND DRIVERS

An integral portion of the overall project was developing the firmware drivers to abstract communication with the hardware from the Computer Science sub-team such that their focus could shift to tactics and gameplay. These drivers were started in Atmel Studio and fleshed out in Microchip Studio's IDE. The drivers cover a variety of low-level functions and are managed by a main firmware loop that processes commands sent from the master device.

The two largest challenges faced in this stage of development were scarcity of documentation of such software and writing on-board navigation for the robots. Often, the best solution to problems with no reference information was to sift through thousands of lines of auto-generated driver example code, or to scour decade-old forums. For implementing navigation, PID control has been transplanted to a lower level in terms of code structure in order to solve this problem because the master device is incapable of sending such a high volume of datagrams at a high frequency to 6 total robots.

3.3.1 Main Loop

The Main Loop of the firmware delegates which necessary tasks must be performed depending on the commands from the master device. After initializing all relevant hardware in the first loop iteration, the main loop polls for a Noah's Packet Protocol datagram. If a datagram is available, the firmware parses it and sets certain flags and stores data as received by the datagram to be dealt with using the driver code. The main loop then determines if it is time to perform new PID calculations for the motor controllers depending on a flag set by a 10 millisecond timer. These calculations are government by the velocity outputs of the navigation code. The main loop then then performs all other relevant tasks given by the NPP, such as charging kicking solenoids, reading power cell voltages, and so on.

3.3.2 Motor Controller

The motor controller driver was written using a PID controller to maintain all four motors simultaneously to obtain the desired velocities of each. The motor controller receives target velocities to produce the segmented path calculated by the RRT algorithm (rapidly exploring random trees). The controller operates at 100 Hz and utilizes both proportional and integral terms to obtain a velocity between positive and negative 10 meters per second.

Every 10 milliseconds, the controller calculates the differences in target and current speeds using the encoder driver, then updates the effort signals accordingly. Since the robot uses electronic speed controllers (ESC) that consume pulse-position modulation input rather than pulse-width modulation input, the motor controllers use the length of a PWM signal rather than its duty cycle. Furthermore, the ESCs do not center the bi-directional motor efforts at a logical 50% duty cycle of the PWM signal. Therefore, the error in speed needed to be mapped to the error in PWM signal, centered at an experimentally determined 0% motor effort value. This linear mapping of speed to PWM is treated as the proportional term of the PID control. It was determined experimentally by finding the PWM dead-band zone of the motor, the PWM zero value, and the PWM signals that produced the closest values to +/- 10 meters per second. Tabulated below are these PWM values and their corresponding velocities.

Table 3.1 Velocity to PWM Mapping

	Measurement ID				
	PWM Minimum	Dead-band Minimum	Zero	Dead-Band Maximum	PWM Maximum

PWM Value	4635	4854	4908	4962	4962
Velocity (m/s)	-10	0	0	0	10

Using the PWM values in the table above, the dead-band correction is +/- 54 (depending on the direction of motion), the zero value is 4908, and the slope to linearly map these quantities is calculated to be 27.25. In algebraic terms, (PWM Error) = (Zero Value) ± (Dead-band) ± KP*(Velocity Error). In numerical terms, this correction is (PWM Error) = 4908 ± 54 ± 27.25*(Velocity Error).

The integral term is calculated as the total sum of velocity error *before* it is mapped to PWM error, with a KI value of 1.0.

3.3.3 Encoders

The encoder driver is responsible for maintaining the encoder count for every motor on the agent. The encoders have quadrature outputs, which are referred to as A and B output signals. Signal A leads signal B with a 90-degree phase difference. A change in these signals trigger interrupts which either incremented or decremented the encoder counts, depending on the current and previous values of A and B. In the below table, each state value, written in blue text, is comprised of two bits. The leftmost bit refers to the signal from channel A, and the rightmost bit refers to the signal from channel B.

Table 3.2 Quadrature Encoder Logic Table

		New State			
Old State		**00**	**01**	**10**	**11**
	00	0	-1	+1	X
	01	+1	0	X	-1
	10	-1	X	0	+1
	11	X	+1	-1	0

At interrupt time, the program compares these two states and determines how to handle the count:

- 0 – no rotation, do not change count.
- +1 – counterclockwise rotation, increment by 1.
- -1 – clockwise rotation, decrement by 1.
- X – not a possible signal transition, handle error accordingly.

Since the processor was having difficulties with higher encoder resolutions, the encoders are configured to 256 PPR (pulses per resolution), or 1.4 degrees of the shaft per pulse. The maximum resolution for the purchased encoders is 2048, or 0.176 degrees of the shaft per pulse. Assuming a maximum speed of 10m/s and an encoder resolution of 256, the number of interrupts per second is: (10m/s) * (rev/π*0.004 m) * (256 pulses/rev) * 4 motors = 814,873 pulses/s = 814,873 Hz.

3.3.4 PWM

This driver is responsible for five PWM signals utilized on the ATSAMS70N20 processor. Four of these signals are dedicated to operating the four drive motors on the robot. The fifth signal handles the speed of the dribbler motor. All signals provided by the PWM feature of the processor are sent to electronic speed controllers (ESCs) in order to use the three phase motors. The hardware dividers on all channels were set to 128 and the PWM signals were set to a period of 5856 pulses. This resulted in a desired PWM frequency of 200 Hz on all channels. To control the duty cycle of each signal, the number of pulses in a period would be adjusted. The equation for duty cycle is given as: Duty Cycle = Pulses/5856.

3.3.5 Battery Management

The battery management driver was written to work with the BQ76925 analog front end. The BQ76925 is, in essence, a multiplexer used to read the voltage, current, and temperature for 3 to 6 series lithium-ion cells. For the project's purposes, it was only used to read the voltage of all 6 cells in the power system. The Battery Management driver utilizes the firmware's I2C instance to communicate with the BQ76925, which returns an ADC reading representing the cell's voltage. The voltage is then calculated and then evaluated.

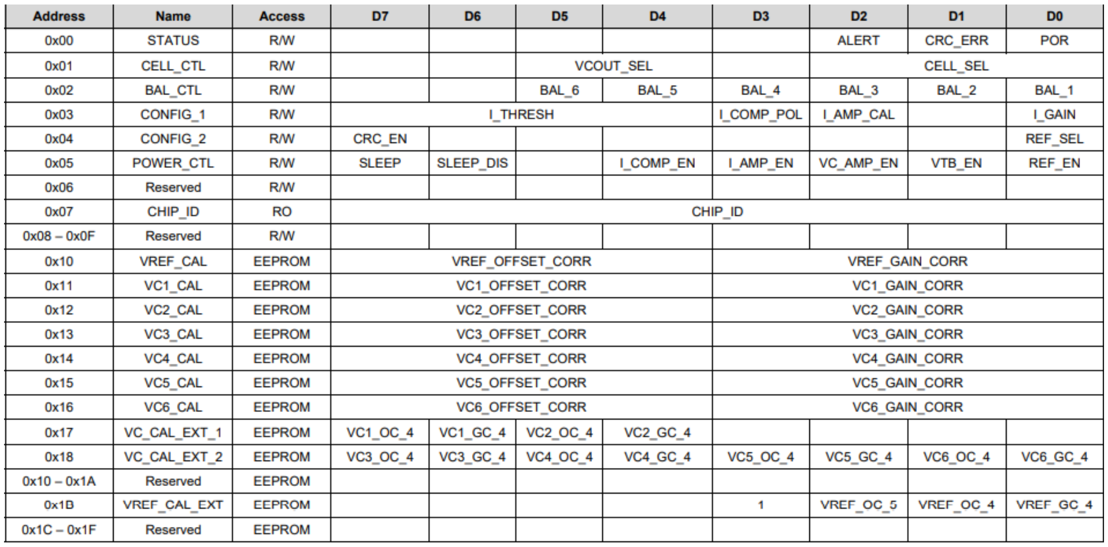
The slave device address of the BQ76925 and the device register address are combined to reduce communications overhead. Therefore, each register in the BQ76925 is treated as if it were its own 7-bit slave address (implying it were its own device), rather than a register on a device. The "device" address conforms to the below configuration:

Table 3.3 Bit-specifications for the Slave Addresses of the BQ76925

B6	B5	B4	B3	B2	B1	B0
0	1	Register Address				

Bits 5 and 6, by the manufacturer's convention, are always 1 and 0, respectively. The remaining bits [4:0] are the 5-bit slave address, specified by the datasheet, show below.

Table 3.4 BQ76825 Register Map



Address	Name	Access	D7	D6	D5	D4	D3	D2	D1	D0
0x00	STATUS	R/W						ALERT	CRC_ERR	POR
0x01	CELL_CTL	R/W			VCOUT_SEL			CELL_SEL		
0x02	BAL_CTL	R/W			BAL_6	BAL_5	BAL_4	BAL_3	BAL_2	BAL_1
0x03	CONFIG_1	R/W	I_THRESH				I_COMP_POL	I_AMP_CAL		I_GAIN
0x04	CONFIG_2	R/W	CRC_EN							REF_SEL
0x05	POWER_CTL	R/W	SLEEP	SLEEP_DIS		I_COMP_EN	I_AMP_EN	VC_AMP_EN	VTB_EN	REF_EN
0x06	Reserved	R/W								
0x07	CHIP_ID	RO	CHIP_ID							
0x08 – 0x0F	Reserved	R/W								
0x10	VREF_CAL	EEPROM	VREF_OFFSET_CORR				VREF_GAIN_CORR			
0x11	VC1_CAL	EEPROM	VC1_OFFSET_CORR				VC1_GAIN_CORR			
0x12	VC2_CAL	EEPROM	VC2_OFFSET_CORR				VC2_GAIN_CORR			
0x13	VC3_CAL	EEPROM	VC3_OFFSET_CORR				VC3_GAIN_CORR			
0x14	VC4_CAL	EEPROM	VC4_OFFSET_CORR				VC4_GAIN_CORR			
0x15	VC5_CAL	EEPROM	VC5_OFFSET_CORR				VC5_GAIN_CORR			
0x16	VC6_CAL	EEPROM	VC6_OFFSET_CORR				VC6_GAIN_CORR			
0x17	VC_CAL_EXT_1	EEPROM	VC1_OC_4	VC1_GC_4	VC2_OC_4	VC2_GC_4				
0x18	VC_CAL_EXT_2	EEPROM	VC3_OC_4	VC3_GC_4	VC4_OC_4	VC4_GC_4	VC5_OC_4	VC5_GC_4	VC6_OC_4	VC6_GC_4
0x10 – 0x1A	Reserved	EEPROM								
0x1B	VREF_CAL_EXT	EEPROM					1	VREF_OC_5	VREF_OC_4	VREF_GC_4
0x1C – 0x1F	Reserved	EEPROM								

As an example, to reference the CELL_CTL device, which controls the cell that is output, bits 6 and 5 would default to 0 and 1, as mentioned above. Then, referring to the register map, 0x01 is the address for CELL_CTL. Therefore, the remaining bits [4:0] would be filled with this value.

TABLE 3.5 EXAMPLE ADDRESS REFERENCE TO CELL_CTL ON THE BQ76925

B6	B5	B4	B3	B2	B1	B0
0	1	0	0	0	0	1

Please refer to the BQ76925 Driver Manual for more information on the functions and their descriptions for this driver.

3.3.6 ADC

The ADC driver utilizes three of ten analog-to-digital conversion (ADC) channels to read voltages on the kicker and power board. Two of the channels read the battery voltage and battery current on the power board to determine how much charge is left before the robot is inoperable. This information can be used in a game to decide to put the robot in a less taxing position if the battery is sufficiently low. The other channel utilized reads the voltage on the capacitors on the kicking board. The data provided here is used to decide whether or not the capacitors are sufficiently charged to operate the solenoids and kick or chip the golf ball at a desired velocity.

3.3.7 TIMERS

Two timers are implemented in the firmware with the purpose of triggering events in the main loop. Both timers operate in the same fashion, with only their period and which events they activate being the distinction between the two. After a timer finishes counting, it sets its flag high. The flag is then read in the main loop to determine its status. Once learning that the flag is high, it enters its designated section of the main loop. After completing this section, the flag is set low, and the main loop will not enter into this part again until the timer causes the flag to go high. The first timer is set at a period of 10 ms and sets the flag for the PID controller. The second timer is set at a period of 100 ms and sets the flag for reading the ADCs.

3.3.8 Kicker

The kicker driver is developed to run the kicker board enabling of the boost converter, reading the current charge levels of the capacitors as the capacitor has a manual shut-off. That is achieved at max charge but since we need a higher level of control of the release of the solenoids. Then we have the IGBT driver which contains 2 methods to operate the solenoids. They run off a PWM low pulse so that means that it is configured in the migrant trip studio software and then if you were to run a normal PWM signal in that case so it doesn't your same reverted PWM signal and then whenever the signal goes low or is essentially on the kicker drained at a specific speed relative to the IGBT because we're using a 12 Volt driver the speed is relatively consistent throughout and we can't really change that using the voltage changes that are attributed to the characteristics of an IGBT So what we do instead is we pulse it and to post it because of the very quick discharge time on the PWM signal we have a very yeah essentially you have a 1/10th ms . And this allows us to control over shot power without degradation of current across. AS described before in conceptual design we there is delay for tun and turn off this can affect the power output by as much as 3.6% of the total period and with a simple offset this allows for us to accurately compensate for turn on and turn off delay as well as fall time and rise time. There were very consistent values in our testing even when there were changes in temperatures and voltages. The values were also consistent with the datasheet, with only a few nanoseconds off the datasheet values. The method to enable the chip is called 'charge enable' and that essentially enables a chip's input. The chip, when given high voltages, disables the temperature and safety circuitry for charging allowing for overvoltages, and temperatures to exceed safe limits. As such we enable this on the chip. Then there is a method that we run to check the capacitor voltage levels called 'checkChargeLevels'. A method that runs the operate Charging circuitry that has a few parameters and robot state checks. The last one we have is the IGBT state driver that operates each of them has a PWM that can be used to change the force requirement.

3.3.9 BNO085

The BNO085 IMU driver and nRF24 wireless module driver were written by ECE student Noah Page, who graduated midway through the project. These drivers are discussed at length in his published MQP paper.

The BNO085 reads and transforms IMU data into the current heading of the robot in the soccer field frame, then outputs this in a global buffer. Since the heading is in the global buffer, it can be read at any time (which would be handled in a polling manner at the relative beginning of the main loop) and therefore does not interrupt the current procedure.

The IMU has not been integrated into the main control of the robot yet, because the team was short on time and resources. Fortunately, SSL vision, the shared vision system of the Small Soccer League provides enough data to determine the heading of the robot with more accuracy than an IMU could. SSL receives a still image of the entire field, runs a processing algorithm, and returns a plethora of information of the game state, chiefly the positions and headings of each robot. The main caveat to the team not implementing the IMU for complementary control is that the update frequency of the vision system is only 60 Hz, the typical shutter rate of a camera. This low frequency heading update means the robot, in between frames, must rely on its own kinematics without the complementary checking of the IMU. Future teams will implement this in the form of Kalman filters. The first case will occur when the robot receives a new heading update with SSL outputs, the kinematics, and the IMU as its inputs. The second and more frequent case will occur in between frames, with kinematics and IMU as the only inputs.

3.3.10 NRF24

The nRF24 module is used to communicate from the master to each agent individually at 2.4 GHz. It was first written and tested for the Arduino Uno, and subsequently moved to be handled by the SAMV71. Eventually, the SAMV71 and SAMS70N20 were used to validate the sending and receiving capabilities of the wireless module in separate embedded systems. The datagram received by the agent is discussed in the following section. At the RoboCup, teams frequently report communications errors during competitions and must therefore be able to support multiple wavelengths, should they be asked by officials to change. The communication driver has not been tested in a high-noise setting, which will be vital to the success at the RoboCup.

3.3.11 NPP

NPP, or Noah’s Packet Protocol, is the byte-specification of the datagram transmitted to the agent and handled by the firmware. It has undergone several revisions, some occurring after Noah left the team. It was initially designed with a heavy emphasis on the IMU’s data; however, this quickly took a back seat to transmissions involving simpler navigation methods. The current iteration of NPP consists of thirteen used bytes, and is listed in Appendix X.

Byte 0 contains a four-bit number of the robot ID the message is meant to be sent to. All robots will be receiving the same message, but the data in the message will only be processed if the target robot ID in byte 0 matches the robot ID on a robot set by the dip switch array. If the target robot ID does not match the robot ID, the message is ignored.

Byte 1 through Byte 10Each drivetrain motor data and the dribbler motor data contain two bytes due to what the robot is receiving. The information being sent is a floating-point value of each motor’s rotational velocity measured in rad/s. The router converts this floating-point value into two bytes by multiplying the desired rotational velocity by 100 to make it an integer. This integer is then masked to separate it into two bytes, with the masking for bits 15:8 being shifted right eight bits. Once the robot receives this data, it converts the two bytes into a floating point by reversing the process in which it was originally converted. This conversion technique permits a range of -327.68 to 327.67 rad/s for the floating-point value. This range is adequate as the max rotational speed at which the motors would be operating is less than 300 rad/s. After extrapolating the rotational velocities, each motor is set to its respective velocity.

For the kicker and chipper, a byte is designated for each. Each byte controls how hard the solenoids should kick or chip the ball, or if they should not be turned on at all. If a byte of 0 is sent, then the solenoids would not actuate. If a byte of 255 is sent, then the solenoids would actuate to the full potential of the kicker board.

3.3.12 Other Peripherals

Two other peripherals exist on the main board which the firmware interacts with: the dipswitch array, and the LEDs. The dipswitch array has each of its eight switches traced to GPIO pins on the processor. The processor reads the state of these eight switches and makes an eight-bit binary number with this information. The first switch is the least significant bit in this number, while the eighth switch is the most significant bit. This eight-bit number is then utilized by other pieces of the firmware. As mentioned previously in NPP, the first four bits are used to set the robot ID. This is accomplished by masking the eight-bit number with the hexadecimal value 0F. The latter four bits are for testing purposes, mainly setting PWM values using a switch statement. Extracting the latter four bits is done by masking the eight-bit number with the hexadecimal decimal value F0 and then shifting the number right by four bits.

The four LEDs available on the processor board are each controlled by four GPIO pins on the processor. When one of the pins is set to a high state, its respective LED turns on. When set to a low state, the LED turns off. These LEDs act as debugging tools for testing other modules within the firmware.

3.3.13 Router

The router is based on the development board purchased after most of the components developed were moved to the prototype processor board. This will take some of the processing away from the AI and move it on to a processor that is not very strong but can still run in real time. The only drivers that carry over from the main board to the router code are the NRF driver. Beyond this, a separate driver is used to run the USB that is made by Microchip Studio. This process is carried out by essentially taking the root positions output by the AI and then running a loop that checks whether it needs to be given the next position, whether it has arrived at its destination, and what it is supposed to do. This is read from the microchip USB protocol. To connect the router to the AI center, the target USB port needs to be plugged into the USB computer, and then we find a need to inform Lib Serial of its properties so it can communicate over USB. On the harbor side, there are definitely a few limitations, as the current build rate is 38400, which allows us to send 4800 bytes per second. Currently, we use a static message length of 256, which only allows us to send 18 messages per second. This is very limited, and could definitely use room for changes to open up this bandwidth. This can be resolved as the current router prototype is a rather quick construction of itself to interface a robot with the main AI.As of this time, the USB protocol only has three different types of messages being sent to the router and one being sent to the AI. The messages are an RT path update, a current update. The right path update is updating where the robots are supposed to go for the play to be evaluated, and then within those two messages, there's a breakup between a breaking message and a continuation message. The breaking message means that a new art path is being sent as the conditions of the game have changed. Pi is evaluating something else, and it has realized that we cannot keep doing the same thing, so it feeds the robot a new path, which empties the queue that currently exists on the router. The second root path update message is a continuation, and it's built as the non-breaking message besides the bit orientation in the first flight. All this does is add more paths to the existing you for rrt*. The next type of message is a current robot update, which contains all data currently relevant to the robot that the AI wants to do with it, such as the observation of position, orientation, LED states, and whether it is currently kicking or chipping. Then the last message is a status update of all the robots on the field, and it contains mostly battery cell voltages and current statuses of the robots, as feeding that back to the game would be very important later in development and a good feature to have. When a display platform for status information on the robot is eventually built, this will provide insight on game states when debugging and practicing.

3.3.13.1 USB MESSAGES

The USB protocol only has three different types of messages being sent to the router and one sent to the AI. One message RRT path update, and one is a current update. The right path update is essentially updating where the robots are supposed to go for the play to be evaluated, and then within those two messages, there's a breakup between a breaking message and a continuation message. The breaking message means that a new RRT path is being sent, as the conditions of the game have changed. Pi is evaluating something else, and it has realized that we cannot keep doing the same thing, so it feeds the robot a new path, which empties the queue that currently exists on the router. The second root path update message is a continuation, and it's built as the non-breaking message besides the bit orientation in the first flight. This adds more paths to the existing for RRT*. The next type of message is a current robot update, which contains all data currently relevant to the robot that the AI wants to use, such as the observation of position, orientation, LED states, and whether it is currently kicking or chipping. Then the last message is a status update of all the robots on the field, and it contains mostly battery cell voltages and current statuses of the robots, as feeding that back to the game would be very important later in development and a good feature to have. The team, in the future, should build a display platform for status information on the robot. This is also very important to have, as it will provide insight on game states when debugging and practicing.

3.3.13.2 WIRELESS MESSAGES

For the router in the aforementioned configuration, we used a very similar packet protocol to the noise packet protocol that was developed when using and testing the router for NPP V1.2. It has a similar starting 12 bytes, but after that, we get into some debug features, which include the ability to change the activation states of all four LEDs from the router, so LED0 is byte 12 and LED3 is byte 16. The purpose of having control over the LEDs was to communicate data back to the ourselves, so we could display robot data. This allows us to be informed in real-time on robot states, as we do not have software that allows us to read into that; we can only otherwise look retrospectively at the logs to do that. This allows us to only update the code on the router while all the code is being transmitted to via the other wireless message, which is from the robot to the router. This message is very similar to the previous message except it carries battery voltages and current wheel speeds, so the first 12 bytes are built very similarly except instead of the desired wheel speed, we get the actual wheel speed of the chassis. The only other important states we currently communicate are battery voltage and current. We use the current charge of the battery as that can provide information to us when debugging and information to the AI itself if it needs to change roles. For example, if a robot is consuming too much current or it was not fully charged in practice, this will be communicated back, and the issue can be solved. It the power cells are too drained to be able to properly perform often, we can put the agent back on defense, where it will move less.

3.3.13.3 PROCESSING

The router holds data structures within it, containing the RRT, game stats, time since last message and a few methods to exist in the RRT process. There are a few arrays that contain placeholder spaces to develop a RRT path and static space. This was developed for testing the robot within the last week of MQP so it's unfinished and not entirely fleshed out. The main processing loop one is done to remove the directing of robots and the continuous updating that is required to do so off the AI to the router as this is not a very intensive process but is very math heavy. For readability purposes, putting it on the router is better when it comes to the messaging protocol complexity. Then, after four-point interpretation to get robot velocities, that is what we send to the robot. We use the “moveTo” robot function that takes points in the original orientation to the next orientation indication, and from there, it derives velocities. This is very similar to the two methods in the code base for the AI systems, which takes positions and orientations and drives until we have velocities on the simulated robot.

3.4 STRATEGY SOFTWARE MODULE

Our strategy module is responsible for all logic relating to performing a complete play. This includes analyzing the GameState based on robot and ball positions, assigning roles, defining roles, and assigning plays. The following sections will go into detail about each of these aspects.

3.4.1 PLAYBOOK STORAGE

Storage of plays was another consideration when designing the system. Plays needed to be saved even when the system was not running. It was decided that CSVs would be the simplest way to store the different plays and lower-level structures. Each line could represent an object, whether that is a Tactic, Role, or Play. From these CSVs they could be easily loaded into the code each time it is run without losing any data along the way.

3.4.2 RUNNING PLAYS

The task of running plays initiated some of the changes made to both Moves and Tactics. The new move functions made it possible for the development of the RunTactic() function. This function handles the initial task of assigning paths as well as monitoring when a skill should be completed according to the given threshold. This requires looping through the function so RunTactic(), and therefore RunPlay() which calls it for each robot, will be called constantly in our loop.

Within RunPlay() several steps are taken to ensure each robot is attempting the appropriate task. First all the Tactics are pulled from the Playbook’s storage. Then the Role Assignment module determines which role should be associated with which robot. Next RunTactic() is called for each of the individual robots to make them move towards their assigned destinations.

3.4.3 ROLE ASSIGNMENT

As part of our Play class, role assignment is responsible for assigning six total roles, one of which is the goalie, for each play. The role of the robot determines which tactics and skills it will use, as well as where to go. The roles we outlined are as follows:

- Goalie
- Primary defense
- Secondary defense

- Tertiary defense
- Primary offense
- Secondary offense
- Tertiary offense
- Formation controlled (offensive or defensive depending on situation)

These roles are assigned based on the position of the robots relative to the ball, and the current gamestate. Table 3.6 below shows the decision process used to assign the roles, as well as the associated actions with each role.

TABLE 3.6 DECISION PROCESS FOR ASSIGNING ROLES

Assigning Offensive Roles				
Game state	Role	Robot-specific situation	Action	
We have the ball OR ball is contested on opponent side	Primary Offensive	Robot has the ball (or is closest)	If on our side:	Pass ball to teammate
			If on opponent side:	Pass or shoot depending on probabilities of success
			If contested:	Approach ball to gain control over ball
	Secondary Offensive	Robot is closest to primary offensive robot	Get open for a pass, prioritizing forward movement toward goal	
	Tertiary Offense	Robot is second closest to primary offensive robot	Get open for a pass, prioritizing forward movement toward goal	
	Formation Controlled Defense	Not an offensive player	Stay towards midfield, one rightfield and one leftfield, prioritize on recovering balls that get out of play	
Assigning Defensive Roles				
Game state	Roles	Robot-specific Situation	Action	
Opponent has ball or ball is contested on our side	Primary Defense	Closest to opponent with ball	If opponent is moving toward our goal:	Stay on middle side of them to push sideways
			If opponent is static:	Get into position that blocks shot on goal

			If contested:	Approach ball to gain control over ball
	Secondary Defense	Second closest to opponent with ball	Block robot with highest probability of receiving passes/shooting on goal	
	Tertiary Defense	Third closest to opponent with ball	Block next robot with highest probability of receiving passes/shooting on goal	
	Formation Controlled Offense	Not a defensive player	Stay towards midfield, one rightfield and one leftfield, prioritize blocking back shots and recovering balls that get out of play	

In the beginning of the match, goalie is assigned to the robot closest to the goalzone. This remains the same robot throughout the entirety of gameplay. This is done to ensure no issues of awkward switches mid-play that may lead to an opening of the goal zone.

To complete role assignment for the five field-playing robots, we first determine whether we are on defense or offense by using the game state analyzer to select which batch of roles to assign. We use defensive roles whenever we are in a state where the opposing team has possession of the ball. We use offensive roles whenever we are in a state where we have possession of the ball, the ball is considered "free," or the ball is contested between teams. From there, the robot closest to the ball acts as the "primary" robot, and is responsible for immediate ball interaction, whether that is blocking the robot with the ball if on defense or passing/shooting if on offense. Secondary and tertiary roles are responsible for supporting the primary robot. On offense, this would be getting in front of the primary robot to get open to receive a pass. On defense, this would be going to opponents and blocking passes. These roles are chosen based on the second and third closest robots to the ball. Finally, the formation-controlled robots are chosen based on the two farthest robots. These robots are responsible for staying towards the midfield, and remaining open for a ball that may come loose from the active play area.

All the roles, along with their associated IDs, are held in the Play class. Once a play is chosen, the tactics assigned to different roles are distributed to the appropriate robot. Since GameState is constantly reevaluated, so are plays and therefore roles. This ensures that roles will be reassigned according to the most current ball and robot locations.

3.4.4 Moves

Moves are one of the most basic building blocks to the plays. They describe all positional and rotational changes that are to be made to a robot. Initially we had designed the move objects to contain specific x and y coordinates for a robot to start at and move to. When it was time to implement this structure, the error in our logic became apparent: most of the time the specifics of the point to go to are not known before implementation and it frequently changes with each update or robot and/or ball movement. To accommodate this new realization, we adjusted the move class to reflect more general options for robot movement. Each move currently has a type which determines what kind of move it should make. Below is a table of all the possible MoveTypes and what they do.

TABLE 3.7 ALL POSSIBLE MOVE TYPES AND THEIR DESCRIPTION

MoveType	Action
MOVE_TO_POINT	Move to a specific point and orientation on the field
MOVE_TO_BALL	Move to the ball and line up the ball manipulator with the ball
MOVE_TO_BLOCK	Move to be in the path of a pass or to block off an area while in a formation
MOVE_TO_PASS	Move to a point that maximizes the probability of a successful pass to a teammate
MOVE_TO_RECEIVE	Move to a point that maximizes the probability of a successful reception of a pass from a teammate
MOVE_WITH_BALL	Dribble with the ball to a specific point on the field

Each of the MoveTypes corresponds to a matching function that handles the calculation of where to go, unless it is directly inputted like in moveToPoint(). The functions also take in a range of x and y values that the robot can go to. The was designed to be able to cut down on the search area when looking for a pass and/or reception point.

3.4.5 TACTICS

As a reminder, the tactics class is the second level to the bottom of our code structure. It corresponds to the common skills of human soccer players such as receiving the ball. Each robot should have at least one tactic depending on the state of the game. The assignment of tactics is built around the roles. Different roles may be associated with different tactics. The effectiveness of tactics should be reflected directly in the behavior of the robots. It is one of the most important classes in our code structure.

The Tactics class was redesigned to reflect the change in the Move class, as well as the removal of the Queues. It was decided that the gamestate changed too quickly to warrant a long string of moves, thresholds, and skills. Instead, one of each component is used to simulate a movement until a skill is performed. A skill like a chip or a kick would likely change the gamestate and/or the possessing robot, which would require a change of play.

3.4.6 GOALIE TACTICS

Unless our team possesses the ball, the goalie should always try to block the ball. The general idea is to use the robot to cover the goal as much as possible. The shooter should always aim for the center of the goal when shooting without the goalie or defender. This will allow the shooter to have a broader range for error and a higher successful rate of scoring. Taking the shooter's viewpoint, the position of the goalie should always be on the line of the shooting trajectory assuming the ball is going straight to the center of the goal. Figure 3.29 demonstrates what it should look like.

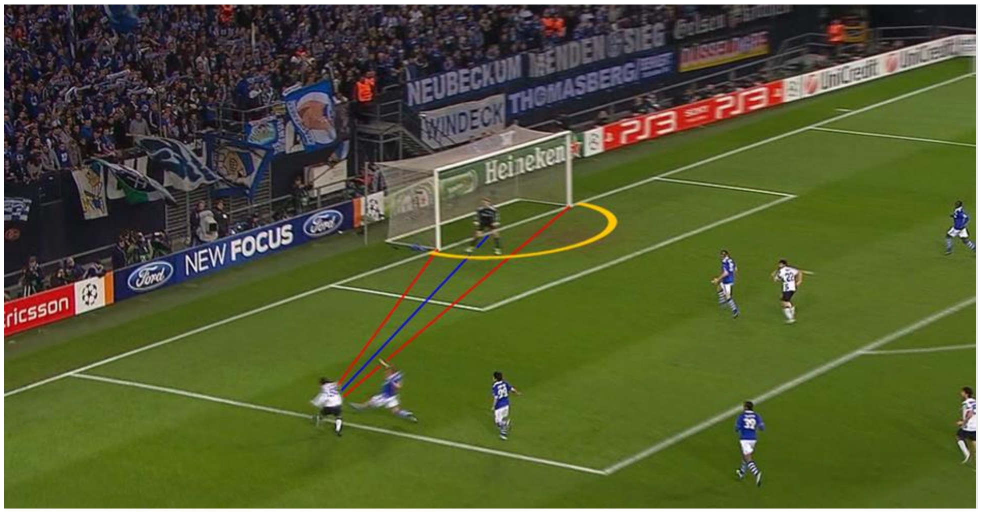


FIGURE 3.29 BEST POSITION FOR THE GOALIE BASED ON GEOMETRY (GASPAR, 2015)

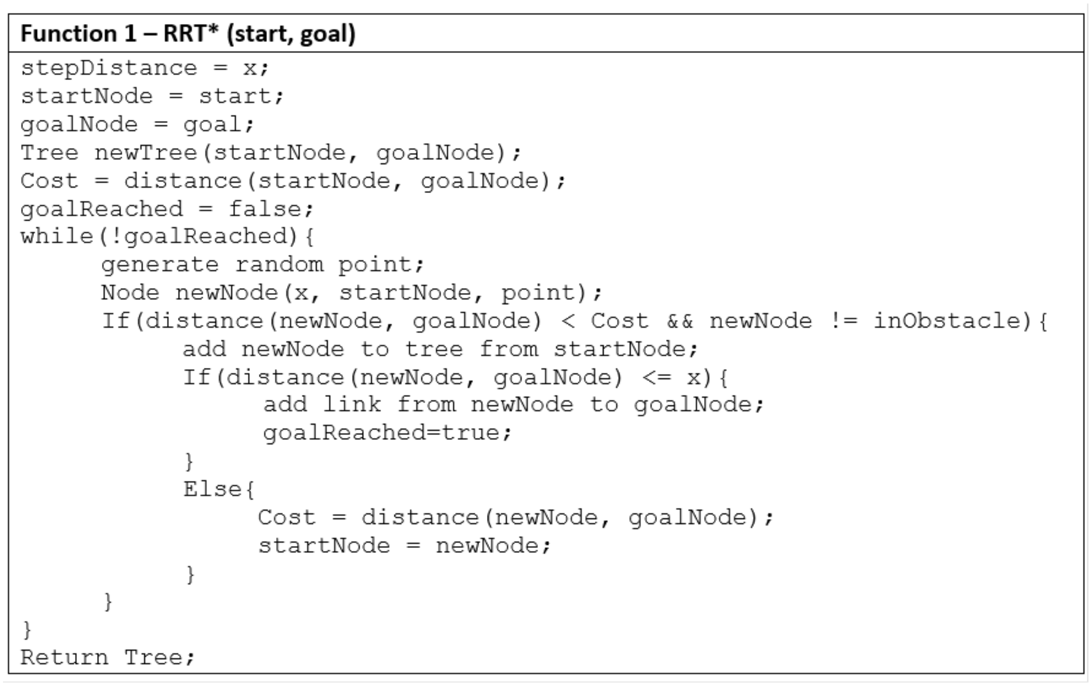
The closer the ball possessor is to the goal, the harder the blocking is. The goalie’s positions should look like a semi-circle when connecting them with a curve. Hence, the ideal position for the goalie should be the intersection point of the semi-circle and the traveling trajectory of the ball, assuming the shooter is aiming for the center of the shooting window. This skill should be sufficient for standard gameplay. If the level of competition goes up or further development is required, we can also program the goalie to move toward the trajectory path of the ball after a shot is made and before it reaches the goal.

3.5 NAVIGATION

The ability to navigate the field autonomously without crashing into other robots or walls was a tremendous skill to have for this project. Navigation was split into two main areas: Path Planning and Motion Control. The goal was to create two main functions that would help us achieve the skill of precise navigation.

3.5.1 PATH PLANNING

The first function used RRT-Star (RRT*) to plan a path from a robot’s starting position to its end goal. The pseudocode in Figure 3.30 below outlines how the RRT* function was written.



Function 1 – RRT* (start, goal)

```
stepDistance = x;
startNode = start;
goalNode = goal;
Tree newTree(startNode, goalNode);
Cost = distance(startNode, goalNode);
goalReached = false;
while(!goalReached){
      generate random point;
      Node newNode(x, startNode, point);
      If(distance(newNode, goalNode) < Cost && newNode != inObstacle){
            add newNode to tree from startNode;
            If(distance(newNode, goalNode) <= x){
                  add link from newNode to goalNode;
                  goalReached=true;
            }
            Else{
                  Cost = distance(newNode, goalNode);
                  startNode = newNode;
            }
      }
}
Return Tree;
```

FIGURE 3.30 PSEUDOCODE FOR RRT*

The step distance is the distance between the Nodes along the path. This distance is equal to the radius of the robot. The path is a Tree of Nodes with the starting and ending Nodes as the starting and ending positions of the robot. New Nodes are added to the tree when their distance is less than the previous distance and the new Node is not located within an obstacle's boundary.

When developing the path planning pseudocode, there were two main issues. One issue was that the path would get stuck on the edge of an obstacle as avoiding it would be a further distance from the goal. To fix this issue, there was a small buffer added to the cost of the next iteration of the while loop. The second issue was that the resulting path was not smooth. To fix this issue, a polynomial of best fit was used to make it smoother.

3.5.2 Motion Control

The other main function was a move-to-point function that took in the coordinates and the direction a robot wanted to move to and then outputted the wheel speeds in order to move the robot to the desired coordinate. The pseudocode in Figure 3.31 below outlines how the move-to-point function was written.

Function 2 – MoveToPoint (x, y, theta)

```
Transform_Linear_To_Wheel = [4][3];
errorX = getDeltaX;
errorY = getDeltaY;
errorTheta = getDeltaTheta;

X_velocity = pid_x(errorX);
Y_velocity = pid_y(errorY);
Theta_velocity = pid_theta(errorTheta);

Velocity(3)(1) = [X_velocity; Y_velocity; Theta_velocity];

wheelSpeeds(4)(1) = Transform_Linear_To_Wheel * Velocity;

Return wheelSpeeds;
```

FIGURE 3.31 PSEUDOCODE FOR MOVETOPOINT

When developing the motion control pseudocode, there was one main issue. The issue was that each point along the path was of equal distance. However, the error distance would approach zero as the agent moved closer to the point but would then grow as the next point is used as the error distance. This would cause the motion to consist of stops and starts. To combat this issue, there were two PID controllers used for each linear velocity (ie. Two controllers for X_velocity). However, the second controller used an overall distance between where the agent was and the last point along the path instead of the next point along the path.

3.6 SYSTEM INTEGRATION

[Describe integration activities and any changes in design resulting from interactions of subsystems. Detail how systems were integrated into overall robotic systems. Include challenges faced and how they were addressed]

3.6.1 MECHANICAL PROTOTYPE 1

After the team was finally able to finalize a design for the robot the next step was to come up with the first prototype of the robot. This process would end up being a much more laborious task than expected but was eventually figured out by the end of the term. This process's largest bottleneck was learning how to use the CAM software required to cut the parts on an endmill and then learning how to use the special endmill itself. The program that needed to be learned was ESPRIT and this task ended up being much more difficult than initially anticipated. After multiple weeks, however the programs counter intuitive nature was figured out and multiple cutting files were made for the base and top plates. Next a session needed to be spent with a lab monitor in Washburn to become familiar with the V2 endmill. This was just one part of thus prototype as the CAD had to be perfected and analysis also had to be done on the parts before they were fabricated. Below we will go in depth a little more on the final CAD and also the analysis of the robot itself to show that our design would be able to theoretically perform its primary function.

The first physical prototype of our robot is still currently in development as we wait for parts that have been ordered to be delivered but below, we will cover what we have gotten built. To start you can see the manufactured base plate in the figure below. This baseplate was cut using the V2 Endmill in the Washburn shops. The process to cut this plate started with making an ESPRIT file that consisted of the following commands. This plate is much trickier than expected to cut as the top indents which the motors will sit in had to be pocketed out by the mill first. As these were pocketed out, I also needed to outline the top half of the plate. Once the top half of the plate was outlined then I had to outline and cut the rest of the plate out all the way down to depth.

The process for this is as follows, the first step in the manufacturing process for either of the plates is to first find a solid block to fixture the stock plate onto. The reason for this is because when the outside path of the plate is cut it will stay fixtured to the base plate instead of dropping away from the drill bit. To fixture the stock to the aluminum plate you cut the standoffs holes that are on the plate into the stock and match those onto the base block. Once these are drilled and tapped and match, you then screw the stock into the base block. Once this is fixed on you place and fixture the base block into the V2 mill. Once this is done you must probe the stock material that is fastened onto the block so that the machine knows where the stock is and can center itself upon it. Once this is done you must align your tools in the machine and then run the simulation (ESPRIT and CNC files will be attached in appendix). Once the simulation is set then the part is ready to be cut.

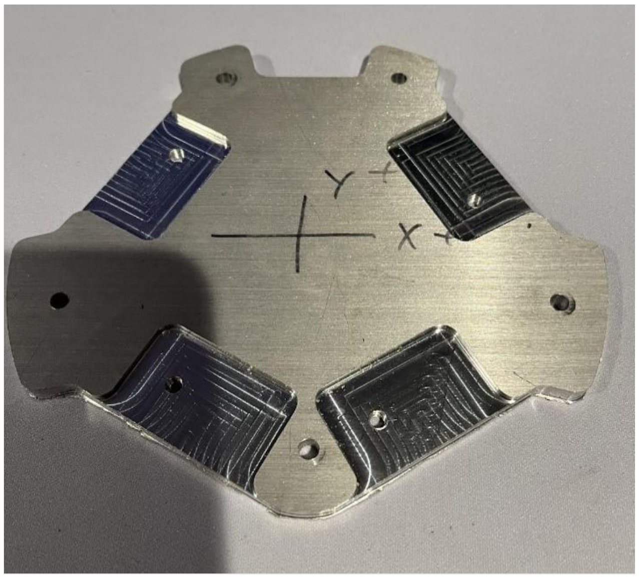


FIGURE 3.32 BASE PLATE PROTOTYPE

After the bottom plate was cut, we next needed to cut the top plate which is shown in the figure below. This cut was much simpler than the previous bottom plate. Like the bottom plate the stock was cut and fastened onto a larger base plate to secure the plate for the cut. Next the ESPRIT file was made and for this all that was needed was the outline to be cut and then the four pockets in the middle. This process was much simpler than the previous cut and was much more time efficient.

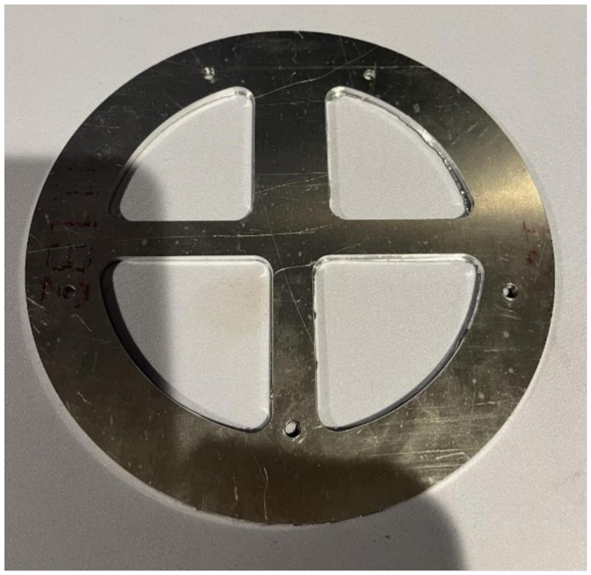


FIGURE 3.33 TOP PLATE PROTOTYPE

Once both top and bottom plates of the robot were cut, they were assembled to form the chassis which can be seen in the figure below. In the figure you can see both the top and the bottom plate fastened together by five three-inch aluminum male-to-female standoffs. The holes in each plate are tapped with 10-32 holes so that the standoffs can be screwed right into each plate and will keep the chassis of the robot rigid and not lose.

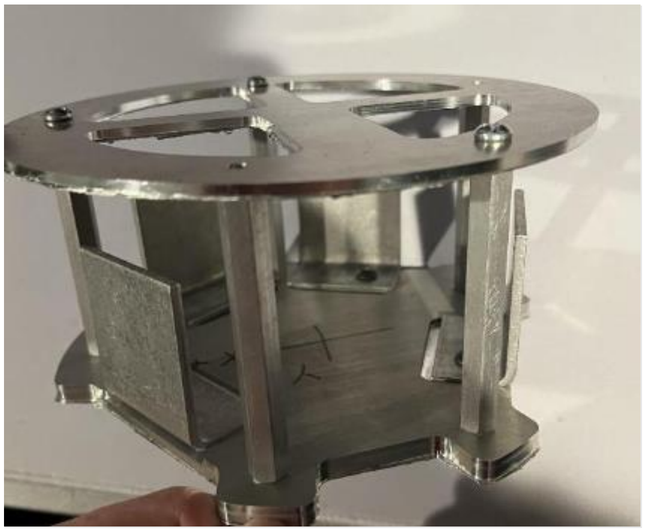


FIGURE 3.34 CHASSIS PROTOTYPE

Finally, the last step of the chassis was next. This was cutting the L Brackets so that they could be attached to the base plate and then motors could also be attached to the wheels as well. This was originally thought to be one of the easier parts of the chassis. Shown in the figure above is the assembled chassis with the L Brackets attached to the base plate. As you'll notice the Motor mount holes have not yet been cut in this prototype.

In the figure it's clear to see the vision of what the chassis prototype should look like. Cutting the Motor mount holes changed the design and was a major cause of change between this prototype and the second. The reason for that change was because when the motor mount holes were cut, a drill press was used to make the holes. The thought process behind this was that we already had the drill bits in the correct sizes so why can't we just layout the pattern and drill them. Unfortunately, this was a mistake because the hole patterns did not come up successfully and were uneven and messy. This can be seen in the figure below. The L Brackets were also accidentally bent while being clamped down to the bed of the drill press. With these challenges, we decided to make a CAM file and NC code to cut the Brackets for the second prototype.

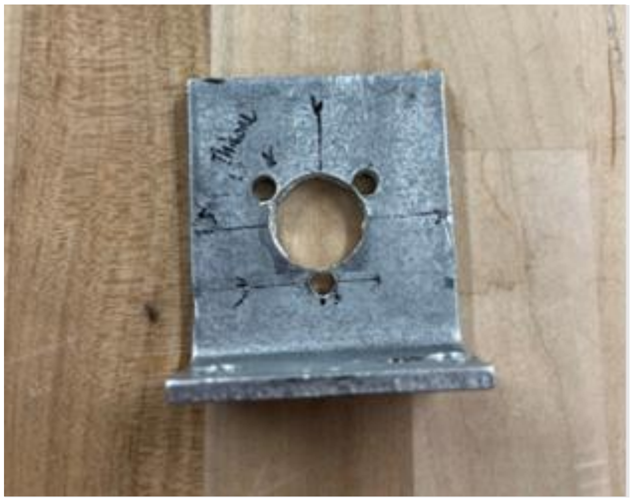


FIGURE 3.35 FIRST PROTOTYPE OF L BRACKETS

With everything manufactured for the first time the first prototype of the robot was ready to be assembled. Once everything was placed together it was not the prettiest of products, but it performed in the primary function of being able to roll around as needed, which was a success for the first try.

3.6.2 MECHANICAL PROTOTYPE 2

With the first prototype assembled, it was evident that many changes needed to be made to better the robot's chassis along with its performance. The smallest and first change was to make the tapped holes on the top plate through holes, so it was easier to attach the top plate to the standoffs. Once this was done more major changes needed to be made. The first of which was cleaning up the bottom plate. The first prototype of the bottom plate was much thicker than it needed to be. To remedy this, we manufactured another identical bottom plate and then planned the bottom surface with the mill to take of some of the material, giving us more clearance room. Another fix was also with the clearance of the robot. All the screws that attached any part of the chases together with the bottom plate protruded under the plate. To fix this we used a manual mill to surface these screws and make them flush with the baseplate.

With those fixes accomplished, the most major change of the L Brackets needed to be made. This was much simpler than expected as all we needed to do was make an NC file for the Hass Mini Mills in Washburn shops. This file told the machine to use two different tools to drill the hole pattern into the L brackets which we had pre purchased. By fastening them into a vise within the mill, all that needed to be done was to probe the Z and Y surfaces. The X surface was probed by using a different method called the x-min probe which would only probe one side of the bracket on the X axis. In the figure below, you can see what these newly machined L Brackets look like which compared to the old one, are significantly cleaner and more precise.

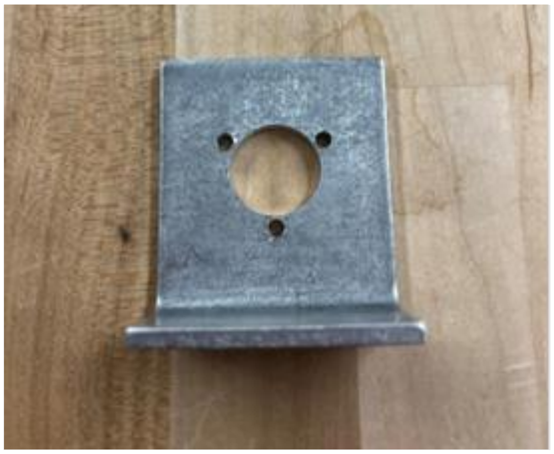


FIGURE 3.36 MILL MACHINED L BRACKETS

Once these changes were made, the newly manufactured second chassis prototype could be assembled. Show in the figure below, you can see how the second prototype looked after being assembled. The changes are evident in how the new L brackets are significantly better than the first prototypes and are also much straighter.

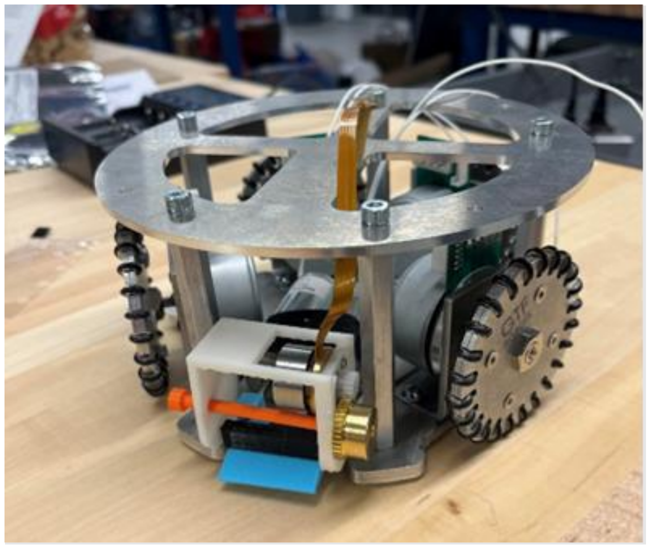


FIGURE 3.37 SECOND PHYSICAL PROTOTYPE

The final addition for this generation of the prototype robot was the updated ball control assembly. A geared dribbler system, bolstered mounting structures, and simplified geometries for ball control components such as the dribbler, kicker, and chipper highlight this subsystem iteration. These modifications were devised during the rapid prototyping phase of the system development in which design ideas were created in CAD, 3D printed, assessed for points of strength and weakness, and iteratively tweaked and improved, resulting in a compliant part.

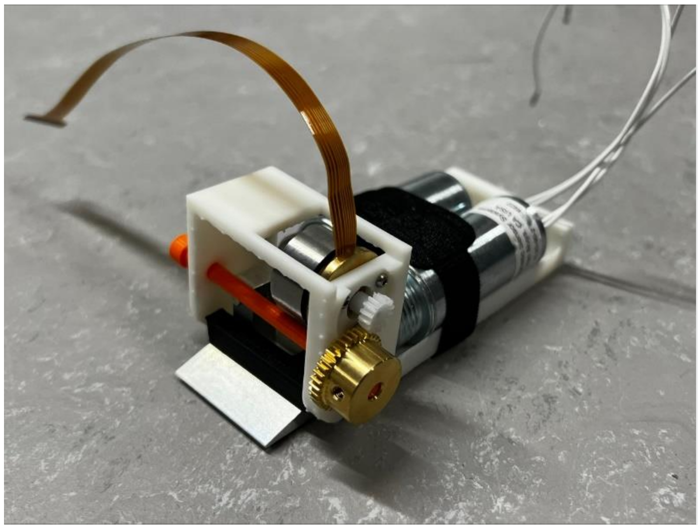


Figure 3.38 Assembled Ball Control Module

3.6.3 Electrical Integration

Electrical Integration involved assembling the power board, kicker board, and processor board by soldering electrical components to their respective boards. The boards were ordered and manufactured by JLCPCB and PCBWAY, while components were ordered primarily from Digi-Key.

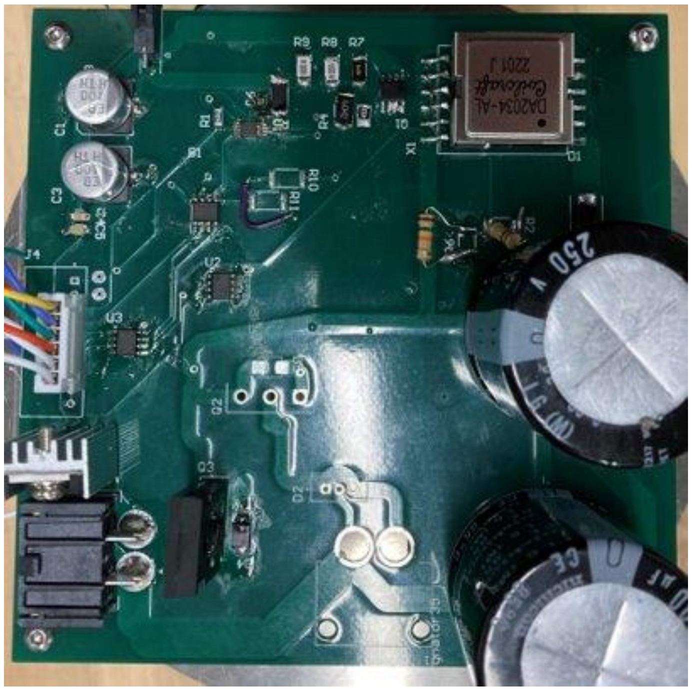


Figure 3.39 : Finished Kicker PCB

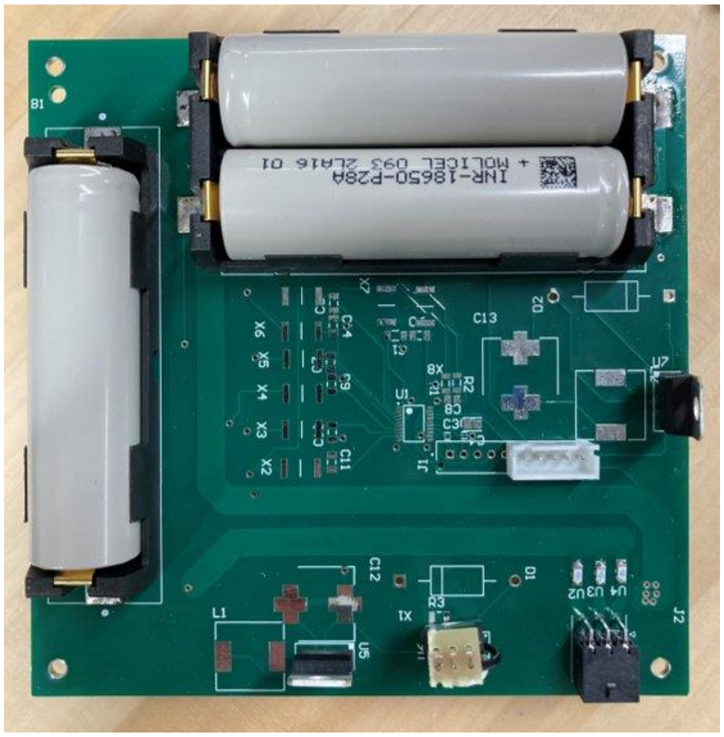


FIGURE 3.40 FINISHED POWER PCB

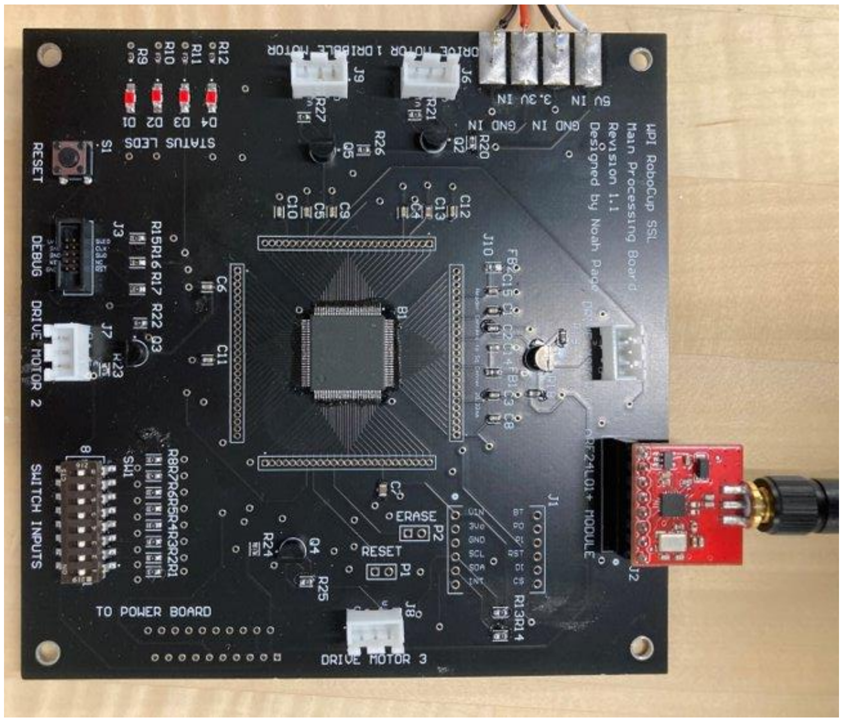


FIGURE 3.41 FINALIZED PROCESSOR BOARD

3.6.3.1 ADCS

While implementing firmware on the processor, an oversight was discovered regarding the ADC. Initially, the positive reference pin was to be set to 3.3 V using an external connector to another board. However, during the design process the decision to set the positive reference voltage on another board was eliminated. Unfortunately, the processor boards were already designed and ordered. A remedy was made by soldering a wire from the positive reference voltage pin to another pin connected to the 3.3 V power on the board.

3.6.3.2 ESCS

Changes were made to the firmware to address an issue discovered with the ESCs. During prior testing with the ESCs, they only operated in one direction within a limited PWM duty cycle range. The lower limit of the duty cycle was 20%, at which point the ESCs will stop moving the motors. The upper limit was a duty cycle of 40%, causing the motor to spin at max speed. When the ESCs were incorporated into the drivetrain of the robot, the firmware on the ESCs was updated to allow bidirectional operation. However, doing this changed the PWM duty cycle range the ESCs accepted. It put the ESCs into a bidirectional mode which only supported pulse position modulation (PPM). The team did not have time to acquire a PPM to PWM converter for testing purposes, nor was there time to integrate it in the processor board. Through trial and error, it was found that the ESC could operate using PWM interpreted as PPM in a bidirectional manner, with roughly half the range representing velocity in one direction, and half the range the other. The "PPM" flashed to the ESC was the range of PWM pulses between 4286 and 5474, which correspond to a pulse range of between 3.6ms and 4.6ms based on a 200 Hz PWM instance. The firmware was updated to reflect this new range of accepted PWM signals.

CHAPTER 4 SYSTEM TESTING AND VALIDATION

This chapter outlines how we tested and validated all sub-systems of the design. We include results from our tests as well as metrics to evaluate performance.

4.1 TESTING MECHANICAL SYSTEM

4.1.1 System Requirements

The sub-team identified key system requirements for the shell that were necessary to other sub-teams in this next phase.

4.1.1.1 FUNCTIONAL REQUIREMENTS

The functional requirements of the system are as follows:

FR-I. The prototype must roll freely on a carpeted surface (analogous to the regulation RoboCup playing field) without interference or hinderance from other components (bolts/threads from standoffs rubbing on surface, drive motors not rubbing against solenoids)

FR-II. The prototype must support installation of electronic systems such as PCBs and batteries.

FR-III. All subsystems of the prototype must be fully assembled and implemented such that the operational requirements of the system can be satisfied.

4.1.1.2 OPERATIONAL REQUIREMENTS

The operational requirements of the system are as follows:

OR-I. The prototype must operate (move, manipulate ball, communicate with code) as a physical analogue of its virtual counterpart in grSim.

OR-II. The prototype must provide an accurate and reliable physical basis for data collection that indicates areas of improvement for electromechanical systems, programming, and other physical parameters such as weight distribution and geometry.

4.1.2 System Outcomes

The milestones we met during C-Term can be used to evaluate how well these requirements were addressed and fulfilled. Based on the progress reported in the sections above for this term, the system requirements are addressed as shown below:

4.1.2.1 FUNCTIONAL REQUIREMENTS

FR-I. The team demonstrated the prototype to roll freely upon its completion at the end of C-Term, and verified that the previously mentioned hinderances are not a factor on a carpeted surface.

FR-II. When installed, all electromechanical components' electrical leads and connection points are accessible and conveniently placed for integration with their corresponding hardware and circuitry. Furthermore, threaded holes are placed on the top plate to mount any PCBs above the robot using standoffs (which are in our possession) as specified by the ECE sub-team.

FR-III. The prototype is assembled in our lab space in Unity Hall and ready for testing by the other sub-teams to verify compliance with the operational requirements.

4.1.2.2 OPERATIONAL REQUIREMENTS

OR-I. This requirement will be evaluated in D-Term by the other sub-teams. The ME sub-team will be on standby to support the successful integration and verification of this requirement.

OR-II. This requirement will also be evaluated in D-Term by the other sub-teams. The ME sub-team will be on standby to support the successful evaluation of this requirement.

4.1.3 PLANNED TESTING AND VALIDATION

The ME sub-team supported the other sub-teams in integrating our work with theirs and ensuring that all needs were met by these teams in their testing and validation. The testing and validation activities are motor testing and tuning, ball control capabilities testing, vision testing, communication testing, and motion planning testing.

Motor testing and tuning consists of installation and integration of drive motor into the associated circuit and characterization of their capabilities with respect to the desired performance of the prototype. The team was tasked with assisting in the installation, data collection, and speed/torque profile tuning of the motors.

Ball control capabilities testing consists of performing routine actions with a regulation ball such as kicking, chipping, and dribbling. The team was tasked with assisting in the installation and tuning of the dribbler motor and solenoids, as well as characterizing and adjusting the performance of these components with respect to their interactions with the ball.

Vision testing consists of mounting and testing of the camera vision system. The team was tasked with assisting in the installation of acceptable equipment to mount the camera vision system. The team has already identified a telescoping photography backdrop stands that may be acceptable for mounting the camera and has identified a carpeted space in the Sports and Recreation Center that can be booked for testing.

Communication testing consists of ensuring that the prototype's hardware and associated programming properly interacts with the camera vision system.

Motion planning testing consists of ensuring that the electromechanical components such as drive motors behave as their programming and communication with camera vision system would suggest. For these final testing activities, as well as the previously mentioned ones, we will provide support to the sub-teams with any mechanical needs that may arise by performing supplementary simulations, analyses, and fabrication.

While the above activites did not end up taking place in D-term due to budget, time, and supply chain related constraints, the ME sub-team was responsible for the creation of an additional two prototype robots in D-term to support presentation and demonstration activities. These robots were constructed using mostly 3D printed parts including base plates, top plates, motor mounts, and shells. Remaining leftover and unused prototype components from previous design iterations were salvaged and modified to create an additional shell for demonstration purposes including base and top plates and standoffs. Supplementary ball control structures were printed to aid in demonstrations as well.

4.2 TESTING ELECTRICAL SYSTEM

The following sections describe the Electrical and Computer Engineering sub-team's findings during testing of the processor board, the kicker board, the power board, and motor controller system.

4.2.1 PROCESSOR BOARD

The processor board being the brain of the robot had to have a lot of functionality, which required extensive testing to verify all aspects worked appropriately. Configuring the processor and uploading firmware to it went smoothly throughout the time of the MQP. The LEDs on the board used for debugging worked flawlessly. The processor had no issues reading the dipswitch array on the board. The PWM generators worked well, as did the logic converters used to bring the 3.3V logic of the processor up to 5V for the ESCs. The wireless communication module attached to the board was susceptible to noise while operating at 2.52GHz. However, if a signal was being constantly sent from the router the noise was negligible. The ADCs performed well and provided accurate readings within its operable range. The IMU on the processor board was not extensively tested or used due to a focus on other systems. Please refer to Noah Page's report for more information regarding the IMU.

TABLE 4.1 PERFORMANCE ANALYSIS OF PROCESSOR BOARD

System	System Performance
Processor & Uploading Firmware	Well
Debugging LEDs	Well
Dipswitch Array	Well
PWM generators	Well
Logic Converters	Well
Wireless Module	Okay
ADC	Well
IMU	Refer to Noah Page's report

4.2.2 KICKER BOARD

The kicker board initially had a few issues during testing. In the first testing one of the capacitors was damaged because too much current was pulled from them which caused damage to components on the kicker board, the capacitors included. This was changed by adding multiple capacitiors to allow for more current draw at higher voltages. This also allowed the capacitors to keep cool within their safe operating temperatures, preventing further damage. On the kicker board, it was discovered that the voltage coming raw directly off the capacitor was very noisy. This was solved by putting the output into an RC filter of 1K ohms and 100 nano-farads, respectively. After adding the filter, testing showed a smoother signal without eroding our data delay, but consequently, it does mean that on boot there must be a grace period before acquiring accurate results, as the capacitor in the RC filter needs time to charge.

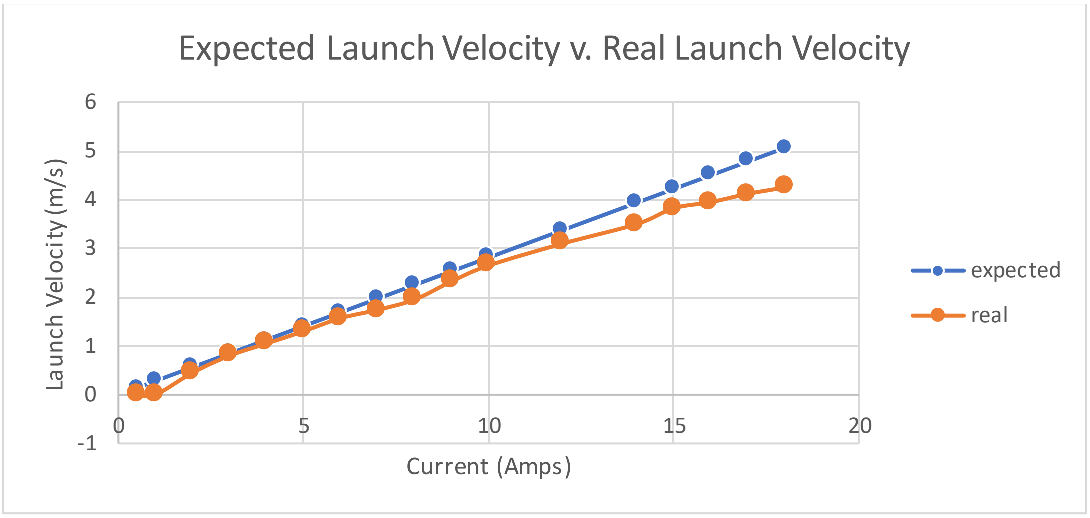


FIGURE 4.1 COMPARISON OF REAL LAUNCH SPEED (ORANGE) AND EXPECTED LAUNCH SPEED (BLUE)

4.2.3 POWER BOARD

When testing the power board, a few discrepancies were discovered. One of the LDOs being used did not supply the exact voltage originally intended. The regulator responsible for 5V power ended up producing 4.75V instead. The 12V regulator had an output of 11.9V on average as well. Despite these errors, the other boards utilizing the voltage from these regulators still operated well. The irregularities in the 5V regulator are likely due to the 3.3V regulator drawing from it. The 3.3V regulator performed well and without issue. The battery pack for the power board was designed with a nominal of 22.2V, considering the batteries being used. However, the batteries were able to be overcharged, bringing the total voltage up to 25.2V. This proved to be fine though, as the motors and voltage regulators were able to operate at 25.2V.

TABLE 4.2 EXPECTED VOLTAGES VS. ACTUAL VOLTAGES ON THE POWER BOARD

Expected Voltage	Actual Voltage
22.2V	25.2V
12V	11.9V
5V	4.75V
3.3V	3.3V

4.2.4 ESCS & MOTOR CONTROLLER

Testing the ESCs and the motor controller involved several iterations of firmware in order to achieve optimal behavior and overcome unideal aspects. One aspect is that the PWM range of 1188 total pulses is essentially cut in half since each direction reserves half of the resolution, or 594. Furthermore, there is a deadband-zone on each side of the zero-effort value of 54, thereby decreasing the resolution of a single direction to 540 pulses. These pulses represent roughly 0 to 10 meters per second, meaning our theoretical resolution of linear wheel velocity is 0.0185 m/s. It should be noted due to the deadband zone, the smallest linear velocity obtained is +/- 0.6 m/s, which is *not* the same as the resolution of the linear velocity. To be clear, the resolution of velocity is the change in velocity due to the respective change in PWM pulse steps. The *resolution* holds until the linear speed approaches +/- 0.6 m/s, but beyond this threshold is either 0.6 m/s or 0 m/s. This is an unacceptable minimum speed for future iterations of the project as there will be times when the robot needs to go slower than this (particularly in support roles) but not quite 0 m/s.

With the above constraints put in place, we were then able to work on the PI controller. Due to delays in shipping and order errors, the encoders were not available until the final week of the MQP, leaving little time to determine a proper solution for the motor controller under load. Unloaded, the controller converged deftly on any value between +/- 10 m/s (minus the deadband zone), albeit there was often electrical noise between ESCs in which one ESC would catalyze another to begin rotating, even though in software it had been set to zero velocity. The root cause of this problem was never determined as the debugger broke 1 day before project presentation day. Please see the recommendations section for advice on the motor controller in both hardware and software.

4.3 TESTING SOFTWARE

To expedite the testing process of our software architecture, we first had to develop or modify a simulator. Simulation was a very important focus of this project as it enabled us to start testing our software architecture without the actual field equipment. When deciding what simulator to use, we found many reasons to choose grSim over the others. Most importantly, it was an official simulator backed and supported by RoboCup which means, in theory, there should be less problems with it. Also, it provided many basic features for a startup team and had the Simulation protocol already implemented. The overall process of the simulation can be boiled down to a feedback loop. During simulation, the simulator broadcasted the position information of the robots and the ball based on frames. In addition, the simulator reacted to the command it was receiving from our software libraries by updating the positions of the robots and the ball. The data that was being transferred during this process was also called the vision.

There are four main requirements that make up a functional RoboCup simulator: send field information, receive field information, send robot commands, and receive robot commands. Based on our structure of communication, sending field information should be done by grSim. In other words, grSim should be able to broadcast field information without us needing to change anything. grSim keeps track of the position data for each robot and the ball using the OpenGL library. We validated this by checking line by line in the SSLWorld library. From the TDP of grSim, the developers have made grSim send field data based on the framerate of the simulator. We also proved this to be true after tracking down the details of the code, although there was no documentation or website to help us use the simulator. The vision data is broadcasted at the vision port approximately 60 times every second.

Receiving field information is handled by the robot software we are developing throughout the term. To make life easier, we organized the code into libraries based on their purposes. For example, Navigation, Strategy, and Simulation are the three different libraries since they have three completely different purposes. The Simulation library will automatically listen to the broadcasting ports of grSim and store the field information as a private field if this object class has been initialized. In addition, we parse the information of the incoming data package using Google Protobuf based on the Simulation Protocol. Parsed field information including position data of the robots and the ball will be stored in a private variable. There is no need to update these private variables manually since it updates itself whenever it reads newer information from the UDP sockets, the communication protocol grSim uses.

Sending robot commands is also included in the Simulation library. We constructed a Google Protobuf message that conforms to the Simulation Protocol. Then, we created a UDP socket to set up communications between the simulator and our software. Finally, we sent the message to grSim's IP address through the socket.

Receiving robot commands was handled by grSim. grSim has a few port settings for receiving commands. Once the command message is delivered, grSim will automatically order the robot to do whatever the command message told it. After testing with multiple different commands, we were confident that grSim and the Simulation library worked well. The following testing of the software libraries was done on grSim.

4.3.1 Testing Strategy

Testing the strategy module required the development of most of the following systems beforehand, as testing the entire strategy component tests the compatibility of each function. There were also some new features that came about when modifying the Move and Tactic classes. Testing the new functions that handle all the moves like goToPoint() and goToBall() were tested by using grSim to visually verify the accuracy of the robot's path and destination. These functions were tested using multiple starting configurations to decrease the likelihood that the paths and destinations were correctly determined.

Another way we have been testing the Strategy module is by using a test script and running it on the terminal. This has allowed us to input various configurations and to see how GameStateAnalyzer analyzes the data given. The goal of this script is to provide a comprehensive test of some of the units so identifying the bugs that pop up later will be a simpler task.

4.3.1.1 TESTING PROBABILITY OF A PASS

The numerical value of the probability of a pass being successful is dependent on a few variables: the closest robot from each team's distance to intercept the ball, the two robot's top speeds, and the speed of the ball. The speed of the ball is used to generate time driven parametric points used to determine when each robot can intercept. The robots' respective distances to the ball's path and top speeds are combined to create an estimate of how long it will take for the robot in question to intercept the ball. The difference between these two times is scaled to fit between 0 and 1, with values still outside of the range snapped to the nearest valid value.

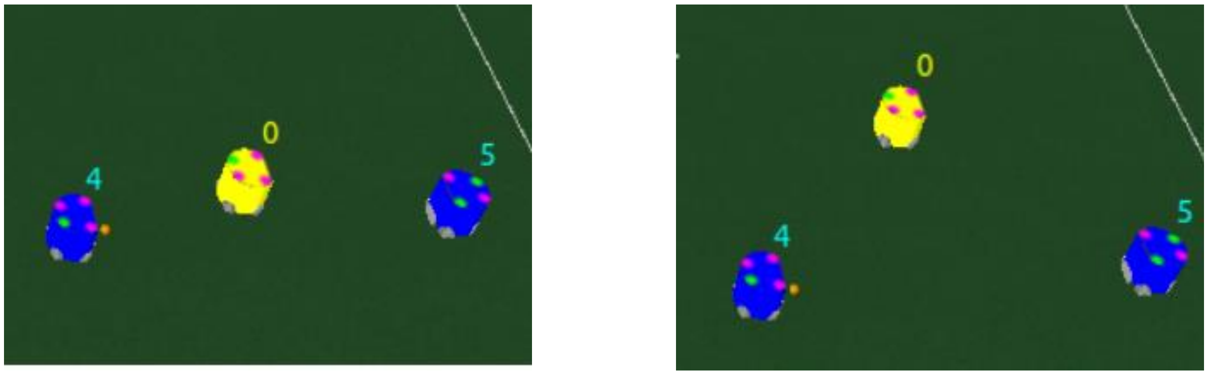


FIGURE 4.2 LOW(LEFT) VS HIGH(RIGHT) PROBABILITY OF A SUCCESSFUL PASS

To test this functionality, two blue robots were set up to complete a pass. One yellow robot was placed at varying distances to the balls path to the second robot. We verified the accuracy of the probability of a pass by checking that the probability of a successful pass increased the further away from the ball's path the yellow robot was. Since the probability value itself is arbitrary and only the relationship between two probabilities matter, this test was enough to properly verify the functionality of the function.

4.3.1.2 TESTING PROBABILITY OF A SHOT

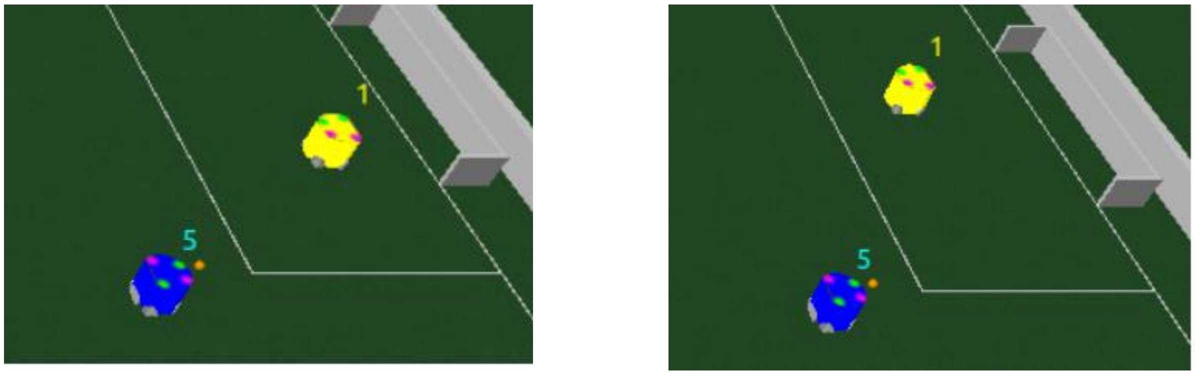


FIGURE 4.3 LOW(LEFT) VS HIGH(RIGHT) PROBABILITY OF A SUCCESSFUL SHOT ON GOAL

The numerical value of the probability of scoring a goal is determined by two components: the probability of interception by the opposing team, and the probability of a goal from the given angle. The probability of interception by the opposing team is calculated similarly to the probability of a successful pass with the time to cross the goal line being substituted in for the time to be received by a teammate. The second component of the probability of a goal from the current angle is used to account for the difference in goal area available to be scored in. This probability is logarithmic to reflect both the minor changes in probability when the angle to the goal line is close to 90 degrees and the larger changes in probability when the angle to the goal line is close to 0 degrees.

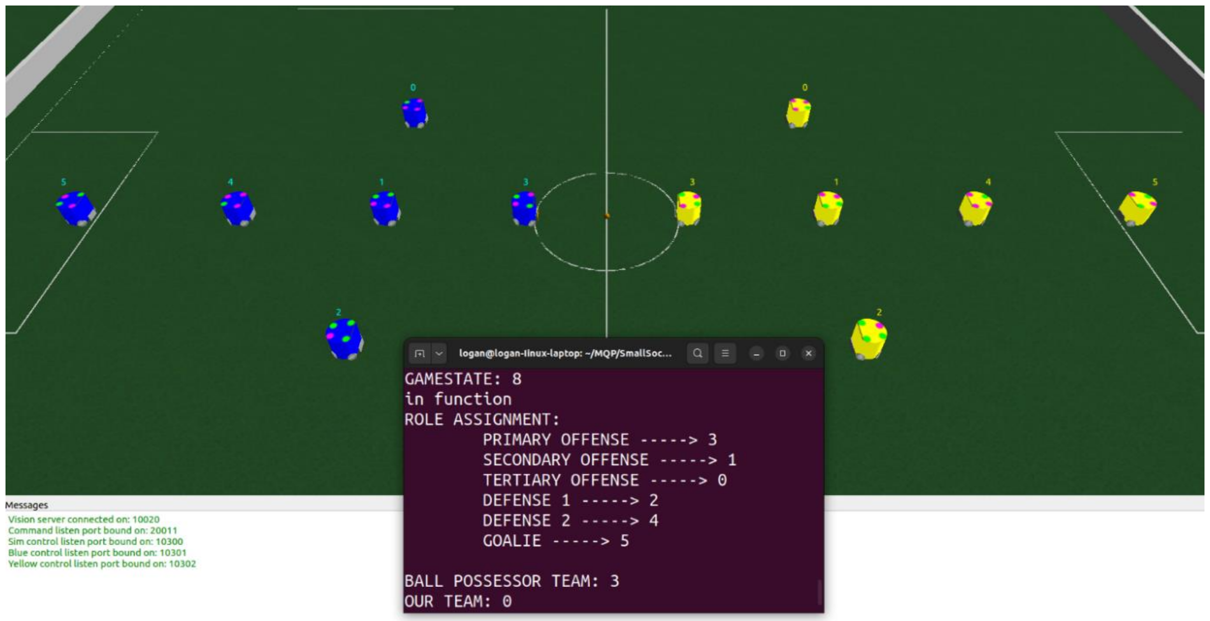
Each component of the probability was tested separately and then again together. First, we tested the logarithmic probability of a goal given the current angle by placing the robot at different points that formed different angles with the goal line and verifying that the probability was larger when the angle was closer to 90 degrees to the goal line. Next, we tested the probability of interception by the opposing team in the same manner we tested the probability of a successful pass. When combined, the probability properly reflected the two components.

4.3.1.3 TESTING MOVING, PASSING, AND DRIBBLING

Testing the actions of the robots in response to commands was done entirely within grSim due to hardware development constraints. Several move functions were developed in order to group like movements together, including moveToBall, moveToPoint, and moveToBlock. Passes were tested by running a tactic to move to the ball and then kick the ball in a specified direction to a waiting teammate. Dribbling was accomplished by spinning the dribbler while the ball was possessed by the given robot. All these actions were verified by running them within the simulator and comparing the robots movements to what their expected actions were. Videos are available in the shared folder.

4.3.2 TESTING ROLE ASSIGNMENT

To assess the success of role assignment, we assigned roles to a multitude of gamestates and robot configurations and compared the results to the expected output. For all configurations, role assignment properly assigned expected roles. The following figures showcase some of these results.



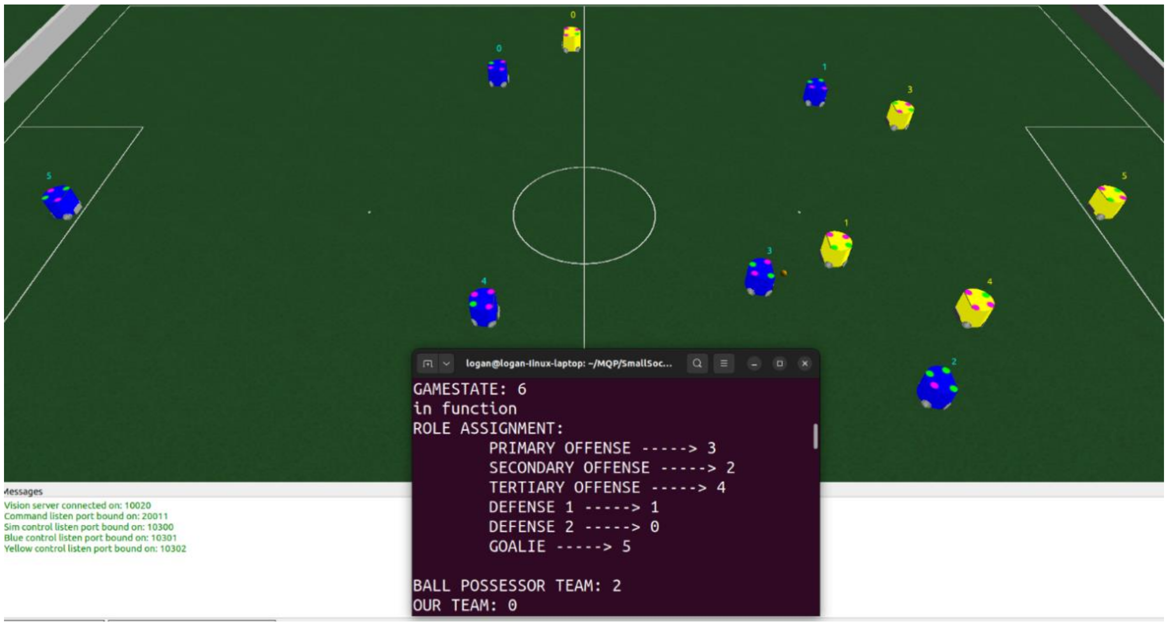
```
GAMESTATE: 8
in function
ROLE ASSIGNMENT:
          PRIMARY OFFENSE -----> 3
          SECONDARY OFFENSE -----> 1
          TERTIARY OFFENSE -----> 0
          DEFENSE 1 -----> 2
          DEFENSE 2 -----> 4
          GOALIE -----> 5

BALL POSSESSOR TEAM: 3
OUR TEAM: 0
```

```
Vision server connected on: 10020
Command listen port bound on: 20011
Sim control listen port bound on: 10300
Blue control listen port bound on: 10301
Yellow control listen port bound on: 10302
```

FIGURE 4.4 INITIALIZATION OF ROLES IN A NEW GAME

Figure 4.4 above shows the initialization of roles in a new game from their default positions. The gamestate in this case is 8 meaning the ball is contested and the ball is on the opponent's side, which is how we evaluate a ball on the line. In this case, there is no goalie previously assigned so role assignment takes the robot closest to the goal and assigned it to goalie, in this case robot 5. Since the ball is contested, the remaining robots are assigned offensive roles. The primary offender is assigned to robot 3, which is closest to the ball. The secondary offender is assigned to the next closest robot, robot 1. For tertiary offender, there is a tie in distance between robot 2 and

0. In this case, role-assignment will choose the robot with the lower ID number by default and assign tertiary offender to robot 0. This leaves defenders 1 and 2 as open roles, which are assigned to robots 2 and 4.

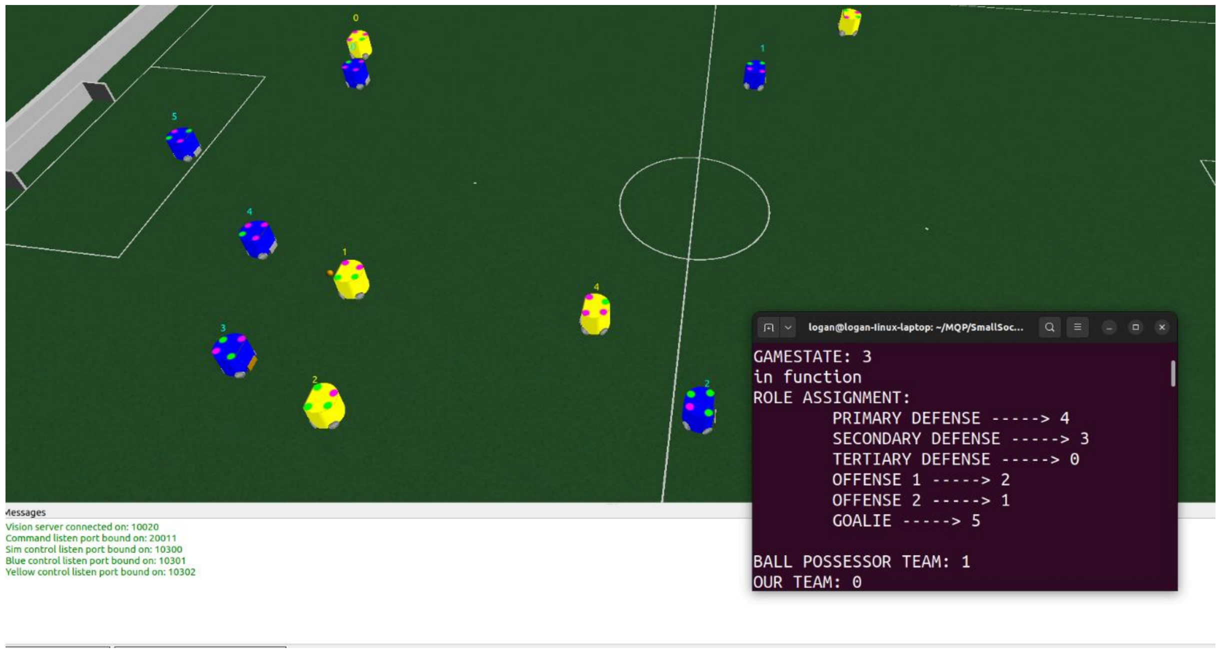


```
Vision server connected on: 10020
Command listen port bound on: 20011
Sim control listen port bound on: 10300
Blue control listen port bound on: 10301
Yellow control listen port bound on: 10302
```

```
GAMESTATE: 6
in function
ROLE ASSIGNMENT:
        PRIMARY OFFENSE -----> 3
        SECONDARY OFFENSE -----> 2
        TERTIARY OFFENSE -----> 4
        DEFENSE 1 -----> 1
        DEFENSE 2 -----> 0
        GOALIE -----> 5

BALL POSSESSOR TEAM: 2
OUR TEAM: 0
```

FIGURE 4.5 ROLE ASSIGNMENT IN OFFENSIVE GAMESTATE

Figure 4.5 above shows role assignment in an offensive game state. In this case, the gamestate is identified as gamestate 6, meaning the blue team, us in this scenario, has possession of the ball and the ball is on the yellow teams, or opponents, side. Using this information, we begin to assign offensive roles. Robot 5 remains the goalie from its initial assignment. The primary offensive role is assigned to robot 3, which is closest to the ball. Secondary and tertiary positions are assigned to robot 2 and 4, the two next closest robots to the ball. Finally, robots 1 and 0 are assigned to defense 1 and defense 2, the formation-controlled roles.



```
GAMESTATE: 3
in function
ROLE ASSIGNMENT:
        PRIMARY DEFENSE -----> 4
        SECONDARY DEFENSE -----> 3
        TERTIARY DEFENSE -----> 0
        OFFENSE 1 -----> 2
        OFFENSE 2 -----> 1
        GOALIE -----> 5

BALL POSSESSOR TEAM: 1
OUR TEAM: 0
```

```
Messages
Vision server connected on: 10020
Command listen port bound on: 20011
Sim control listen port bound on: 10300
Blue control listen port bound on: 10301
Yellow control listen port bound on: 10302
```

FIGURE 4.6 ROLE ASSIGNMENT IN DEFENSIVE GAMESTATE

Figure 4.6 shows how roles are reassigned when the robots shift into a defensive position. The gamestate in this case is 3 meaning that the opponents have possession of the ball, and the ball is located on our side. The goalie continues to remain the same, so again is assigned to robot 5. The remaining robots are assigned defensive roles. Like the above scenario, these are based on distance to the ball. Primary defender is assigned to robot 4, which is closest to the ball. Secondary and tertiary defenders are assigned to robots 3 and 0, the next two closest robots. Finally, offense 1 and 2 are assigned to robots 2 and 1.

Overall, role assignment can successfully assign roles based on distance to ball and current gamestate. The next step in game play is assigning appropriate tactics to the roles.

4.3.3 Testing Role Tactics

The following sections outline the testing of different roles and their associated tactics. While not all roles are complete at the time of submission, tactics and game-play algorithms are thoroughly tested and working.

4.3.3.1 TESTING PRIMARY OFFENDER

The primary offender is responsible for, initially, traveling to the ball location if on offense. Figure 4.7 below shows an example of a primary offender role.

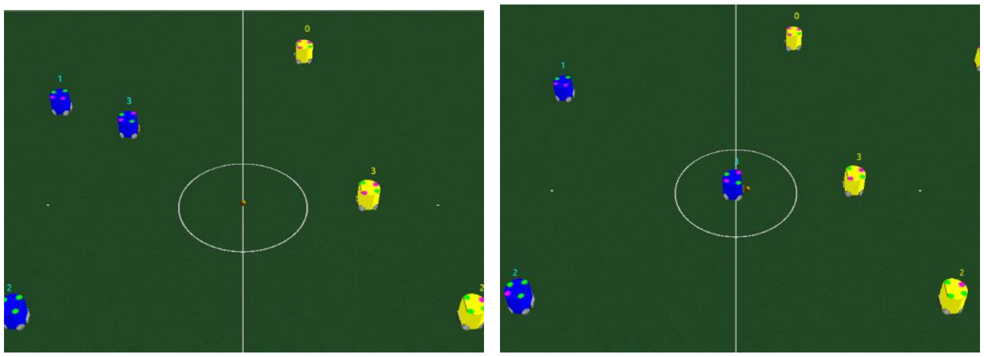


FIGURE 4.7 PRIMARY OFFENDER BEFORE MOVEMENT (LEFT) AND AFTER (RIGHT)

As seen above, robot 3 is the closest robot to the ball. Once assigned to primary offender, it moves to the correct location near the ball, close enough to use the dribbler to hold the ball to itself. At this time, the decision-making algorithm was not advanced enough to be able to choose whether to pass or shoot the ball. However, we are capable of evaluating the probability of success of those options, which could be integrated at a later point.

4.3.3.2 TESTING SECONDARY AND TERTIARY OFFENDERS

The secondary and tertiary offenders are responsible for traveling up the sidelines, in front of the primary offender and ball, and remain open to receive a pass. Figure 4.8 below shows an example of the secondary and tertiary roles acting by themselves.

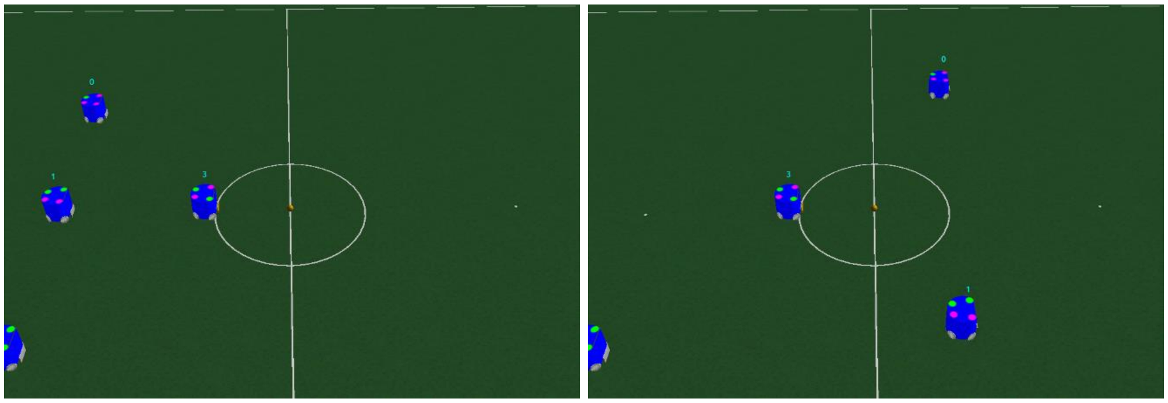


FIGURE 4.8 SECONDARY AND TERTIARY OFFENDER BEFORE MOVEMENT (LEFT) AND AFTER (RIGHT)

As seen in the figure above, the robots properly move to the correct side of the field, and then forward until they are closer to the goal than the primary offender. The robots are assigned to the left side or right side of the field based on their initial position, whether they are already on the left or right side.

One issue is that when the ball is moved before the secondary and tertiary robots reach their final positions, this sometimes causes the robots to stop moving, or continue moving without a destination. We believe this is caused because the secondary and tertiary robots final positions are dependent on the ball location, and when it is moved, the calculations have to be redone, sometimes causing an error in the robot movement. This could also be caused by the switching of roles before the previous robot has completed its movement. Figure 4.9 below shows an example of this error.

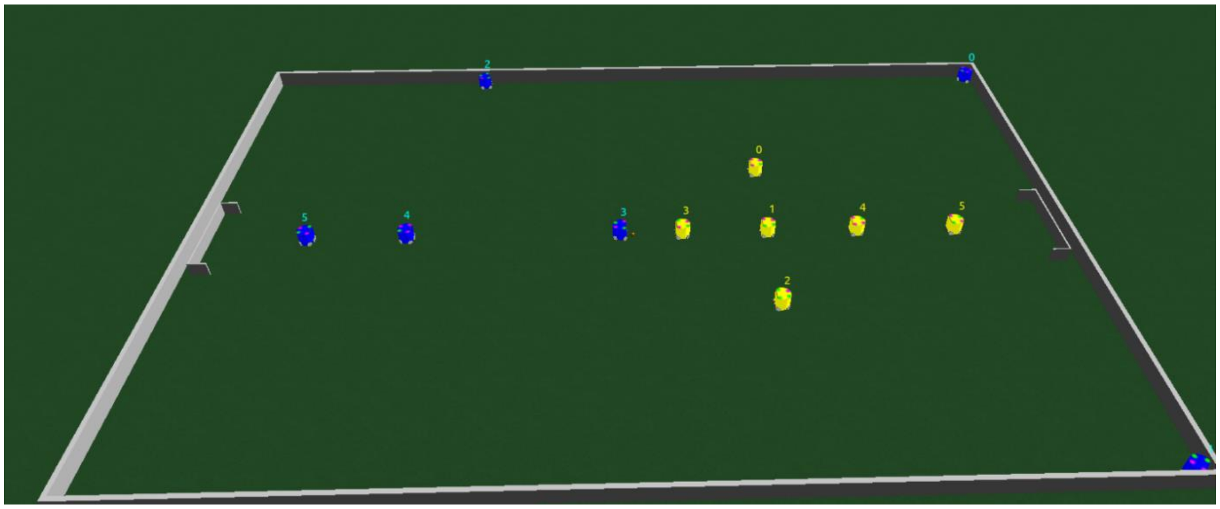


FIGURE 4.9 ERROR WITH MULTI-ROBOT MOVEMENT AFTER BALL MOVES OR GAMESTATE REEVALUATED TO SOON

While these positions are acting properly when moving on their own, when getting into multi-robot movement, there are sometimes issues with how the movements are interpreted, and therefore incomplete movements.

4.3.3.3 TESTING PRIMARY DEFENDER

The primary defender must get in the way of the opposing team ball-possessing ball when on defense. Figure 4.10 below shows the primary defender role against an opposing team.

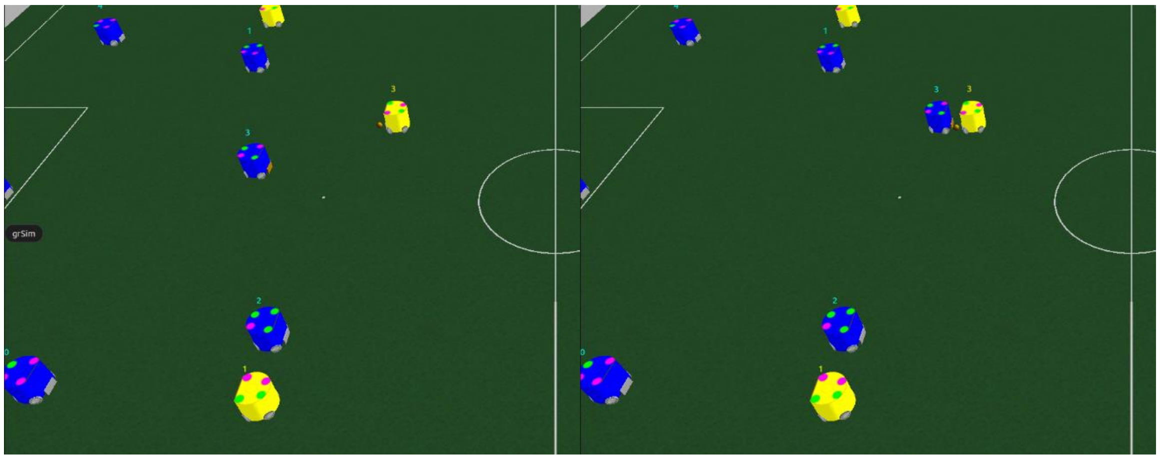
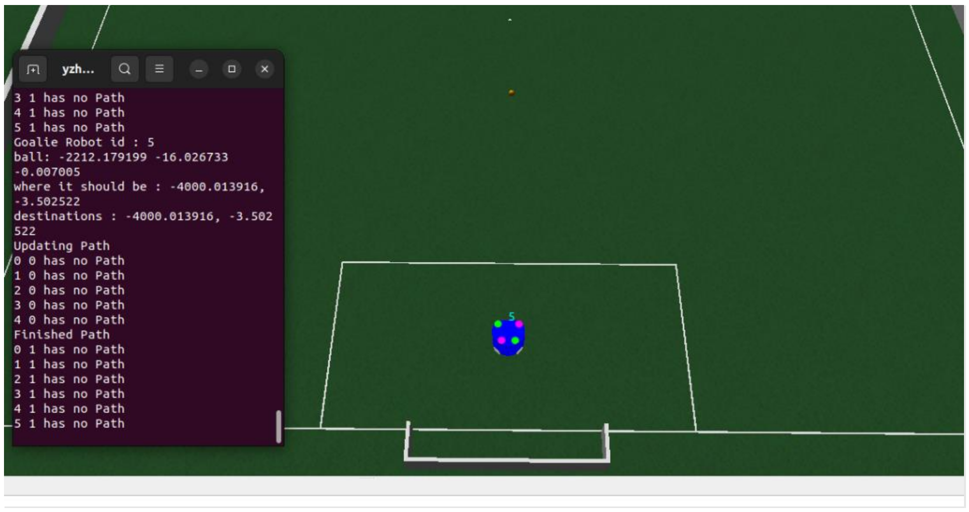


FIGURE 4.10 PRIMARY DEFENDER BEFORE MOVEMENT (LEFT) AND AFTER (RIGHT)

As seen in the figure above, robot 3 is successfully standing in front of the opposing teams offender. This robot successfully blocks off a front shot or pass from this robot throughout the game, preventing the opposing team from making an easy shot on goal. Combined with the goalies blocking, the opposing team will have a hard time making a successful shot on goal.

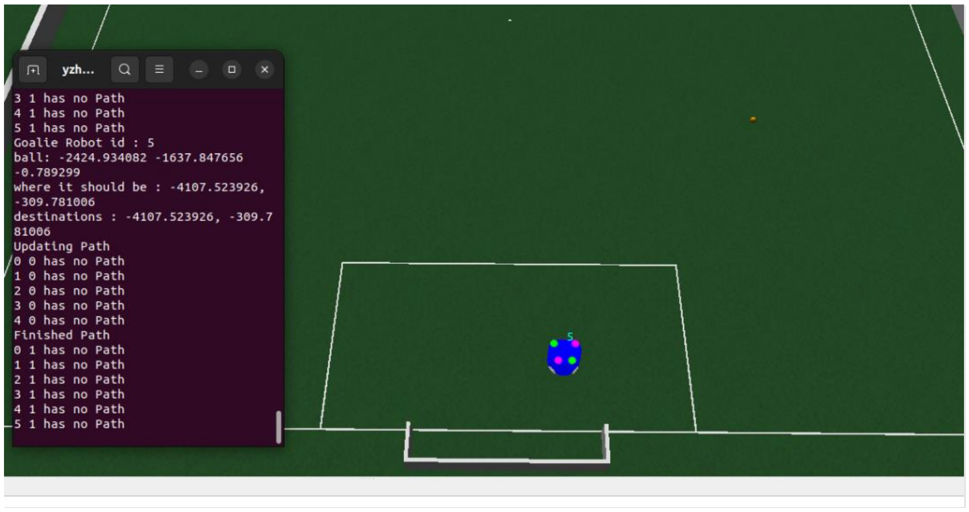
4.3.3.4 TESTING GOALIE DEFENDING TACTICS

A basic test for goalie-defending tactics is to run it on grSim and see if the destination position of matches the expectation. We tested this tactic when the ball is in different positions on the field. We manually turn off the other robots and move them to the outside of the field in order to avoid unnecessary distractions. The outcome of this tactic should be a point closest to the ball on the semi-circle which shares the center point with the goal (and has a diameter equal to the length of the goal). Most of the results matched expectations. The following figures show the results that matched expectations.



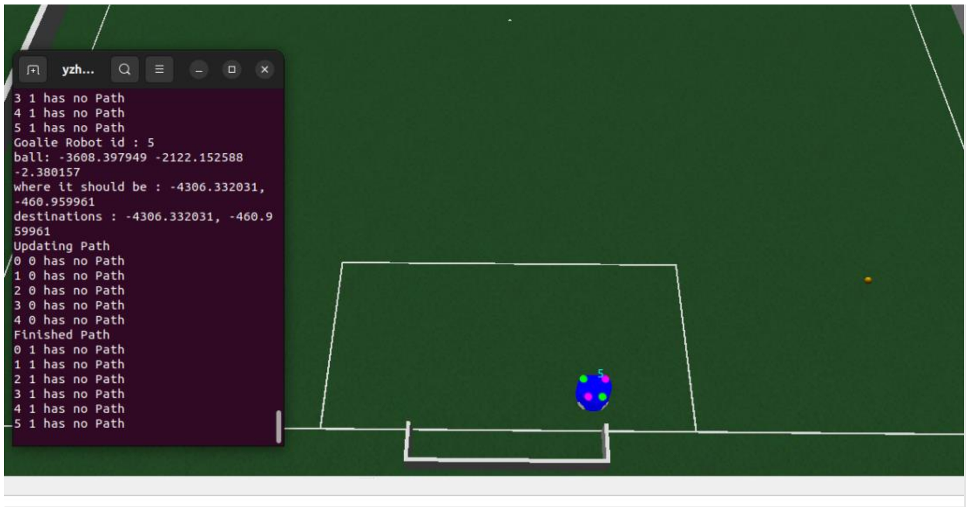
```
3 1 has no Path
4 1 has no Path
5 1 has no Path
Goalie Robot id : 5
ball: -2212.179199 -16.026733
-0.007005
where it should be : -4000.013916,
-3.502522
destinations : -4000.013916, -3.502
522
Updating Path
0 0 has no Path
1 0 has no Path
2 0 has no Path
3 0 has no Path
4 0 has no Path
Finished Path
0 1 has no Path
1 1 has no Path
2 1 has no Path
3 1 has no Path
4 1 has no Path
5 1 has no Path
```

FIGURE 4.11 GOALIE WHEN BALL IS DIRECTLY IN FRONT OF IT



```
3 1 has no Path
4 1 has no Path
5 1 has no Path
Goalie Robot id : 5
ball: -2424.934082 -1637.847656
-0.789299
where it should be : -4107.523926,
-309.781006
destinations : -4107.523926, -309.7
81006
Updating Path
0 0 has no Path
1 0 has no Path
2 0 has no Path
3 0 has no Path
4 0 has no Path
Finished Path
0 1 has no Path
1 1 has no Path
2 1 has no Path
3 1 has no Path
4 1 has no Path
5 1 has no Path
```

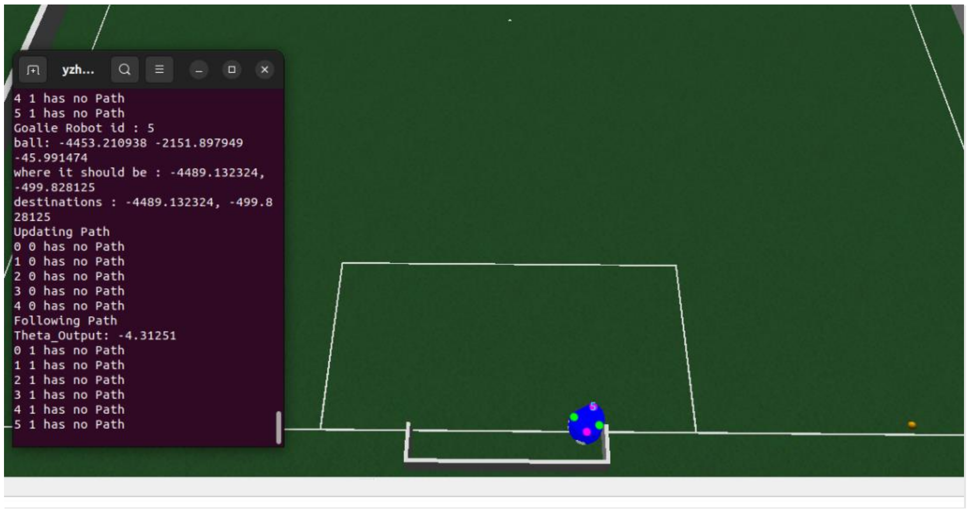
FIGURE 4.12 GOALIE WHEN BALL IS AT A 45 DEGREE ANGLE FROM THE CENTER ON THE RIGHT-HAND SIDE



```
3 1 has no Path
4 1 has no Path
5 1 has no Path
Goalie Robot id : 5
ball: -3608.397949 -2122.152588
-2.380157
where it should be : -4306.332031,
-460.959961
destinations : -4306.332031, -460.9
59961
Updating Path
0 0 has no Path
1 0 has no Path
2 0 has no Path
3 0 has no Path
4 0 has no Path
Finished Path
0 1 has no Path
1 1 has no Path
2 1 has no Path
3 1 has no Path
4 1 has no Path
5 1 has no Path
```

FIGURE 4.13 GOALIE WHEN THE BALL IS AT A 65 DEGREE ANGLE FROM THE CENTER ON THE RIGHT-HAND SIDE

In fact, we have tested the code on both sides of the fields. The results are identical and there is no point in repeating them in the report. The problem arises when there is a corner ball, or the ball is close to the baseline on our side. The outcome of the goalie-blocking-ball tactic will command the robot to drive into one of the posts of our goal because it doesn't consider the collision model of the robot. This bug has already been put on our schedule. It should be fixed in our next code release. Figure 4.14 demonstrates the struggling robot when he tries to drive to the post, but it physically cannot be exactly on point.



```
4 1 has no Path
5 1 has no Path
Goalie Robot id : 5
ball: -4453.210938 -2151.897949
-45.991474
where it should be : -4489.132324,
-499.828125
destinations : -4489.132324, -499.8
28125
Updating Path
0 0 has no Path
1 0 has no Path
2 0 has no Path
3 0 has no Path
4 0 has no Path
Following Path
Theta_Output: -4.31251
0 1 has no Path
1 1 has no Path
2 1 has no Path
3 1 has no Path
4 1 has no Path
5 1 has no Path
```

FIGURE 4.14 GOALIE WHEN THE BALL IS ON THE BASE LINE. THE ROBOT RUNS INTO THE POST. IDEALLY, IT SHOULD STAY ROUGHLY AT THE POSITION WHERE THE ROBOT IN THE PREVIOUS FIGURE IS.

4.3.4 Testing Navigation

The goal of the Navigation Module was to have two main functionalities. The first was to plan the path and the second was to calculate the kinematics to move to each point along the path. Therefore, to test that this goal was reached, we created a way for the input to be the destination (X, Y, and Theta) then to use RRT* to create a path using the inputted X and Y and a robot's starting point. Lastly, we used the points within the path outputted by RRT* with the inputted Theta to create an input for the move-to-point function. This process is then repeated for every point within the path.

To test path planning, the team tested the RRT* function with multiple starting and end goal positions, as well as various amounts of distributed obstacles. The figures below show the blue robot 3 at the coordinate {X = -0.6, Y = 0}. As well as the yellow robots 3 and 1, at the coordinates {X = 0.6, Y = 0} and {X = 1.5, Y = 0} respectively. The plots of the resulting path of the blue robot moving to {X = 1, Y = 0} before and after smoothing are shown in Figures 4.15 and 4.16.

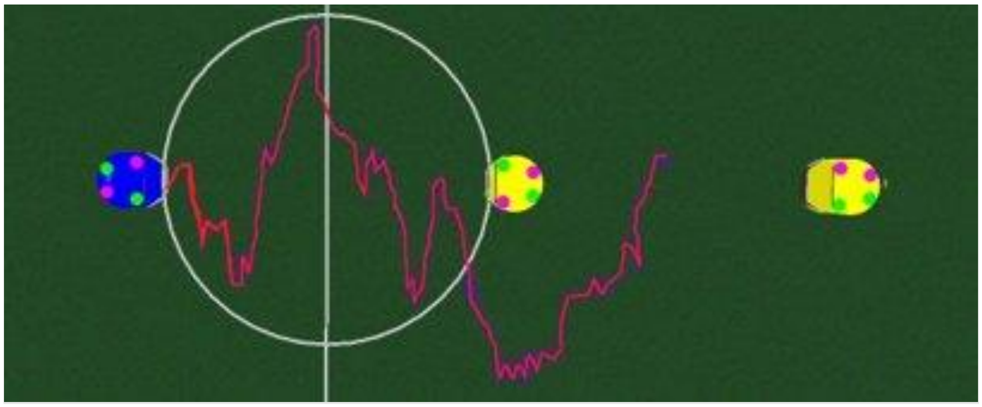


Figure 4.15 Blue robot's path using RRT* before a polynomial curve of best fit is applied.

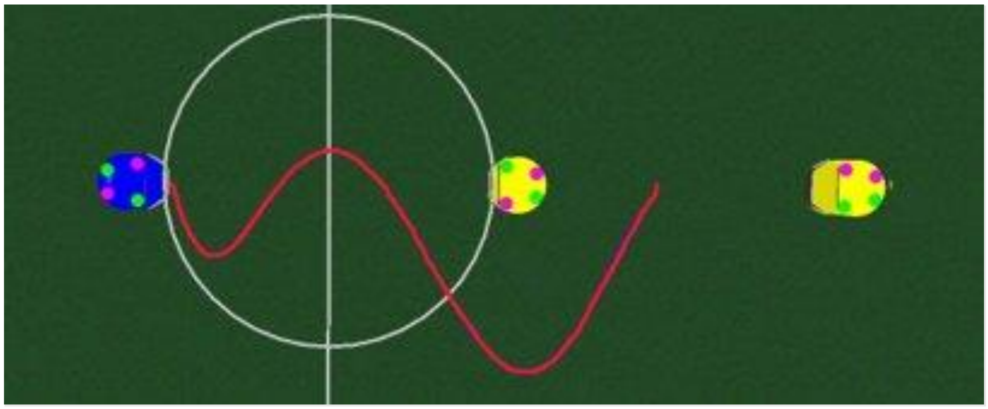


Figure 4.16 Blue robot's path using RRT* after a polynomial curve of best fit is applied.

To test the motion control, the team tested the move-to-point function with destinations consisting of various travel distances and turning angles. Each test would help tune the PID controllers.

CHAPTER 5 DISCUSSION

In the following sections, we will discuss the results of each subsystem as described in Chapter 4.

5.1 MECHANICAL SYSTEM

The mission for this MQP, in its first year, was to create a strong foundation in which future teams could build upon, ultimately leading to an established WPI RoboCup Small Size League team with continuity from year to year, similar in concept to WPI Formula SAE Electric Car MQP. In this way, the progress made this year was crucial to setting the precedent for future teams and allowing their work to be focused on more in-depth testing, validation, and system optimization, rather than development and production. Our current prototype provides a robust basis for experimentation and is designed to support improvements and optimizations with its modular design and capacity for weight reductions as the design becomes more developed and refined.

Our resulting prototype is serviceable in comparison to how other RoboCup teams conceivably looked in their early developmental years. It is unrealistic to compare our progress this first year as a sub-team of three (at full strength) and limited budget with teams who have had the benefit of, in some cases, decades of developmental refinement and experience, with longstanding industry sponsorships, and much larger budgets and team sizes. With this in mind, the success of our sub-team can be best judged with respect to other high functioning, legacy RoboCup programs by the long-term outcomes of progress made in our eventual program's developmental years, and in the short-term by the prototype's ability to garner student interest, university funding, and industry partnerships for the team.

We secured a large partnership with German electronics manufacturer Nanotec that proved vital to the achievement of our goals. Nanotech provided us with an excess of drive and dribbler motors that amounted to thousands of dollars in hardware. Without this contribution, the team would be thoroughly past the budget. In the coming years, the small-size soccer team at WPI will be able to advertise Nanotec via decals on the PLA shell of the robot and on team members' jerseys. This important partnership provided insight into professional relationships in the industry with outside companies, as we were able to discuss needs and specifications for the project and come to a satisfactory agreement for both parties.

The team also gained a smaller contribution from Chinese custom PCB-printing and CNC manufacturer PCBWay upon requesting monetary support. This MQP was advertised on their website's project sponsorship page to garner potential benefactors. The listing has gained over 300 views to date and managed to get the support of PCBWay itself. The support was in the form of an account balance and coupons that were used in the fabrication of the final version of the chipper. The team is currently exploring the relationship with PCBWay further as it will be vital for both speed and quality of manufacturing future renditions of the robot when the need for improvement arises. Working with a CNC manufacturer directly decreases the time required to train and learn the manufacturing process with the positive tradeoff being more time and manpower allocated to design, testing, and system analysis. The ultimate goal is to get all parts from PCBWay for free to maximize this tradeoff.

The system's performance cannot be compared to other teams at present because it has not been truly put to the test to this point. It is hard to quantitatively or qualitatively discuss the prototype's performance in general for this reason. Rather, it would be better to discuss the potential of the prototype to perform. Our system's potential to perform routine game tasks such as maneuvering, kicking, and dribbling is present based on the robust and rudimentary design and construction of the constituent subsystems. Furthermore, a strength of these subsystems is their simplicity in that they will be easy to understand, troubleshoot, and improve by our and future teams. The subsystems' ability to perform in conjunction with one another will be best evaluated in the testing and validation stage in the coming term.

Our prototype consists of similar materials and was fabricated using many similar manufacturing methods to other major teams. However, the similarities generally end there. Major differences in our system compared to other teams relate to design complexity and component selection. Other teams, with the benefit of years and decades have had the ability to refine and fine-tune their designs, to the point of granular detail within subsystems that results in optimal performance. Our more rudimentary designs and solutions are reflective of comparatively minute development, experience, and team size. Similarly, while other teams were able to develop and implement bespoke components such as motors, plungers, and wheels, our team was constrained to consumer grade, mass production components that lacked the performance and refinement of the competitors.

Overall, the success of our system was limited to factors such as budget, experience level, and personnel continuity. As a first-year team, budget and experience were bound to be factors. We often found ourselves opting for options with regards to electromechanical components and manufacturing methods that were not in the best interest of performance, but rather in price. For example, the solenoids we purchased are of considerably lower quality than other team's solutions, yet they still took up a sizable chunk of our budget, forcing us to make further tradeoffs elsewhere that may hinder our prototype's performance potential. Furthermore, our team had minimal fabrication experience, and this translated to setbacks related to part tolerances and interfacing issues. Finally, the sub-team underwent fluctuations in personnel numbers which led to continuity issues.

5.2 ELECTRICAL SYSTEM

The goal of the electrical sub-team MQP was to create the electronics and corresponding firmware layer of the robot to interface with the mechanical and computer science sub-teams' contributions. We can power all the components on all boards created, can actuate the kicking and chipping mechanisms, are able to run the dribbler mechanism with the dribbler motor, can control the drive system with closed-loop feedback, and can communicate with the AI wirelessly. The team also integrated the electronic systems with the mechanical team's drive system, chassis, and ball control module, and the computer science team's codebase to create a working Small Size Soccer Robot Prototype.

Compared to teams in their first year of development at the inception of Small Size League, we are quite far along as we were able to use proven data from years of winning systems. This would not have been possible without the TDPs to reference, and the SSL Discord Server in which league members extend helping hands when needed. Compared to teams that actively play in the Small Size Soccer League, there is a gap that must be closed by the teams that succeed us. While the systems we designed are a solid foundation to build upon and provide all the basic functionality we described, they are not yet up to par with existing teams. Specifically, the drive system must be considerably more robust before the computer science team is able to perform the complex navigation it desires. The ESCs, as mentioned in the coming recommendations section, must be replaced with dedicated motor drivers that perform BLDC motor control for wheeled robots better. The PCB stack can also be consolidated to conserve height and obtain the low center of gravity the mechanical team desires, by combining the kicker and power boards.

On the kicker board we do have more room in the power budget to upgrade the chip lt 3751 and as well as a using temperature as the biggest reason that caused deviation in expected versus real results was due to a few ideas in place one is that a thermometer is required as solenoid heats up its resistance changes and requires a change to the model so when getting the resistance of the solenoid changes and the amount of current and the emptying time of the capacitor changes cause even more deviation from real and expected. This is one of the causes for deviation in the times but also the solenoid is reverse engineered from force calculations as none of the parameters were actually available from the solenoid website and there is no description about how the coil is done so I recommend finding a different distributor as this led to us having to different curves to fit to as one of the solenoids is has a 4 ohm internal resistance instead of the allotted. This requires us to have to have multiple models of solenoids.

5.3 SOFTWARE ARCHITECTURE

Overall, the Computer Science sub-team’s goal was to test and develop a software system capable of playing a complete game of 6v6 soccer. To test the system, we were successfully able to set up *grSim*, our simulator. The in-house software architecture was designed with two main focuses: Strategy and Navigation.

Since most of the tests were done on grSim before transferring onto the actual robots, the simulator worked reasonably well in terms of verifying the functionality of our software. One of the bright sides of grSim was the fact that it provided all the functionality we need based on our design of the robots. However, one of the worst things about grSim was the lack of documentation. Since the team who created and developed grSim was not maintaining the software anymore, there was no other method to learn grSim apart from trying everything ourselves. In addition, the grSim Github repository was not detailed enough to provide guidance. This partially explained why the team got stuck with grSim in the early phases of development. Furthermore, robot models in grSim were based on the creator's design of soccer robots. They did not match ours and we accommodated this by creating transfer functions in the navigation library. There were also other minor mismatches between grSim robots and our actual design, but they could be handled from the software side and should be aware of when testing.

The GameState was used to describe a set of like situations based on ball possession and location. We were able to successfully determine when the field was set up in each of the possible game states. This feature was tested thoroughly by setting up the field in many different configurations and verifying that the correct GameState had been evaluated.

In terms of plays, we set up the foundation for them to be implemented early on by the next team. Individual tactics can be run on the simulator as well as the role assignment, which means the implementation of a full play should be relatively simple, since all the components are developed, they just need to be combined.

We tested our role assignment thoroughly in different GameStates to ensure that the roles being assigned were appropriate for the field's setup. We successfully assigned offensive roles when on offense and defensive roles when on defense. Using a distance-to-ball algorithm, all roles were assigned as we expected. This testing shows that role-assignment performance is successful.

When it came to assigning role tactics, we were limited by time towards the end of the project. For offensive roles, we can successfully determine the appropriate location to travel to for the primary, secondary, and tertiary positions. The destinations that were tested were accurate for both the yellow and the blue team. The destinations put the robots in a position to pass up the field moving towards the goal, or even shoot the ball. By integrating the probability of a successful shot or pass, the role tactics would also be able to make the decision for themselves whether to shoot or pass the ball. The limitation to the offender role tactics was when it came to moving multiple robots. If the GameState was updated before a robot reached its final destination, there could be errors with an incomplete path, causing the robot to continue to move in its most recent direction and not stop. There is further testing to be done to determine the root cause of this error. The primary defender is responsible for shot or pass blocking. To do this they must also go to the correct location, however we expect that when additional defenders are in motion there may be similar errors to the one listed above. On the other hand, the blocking ball tactic for the goalie went as planned. By analyzing the shooting trajectory of the ball while it's in the opponent's possession, the strategy library was able to produce a tactic that allows the goalie to block the ball with the greatest chance. Section 4 included a detailed explanation of how things work. This tactic was also running fine while the program is making strategies for our robots.

To determine the best course of action, we developed a set of probability deducing functions to numerically gauge the likelihood of a positive outcome from a given move. Passing and shooting were two actions that required this kind of evaluation. The probability of a pass being successful was calculated by gauging how long it would take for each robot to move to a point that would intersect with the ball's time driven parametric path.

We tested our navigation module thoroughly. First, we tested path planning with varying amounts of obstacles to perfect obstacle avoidance. Then, we tested the motion control at different speeds, locations, and angles to guarantee a motion that was both controlled and efficient. In highlight of our successes, we were able to plan paths that were optimal and ensured obstacle avoidance and we were able to use the robot's kinematics to go from linear velocities to four independent wheel velocities. As for the limitations, the overall movement system was the biggest hindrance. To be specific, the way the PID controllers were used was not reliable. As expected, the error for each PID controller would reset when processing the next point along the path. This would cause the motion to start then stop throughout the path. This was later bypassed by driving at slow speeds and using a directional coordinator. Meaning that as the next point along the path was processed the direction in which the robot drives was updated. This performed well; however, the precision of the path was not the best and the motion was susceptible to oscillation near the end of the path as the robot began to stop.

While we were not able play a complete game of soccer, we were able to set a strong foundation that can be built upon by teams. We were able to set up a simulator that met the requirements of testing a 6v6 soccer game, as well as individual functionality components. Using information sent from grSim, we were able to calculate and smooth ball and robot velocities as well as analyze the position of the ball and all 12 robots to accurately determine the state of the game. We developed the function to run individual tactics successfully, each of which is comprised of a move, skill threshold, and skill. Once the GameState was determined, we can assign each robot's offensive or defensive position based on their distance to the ball and the goal. Due to limitations in time, we were not able to complete every role's tactics, however we can identify proper destinations for primary, secondary, and tertiary offense, primary defense, and the goalie. This capability can be replicated to complete the destination calculation for other positions in the future. We were also able to create probability functions that calculate the probability of a successful pass or shot, which can eventually be integrated with role tactics to develop decision making algorithms for different positions. The functions created to find the best pass option will be used when implementing tactics that require a robot to move to an optimal passing position. Our robots are able to move to their dedicated positions while isolated using our navigation module. The navigation module ensures proper obstacle-avoiding paths and the ability to follow the generated path. However, there are limitations when it comes to GameState being reevaluated before a path is completed that sometimes prevent the robots moving to the correct locations.

CHAPTER 6 CONCLUSION

Over the course of four terms, this first-year MQP team researched, designed, developed, analyzed, fabricated, assembled, and integrated a prototype robot that has set the groundwork for an upcoming WPI RoboCup Small Size League program. The team was able to create a robot in a modular setup as well as develop a codebase with the necessary class structure to ensure all strategies and mechanics are tested and fine-tuned. The robot design consists of a mechanical system, an electrical system, and the software system to enable full autonomy with a master to agent relationship.

The mechanical system includes a chassis consisting of a CNC machined base plate, top plate, and drive motor mounts, a ball control assembly including a bespoke dribbler motor and solenoid mount superstructure and custom fabricated kicker, dribbler, and chipper, and an offset omnidirectional wheel configuration independently driven by high performance brushless DC motors. One of the key outcomes of this project is this prototype, which is the physical basis for all testing and validation for the foreseeable future as the project progresses. Our contribution to this hopefully ongoing project is the fundamental theoretical, physical, and experimental knowledge of the system and the groundwork for WPI's eventual permanent RoboCup Small Size League team.

The electrical system is broken down into the hardware and the firmware. The hardware consists of the DF45 drive motors and encoders, the DF20 dribbler motors and encoders, the solenoids needed for chipping and kicking the ball, and the PCB stack, which contains the Processor, Power, and Kicker boards. The firmware was developed in Microchip Studio using an Atmel Start project. The project consists of 13 drivers written in C and C++ within a C++ environment for easy integration with the software system.

The software system includes a well-established simulated environment that provides a safe way to test the algorithms and structures within the software architecture without using physical robots. This simulator can provide all the information available during a regulation match, as well as allow for control of all robots on the field through the same communication protocol.

The tested software architecture is mainly comprised of two focuses: strategy and navigation, all written in a C++ environment. The strategy component interprets the game information provided by the simulator and then provides the structure of the plays to be run. A play can be further broken down into roles, tactics, and skills. The navigation aspect refers to the motion of the agents when carrying out a play. It is responsible for a robot to drive to its destination in a way that avoids obstacles.

At a base level, the MQP is an extended period of experiential learning that capstones the WPI education. The team learned a lot throughout this process including what it takes to start something from scratch. What may seem to be a menial or trivial task can prove to be much more complex and intensive than at face value. We learned how to approach these complex tasks and leverage our strengths and weaknesses to get the job done. We also learned how to manage unexpected circumstances and how to be flexible in the face of these circumstances for the betterment of the team.

CHAPTER 7 FUTURE RECOMMENDATIONS

7.1 MECHANICAL RECOMMENDATIONS

For further improvement, it is recommended the system be rigorously tested and verified to ensure that the team is getting the performance desired from their robots. From there, up to two full teams of seven optimized robots each are recommended to be fabricated so that game state testing and validation can be performed at full scale, perhaps by working up to this number through increasingly larger simulated competitions. Furthermore, it is recommended that future teams investigate and solidify a test field and camera vision system mount solution so that testing and validation can be reliably performed at will. These recommendations arise from activities that the team was unable to perform due to limitations related to time, budget, equipment or otherwise. Additionally, the ME subteam was able to scale up production of two additional prototypes using 3D printing in D-term, effectively doubling output in half of the time. It is recommended that future teams take advantage of these campus resources for rapid prototyping as it is cheaper and quicker while making the robots lighter as compared to the aluminum counterpart.

7.2 ECE RECOMMENDATIONS

Throughout a rigorous year of development, the team discovered several areas of improvement which were not acted on due to lack of time, funding, or a combination of the two. Should this project be continued, these recommendations would serve as an important starting point for progress on the electrical and computer engineering side of the project.

7.2.1 BOARD DESIGN

In the future, it's recommended using different connectors and integrating the board into a slightly larger form factor closer to 18 centimeters. This can be done by importing a DWG file to Altium. The team didn't learn of this until late in the MQP. This means a 18cm diameter circle can be put into the chassis' four boards.

7.2.2 MOTOR DRIVERS

The nature and behavior of the ESCs was undesirable for developing the motor controller, as well as general operation limitations. Due to the ESCs having a dead-band, their behavior was not linear and made tuning the motor controller difficult. Additionally, the full potential and precision available from the PWM signals was not able to be used due to the limited range of duty cycles the ESCs accept. It is recommended that a switch be made from ESCs to motor controller ICs to allow for a linear behavior and to use the full range available from the PWM signals.

7.2.3 MOTOR CONTROLLER DERIVATIVE TERM & I-LIMIT

Due to the procurement of encoders in the last week of MQP and the figuring out of ESCs, the team was only able to get the motor controller working unloaded with only proportional and integral terms in the 4 days leading up to Project Presentation Day. To get the controller to stabilize and converge under the weight of the chassis, we recommend adding a derivative term to the controller. This will account for acceleration and therefore decrease the spikes in effort as the controller works overtime. Furthermore, the I-limit of the controller should be implemented to ensure the controller doesn't tend to positive or negative infinity. First, not having a limit will accumulate "integral windup" and make the controller response far larger than it should be, decreasing its stability. Second, as soon as the ESC receives a command out of its calibrated range, the motor will immediately halt, which could be very dangerous in the middle of operating (the robot may tip depending on the speed).

7.2.4 Adding a Thermistor to Solenoids

When used for extended periods of time, the solenoids heat up quickly as they dissipate a lot of energy. The resistor in this case would be used to calculate a new model for the solenoid based on temperature. The heat of the solenoid affects how current flows through it which then affects the behavior of the plunger at known voltages. We would need to compensate for this on the hardware side to allow higher error tolerance for the software. This could result in failed passes due to the ball being kicked too fast.

7.2.5 BNO085 Integration

We recommends the BNO085's use be fused with SSL vision and the kinematic model using the Kalman Filter to increase precision of motion as navigation becomes more fine-tuned. Although its development took a lot of time and effort, the BNO085 (IMU) is not currently in use by the main firmware loop. We made this decision because the vision system was not set up until the last week of MQP. Fortunately, for the foreseeable future, the shared vision system of the Small Soccer League provides enough data to determine the heading of the robot with more accuracy than an IMU could. SSL receives a still image of the entire field, runs a processing algorithm, and returns information of the current game state, chiefly the positions and headings of each robot.

As the continuing team tests and develop the robots' navigation further, higher precision between frames will be desired. The update frequency of the vision system is only 60 Hz, the typical shutter rate of a camera. This low frequency heading update means the robot, in between frames, must rely on its own kinematics without the complementary checking of the IMU, which will lead to considerable error depending on the speed of the robot. There will be two cases to implement the Kalman filter with the IMU with different weights, determined experimentally. The first case will occur when the robot receives a new heading update with SSL outputs, the kinematics, and the IMU as its inputs. The second case (in which the IMU's data will be weighted heavier) will occur in between SSL frames, with kinematics and the IMU as the only positional inputs.

7.2.6 ECE Team Members

A major challenge we encountered throughout the project was the number of team members knowledgeable in certain facets of the project. These facets can be broken down into three areas of the ECE curriculum: power, microelectronics, and computer engineering. While all team members were well learned with computer engineering, and only one member experienced with microelectronics, none of the members were experienced with high voltage and high current systems prior to starting or joining the MQP. One of the team members had to learn about power electronics during the MQP, and while the power and kicker boards were able to be designed to a sufficient degree, it was still an undesirable situation. It is recommended to have a minimum of three students with proficient knowledge in electronic computer engineering for the next year's team, with at least one student having taken power electronics courses, one student having taken microelectronics courses, and one student having taken computer engineering courses. This will help divide the workload among the members and allow them to focus on their areas of expertise.

7.2.7 Transition to MPLab from Microchip Studio

The firmware was originally developed in Microchip Studio, but we ran into router complications as it is not supported on Ubuntu. Switching over to a different integrated development environment (IDE) called MPLab, will enable one to work with the router whilst using Ubuntu. This will require different compilers to be downloaded and it will have a different design structure built more on applications instead of tools that solve most of the problems you will find in Microchip Studio.

7.3 SOFTWARE RECOMMENDATIONS

Throughout our development, we have noted areas where improvements could be made. These recommendations pertain to the navigation module, multi-robot movement, plays, and strategy in general.

7.3.1 Navigation

There are still some improvements to be made in both the path planning and the motion control areas of the navigation module. To improve the accuracy of paths, a Kalman filter should be implemented. The estimation of the other team's upcoming position can be used to better tune the path planning. We have also created a way to check for oncoming collisions. However, this was never implemented. This method of collision detection can be used to recalculate paths if necessary.

As for motion control, the PID controllers still need some work. The controllers use the distance between the robot's current position and the robot's end goal as well as the distance left to travel to the robot's next point along the path to adjust the linear velocities of the robot. This has been the best implementation for the PID controllers; however, the tuning was never finished to create a stable outcome. Another area that must be investigated is the update rate of each robot's position. Since the update rate of the robot's position is limited to the framerate of the camera, a Kalman filter can also be used to predict where the robot is heading to. Lastly, there needs to be some error handling in the navigation module. This was later found to be essential for efficient testing of the rest of the strategy module.

7.3.2 Multi-Robot Movement

To alleviate issues with running role tactics at the same time where multiple robots are moving towards different destinations, there needs to be some way to ensure that the robots do not get off course and continue movement towards the edge of the field. Two options we have considered are checking for path completeness, reducing the frequency of reevaluation and including stops after reevaluation. The first option would involve making sure all paths are complete before performing a reevaluation of game state and role assignments. This would ensure that all robots would arrive at their desired destinations before a switch occurs. To help with this, perhaps speeds could be adjusted to make it so all robot get to their destinations at the same time in order to avoid having a robot sit still during game play. The other option would be to reduce the frequency at which gamestate is analyzed, and therefore reduce the rate at which roles are reassigned. This would give the robots time to get to their destinations, or at least closer, before they switch roles. If the frequency of role assignment is reduced, it also may allow a robot to remain in it’s role longer, meaning less overall movement around the field and cleaner gameplay. While these options both involve changing the strategy code, there also could be a solution found in the navigation module to account for what to do when paths are not complete, or how to reevaluate paths to readjust to new destinations being found.

7.3.3 Reinforcement Learning

Given the nature of soccer, where it is likely that other teams will show patterns in the strategy decisions their software makes, it would be prudent to implement reinforcement learning. This would track the outcome of each play implemented and give rewards for positive outcomes like regaining the ball or scoring goals and give negative rewards for negative outcomes like losing the ball or goals scored against us. These rewards could be used to rank the available plays for a given gamestate and help determine which should be used. Initially ranking plays based on a given database but slowly weighting the rewards that are awarded throughout a match more heavily would allow for the existing precedent to stand while being continuously modified by the newer patterns being displayed.

7.3.4 Scouting Unit

Since other teams have been active in the league for years, there is a good amount of existing footage of games. This scouting data could be used to our team's advantage if we set up the code infrastructure to be able to learn from these past games. This would entail creating software capable of analyzing the patterns of other teams and storing that data to be used when our team is actually competing.

REFERENCES

ATSAMV71-XULT. (2022). Microchip.com. https://www.microchip.com/en-us/development-tool/atsamv71-xult

Chinenov, T. (2019, February 13). *Robotic Path Planning: RRT and RRT* - Tim Chinenov - Medium*. Medium; Medium. https://theclassytim.medium.com/robotic-path-planning-rrt-and-rrt-212319121378

Code of Ethics | National Society of Professional Engineers. (2013). Nspe.org. https://www.nspe.org/resources/ethics/code-ethics#:~:text=1%20Engineers%20shall%20hold%20paramount%20the%20safety%2C%20health%2C,...%205%20Engineers%20shall%20avoid%20deceptive%20acts.%20

Gaspar, D. (2015, May 15). *Goalkeeper positioning and where to stand are important to Angles.* Keeperstop.com. https://www.keeperstop.com/goalkeeper_drills-angles_positioning-angle_play_and_goalkeeper_positioning

Martinez-Gomez, L., Moneo, F., Sotelo, D., Soto, M., Weitzenfeld, A., & Mx, A. (n.d.). *VII SBAI/ II IEEE LARS. São Luís, setembro de 2005 DESIGN AND IMPLEMENTATION OF A SMALL SIZE ROBOCUP SOCCER TEAM*. https://weitzenfeld.robolat.org/wp-content/uploads/2015/01/Weitzenfeld10691.pdf

Page, Noah. "Small Size Soccer Robots: The Design of a Custom Embedded System for Use in The RoboCup SSL Competition ." *WPI*, 2022, www.wpi,edu.

RoboCup Federation. "A Brief History of Robocup." *RoboCup Federation Official Website*, https://www.robocup.org/a_brief_history_of_robocup.

Simrock, S. (2019). Control theory. *From Vehicles to Grid to Electric Vehicles to Green Grid.* http://cds.cern.ch/record/1100534/files/p73.pdf

Zickler, S., Bruce, J., Biswas, J., Licitra, M., & Veloso, M. (n.d.). *CMDragons 2009 Extended Team Description*. https://www.cs.cmu.edu/~mmv/papers/09robocup-cmdragons.pdf

APPENDICES

Noah’s Packet Protocol

TABLE 0.1 NPP V1.2

Noah’s Packet Protocol v1.2							
Byte 0: Information Byte							
Bits 7:4: Reserved				Bits 3:0: Target Robot ID Number			
Bit 7: Set as 0	Bit 6: Set as 0	Bit 5: Set as 0	Bit 4: Set as 0	Bit 3: Target Robot ID MSB	Bit 2: Target Robot ID	Bit 1: Target Robot ID	Bit 0: Target Robot ID LSB
Byte 1: Drivetrain Motor 0 Data Byte 0							
Bit 7: Drive Motor 0 Data	Bit 6: Drive Motor 0 Data	Bit 5: Drive Motor 0 Data	Bit 4: Drive Motor 0 Data	Bit 3: Drive Motor 0 Data	Bit 2: Drive Motor 0 Data	Bit 1: Drive Motor 0 Data	Bit 0: Drive Motor 0 Data LSB
Byte 2: Drivetrain Motor 0 Data Byte 1							
Bit 15: Drive Motor 0 Data MSB	Bit 14: Drive Motor 0 Data	Bit 13: Drive Motor 0 Data	Bit 12: Drive Motor 0 Data	Bit 11: Drive Motor 0 Data	Bit 10: Drive Motor 0 Data	Bit 9: Drive Motor 0 Data	Bit 8: Drive Motor 0 Data
Byte 3: Drivetrain Motor 1 Data Byte 0							
Bit 7: Drive Motor 1 Data	Bit 6: Drive Motor 1 Data	Bit 5: Drive Motor 1 Data	Bit 4: Drive Motor 1 Data	Bit 3: Drive Motor 1 Data	Bit 2: Drive Motor 1 Data	Bit 1: Drive Motor 1 Data	Bit 0: Drive Motor 1 Data LSB
Byte 4: Drivetrain Motor 1 Data Byte 1							
Bit 15: Drive Motor 1 Data MSB	Bit 14: Drive Motor 1 Data	Bit 13: Drive Motor 1 Data	Bit 12: Drive Motor 1 Data	Bit 11: Drive Motor 1 Data	Bit 10: Drive Motor 1 Data	Bit 9: Drive Motor 1 Data	Bit 8: Drive Motor 1 Data
Byte 5: Drivetrain Motor 2 Data Byte 0							
Bit 7: Drive Motor 2 Data	Bit 6: Drive Motor 2 Data	Bit 5: Drive Motor 2 Data	Bit 4: Drive Motor 2 Data	Bit 3: Drive Motor 2 Data	Bit 2: Drive Motor 2 Data	Bit 1: Drive Motor 2 Data	Bit 0: Drive Motor 2 Data LSB
Byte 6: Drivetrain Motor 2 Data Byte 1							
Bit 15: Drive Motor 2 Data	Bit 14: Drive Motor 2 Data	Bit 13: Drive Motor 2 Data	Bit 12: Drive Motor 2 Data	Bit 11: Drive Motor 2 Data	Bit 10: Drive Motor 2 Data	Bit 9: Drive Motor 2 Data	Bit 8: Drive Motor 2 Data

MSB							
Byte 7: Drivetrain Motor 3 Data Byte 0							
Bit 7: Drive Motor 3 Data	Bit 6: Drive Motor 3 Data	Bit 5: Drive Motor 3 Data	Bit 4: Drive Motor 3 Data	Bit 3: Drive Motor 3 Data	Bit 2: Drive Motor 3 Data	Bit 1: Drive Motor 3 Data	Bit 0: Drive Motor 3 Data LSB
Byte 8: Drivetrain Motor 3 Data Byte 1							
Bit 15: Drive Motor 3 Data MSB	Bit 14: Drive Motor 3 Data	Bit 13: Drive Motor 3 Data	Bit 12: Drive Motor 3 Data	Bit 11: Drive Motor 3 Data	Bit 10: Drive Motor 3 Data	Bit 9: Drive Motor 3 Data	Bit 8: Drive Motor 3 Data
Byte 9: Dribbler Motor Data Byte 0							
Bit 7: Dribbler Motor Data	Bit 6: Dribbler Motor Data	Bit 5: Dribbler Motor Data	Bit 4: Dribbler Motor Data	Bit 3: Dribbler Motor Data	Bit 2: Dribbler Motor Data	Bit 1: Dribbler Motor Data	Bit 0: Dribbler Motor Data LSB
Byte 10: Dribbler Motor Data Byte 1							
Bit 15: Dribbler Motor Data MSB	Bit 14: Dribbler Motor Data	Bit 13: Dribbler Motor Data	Bit 12: Dribbler Motor Data	Bit 11: Dribbler Motor Data	Bit 10: Dribbler Motor Data	Bit 9: Dribbler Motor Data	Bit 8: Dribbler Motor Data
Byte 11: Kicker Data Byte							
Bit 7: Kicker Data MSB	Bit 6: Kicker Data	Bit 5: Kicker Data	Bit 4: Kicker Data	Bit 3: Kicker Data	Bit 2: Kicker Data	Bit 1: Kicker Data	Bit 0: Kicker Data LSB
Byte 12: Chipper Data Byte							
Bit 7: Chipper Data MSB	Bit 6: Chipper Data	Bit 5: Chipper Data	Bit 4: Chipper Data	Bit 3: Chipper Data	Bit 2: Chipper Data	Bit 1: Chipper Data	Bit 0: Chipper Data LSB
Bytes 31:13: Reserved (Should all be set as 0)							

Minimum Trace Width Calculations

TABLE 0.2 MINIMUM INTERNAL TRACE WIDTHS

Minimum Internal Trace Widths						
Current	Weight(oz/ft2)	Temp Delta	Initial Temp	Area	Width(mm)	Width(mm)
1	1	100	10	10.480893	7.622467637	0.193610678

5	1	75	25	12.10474263	8.803449184	0.223607609
10	1	75	25	20.00793679	14.55122676	0.36960116
30	1	75	25	44.3726347	32.27100705	0.819683579
50	1	75	25	64.26219563	46.73614227	1.187098014
1	2	75	25	3.768793192	1.370470252	0.034809944
5	2	75	25	12.10474263	4.401724592	0.111803805
10	2	75	25	20.00793679	7.275613379	0.18480058
30	2	75	25	44.3726347	16.13550353	0.40984179
50	2	75	25	64.26219563	23.36807114	0.593549007
1	3	75	25	3.768793192	0.913646834	0.02320663
5	3	75	25	12.10474263	2.934483061	0.07453587
10	3	75	25	20.00793679	4.850408919	0.123200387
30	3	75	25	44.3726347	10.75700235	0.27322786
50	3	75	25	64.26219563	15.57871409	0.395699338
1	4	75	25	3.768793192	0.685235126	0.017404972
5	4	75	25	12.10474263	2.200862296	0.055901902
10	4	75	25	20.00793679	3.63780669	0.09240029
30	4	75	25	44.3726347	8.067751763	0.204920895
50	4	75	25	64.26219563	11.68403557	0.296774503
1	6	75	25	3.768793192	0.456823417	0.011603315
5	6	75	25	12.10474263	1.467241531	0.037267935

10	6	75	25	20.00793679	2.42520446	0.061600193
30	6	75	25	44.3726347	5.378501175	0.13661393
50	6	75	25	64.26219563	7.789357046	0.197849669

TABLE 0.3 MINIMUM EXTERNAL TRACE WIDTHS

Minimum External Trace Widths						
Current	Weight(oz/ft2)	Temp Delta	Initial Temp	Area	Width(mm)	Width(mm)
1	1	100	10	4.028881559	2.930095679	0.07442443
5	1	75	25	44.16781818	32.12204958	0.815900059
10	1	75	25	114.8999219	83.56357955	2.122514921
30	1	75	25	522.8992462	380.2903609	9.659375167
50	1	75	25	1057.831003	769.3316384	19.54102362
1	2	75	25	4.797438196	1.744522981	0.044310884
5	2	75	25	44.16781818	16.06102479	0.40795003
10	2	75	25	114.8999219	41.78178978	1.06125746
30	2	75	25	522.8992462	190.1451805	4.829687583
50	2	75	25	1057.831003	384.6658192	9.770511808
1	3	75	25	4.797438196	1.16301532	0.029540589
5	3	75	25	44.16781818	10.70734986	0.271966686
10	3	75	25	114.8999219	27.85452652	0.707504974
30	3	75	25	522.8992462	126.7634536	3.219791722
50	3	75	25	1057.831003	256.4438795	6.513674538

1	4	75	25	4.797438196	0.87226149	0.022155442
5	4	75	25	44.16781818	8.030512396	0.203975015
10	4	75	25	114.8999219	20.89089489	0.53062873
30	4	75	25	522.8992462	95.07259023	2.414843792
50	4	75	25	1057.831003	192.3329096	4.885255904
1	6	75	25	4.797438196	0.58150766	0.014770295
5	6	75	25	44.16781818	5.353674931	0.135983343
10	6	75	25	114.8999219	13.92726326	0.353752487
30	6	75	25	522.8992462	63.38172682	1.609895861
7.3.4.1.1 50	*7.3.4.1.2 6*	*7.3.4.1.3 75*	*7.3.4.1.4 25*	*7.3.4.1.5 1057.831003*	*7.3.4.1.6 128.2219397*	*7.3.4.1.7 3.256837269*

7.4 LT 3750 CALCULATIONS

CONSTANTS AND DEFINITIONS

$$ratio\ of\ coils\ primary\ to\ seco\ ndary\ =\ N\ =\ 10$$

$$v_0\ =\ voltage\ out$$

$$I_{Pk}\ =\ peak\ current$$

$$transistor\ drive\ voltage\ =\ v_{ds}\ =\ 100V$$

Constants that are used in equations and are underdefined in the following equations.

DETERMINING AVG CHARGING CURRENT

$$I_A = \frac{I_{Pk^*} v_t}{2(v_0 + N \times V_t)}$$

$$I_A = \frac{22.2 * 12}{2(249 + 10 \times 12V)}$$

$$average\ current\ =\ I_A\ =\ 7.9937$$

It is recommended solving for this value as it plays into the simulation as it plays into simulating the simulation of the temperature of the circuit. Ipk is pulled from the transistor datasheet as its peak current not it continuous current.

SETTING CURRENT LIMIT RESISTANCE VOLTAGE

$$.78v/mohm \ * 8\, mohm \ = \ 6.24V$$

By picking a 8 mohm sets the stages minimum voltage on the pulse this plays into the efficiency of the power transfer between the stages.

TRANSFORMER PRIMARY INDUCTANCE

$$Lpri \ \geq \ \frac{v_{out} \ * 1us}{N * I_{pk}}$$

$$Lpri \ \geq \ \frac{250 * 1us}{10 * 22.2}$$

$$Lpri \ \geq \ 1.261$$

Lpri is an output of the primary stage coil to secondary in this use case we have to select a transformer coil. From this we selected the 2034-ald which has 4:1 coil ratio.

OUTPUT DIODE SELECTIONS

$$Peak\, repetitive\, forward\, current\, rating \ \geq \ \frac{22.2}{10}$$

$$prfcr \ > \ \ = \ 2.22$$

$$Peak\, reverse\, repetitive\, voltage\, rating \ \geq \ vout \ + \ vtrans * N$$

$$peak\, reverse\, repettive\, voltage\, rating \ \geq \ 370.78$$

Recommended selection from this diodes found on Digi key is the murs160 from diode incorporated. It has a property that make the second stage more efficient that allows for a more efficient power transfer and generating less heat.

SETTING THE TARGET VOLTAGE

$$v_0 = \left(1.24v \cdot \frac{R_v}{R_{(BC_I)}} * N\right) - v_d$$

$$v_0 = \left(1.24v \cdot \frac{60.4}{2.49} * 10\right) - 50$$

$$v_0 = 250.78$$

CAPACITOR DISCHARGE TIME AND MONITORING CIRCUITRY

VOLTAGE DIVIDER

$$vin/vout = r2/(r1 + r2)$$

$$vin/vout = (10M + 133k)/133k$$

$$vin/vout = 76.67$$

The voltage divider moves the ratio that moves the 250V limit of the capacitors to a 3.3v maximum of the ADC.

RC FILTER

$$RC = 1 * 10^3 * .1 * 10^{-6}$$

$$RC = .1ms$$

$$2piRC = .2pi$$

$$1/(2piRC) = 1519hz$$

An RC filter created to remove the noise of the charging circuit from the measurement of the capacitor charge detection.

CAPACITOR DISCHARGE CHARACTERISTICS

4 OHM 900 TURN SOLENOID

$$\tau = RC$$

$$\tau \; = \; 4 \cdot 3.6 \cdot 10^{-3}$$

$$\tau \; = \; 4 \cdot 3.6 \cdot 10^{-3}$$

$$\tau \; = \; 0.0144$$

$$adequate\,'disch \arg e'\; time \; = -\, 5\tau \; = \; .072$$

$$V_C \; = \; V_s \cdot e^{-\frac{t}{RC}}$$

$$V_C \; = \; 250 \cdot e^{-\frac{t}{4 \cdot 3.6 \cdot 10^{-3}}}$$

This is the approximate amount of time until a capacitor is adequately discharged in the 4 ohm consideration is an insight into the power drop off upon activation. These are the equations we use to model the discharge and to set the timings for current discharge. Problems exist as in the later stage of testing we ran into the issue of temperature changing the resistance of the solenoid.

12 OHM 3510 TURN SOLENOID

$$\tau \; = \; RC$$

$$\tau \; = \; 12 \cdot 3.6 \cdot 10^{-3}$$

$$\tau \; = \; .0432$$

$$V_C \; = \; 250 \cdot e^{-\frac{t}{12 \cdot 3.6 \cdot 10^{-3}}}$$

This calculation is like the above but has a 4 ohm 900 amperes turn except we have a second solenoid that was configured differently and has slightly different properties. These were characterized to be modeled and used on the robot as well.