Autonomous Soccer Robot

MCEN 5115: Mechatronics and Robotics 1 Final Project

Team: Durango Mangoes

December 9, 2021

Contents

1 Introduction

Inspired by the popular video game Rocket League, the University of Colorado Fall 2021 Mechatronics class attempted to build autonomous robots to compete with other teams in one-on-one robot soccer. The challenge was to build a robot, within specified parameters and budget constraints, that could autonomously navigate a 12' x 8' playing field and push a foam soccer ball into its opponents goal while defending its own goal against an opposing robot.

This report details the design, construction, programming, and performance of the robot for team Durango Mangoes. The design pursued was a three-level rectangular robot controlled by a Raspberry Pi (RPi) single-board computer and two Arduino Mega 2560 microcontrollers. Mounted on the top level of the robot was a pan-tilt camera system for tracking the ball. The middle level contained the RPi, Arduinos, batteries, and an accelerometer and electronic receiver for tracking the robot's position in the playing field. The bottom level of the robot contained its drive system, consisting of four Mecanum wheels driven by four brushed DC gear motors.

Overall, the envisioned robot functionality and performance was not fully achieved by the time of the runoff. The team was unable to get the camera system working effectively and eventually decided to remove it as it was unnecessary for the robot's motion planning. All other subsystems as well as the robot's path planning algorithm functioned well independently, but the team had trouble with system integration. In particular, the Arduino coupled with the receiver and accelerometer had problems communicating with the Raspberry Pi, giving inaccurate position data which caused the robot to often move incorrectly in the field. The resulting performance during competition was a first round win on penalty kicks but subsequent elimination from the tournament after two losses.

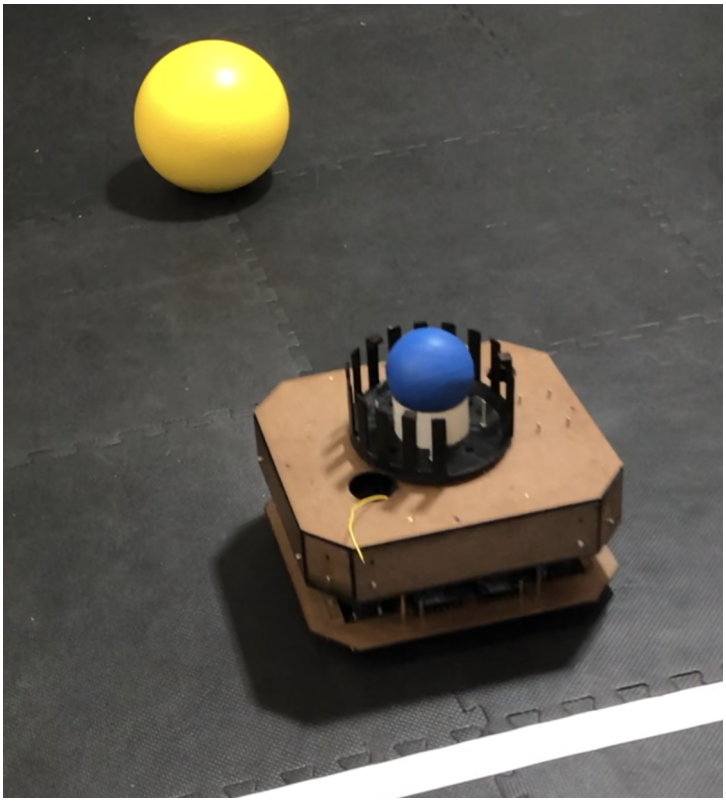

Figure 1: Robot in playing field with foam soccer ball

2 Design Process

The design process for the robot was iterative, but most major design decisions were made at the start. The team envisioned a fast and nimble robot that would be able to track the ball without needing to rotate the whole chassis, and would move quickly and omnidirectionally within the playing field. The team also wanted the robot to be close to the maximum allowable size to give it the best chance of hitting the ball on both offense and defense.

With these design goals in mind, a three-level design was developed to provide ample space for all components, as shown in Figure 2. To keep the robot lightweight yet durable, the chassis was constructed from Eucaboard and brass standoffs. Meanwhile, much of the team's budget was spent on a powerful drive system in hopes of being able to move quicker in the field than opposing robots. The design used omnidirectional Mecanum wheels to allow for the greatest maneuverability within the field and provide flexibility in motion planning.

Although not used in the final design, a pan-tilt camera system was mounted on top of the robot, as shown in Figure 2. The intention of the camera system was to rotate and track the ball without having to rotate the entire robot. Meanwhile, the robot's rotational position would be tracked by an accelerometer mounted at the center of the second level. The translational position—as well as the opposing robot's position and a secondary source on the ball's position—would be provided by an electronic transceiver that would receive transmitted data from the camera system mounted above the playing field.

Further details on the design decisions for every aspect of the robot are discussed in Sections 3 - 6.

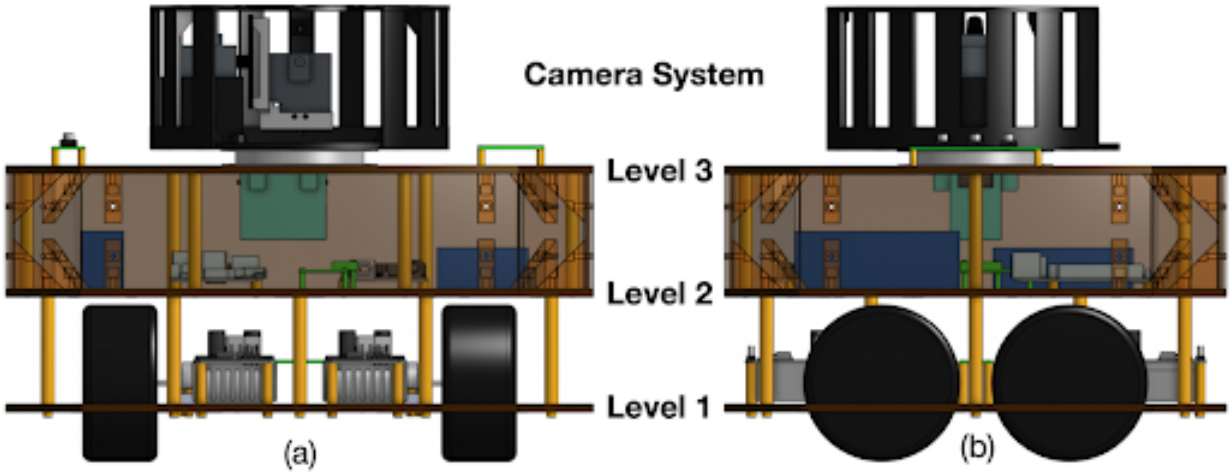

Figure 2: Front view (a) and side view (b) of robot CAD model with the bumpers and bumper mounts at 25% transparency

3 Mechanical Design

The robot was designed using SolidWorks CAD software. At 14" wide and 12" long, the robot was designed to take advantage of maximum allowable two-dimensional footprint. To avoid catching its corners on the other robot or goals, the corners of the otherwise rectangular footprint were cut to 135°, making the robot an elongated octagonal shape with 2.5" corner cuts. The robot consisted of three levels to provide space for all of the electrical and mechanical components. See Figure 2 for the labeling of each layer of the robot that will be used throughout the paper.

3.1 Chassis

The three levels of the robot were made out of 3/16" Eucaboard and were connected to each other using M3 brass standoffs, of various lengths, and M3 screws.

Level 1 of the robot was designed to house four motors, four motor drivers, and four 97 mm Mecanum wheels along with their soldered printed circuit board (PCB). The bottom of the Level 1 plate was 1.22" off the ground. The wheelbase of the robot was 4.5" and the track width was 8.56". Four large rectangular holes were cut out of the Level 1 plate to accommodate the wheels. Numerous small circular holes were cut into the board to accommodate the hole patterns for the motor mounts, drivers, PCB, and Level 2 standoffs. The eight standoff sections supporting the Level 2 platform consisted of a combination of M3 brass standoffs totaling 6 5mm (2.56") each.

Level 2 was designed to house two Arduinos, one Raspberry Pi, the accelerometer PCB, three batteries of various sizes, and the perimeter bumper. The bottom of the Level 2 plate was 65 mm above Level 1. Two large circular holes were cut into the Level 2 plate to allow the wires from the motor drivers and PCB on Level 1 to connect to the motor control Arduino on Level 2. Numerous smaller circular hole patterns were cut into the Level 2 plate to attach all the components and the Level 3 standoffs. The six standoff sections supporting the Level 3 platform consisted of a combination of M3 brass standoffs totaling 70 mm (2.76") each. The perimeter bumper, also made of Eucaboard, covered the 70 mm gap between the Level 2 and 3 platforms around the perimeter of the robot and was mounted to Level 2 using 3D-printed mounting brackets.

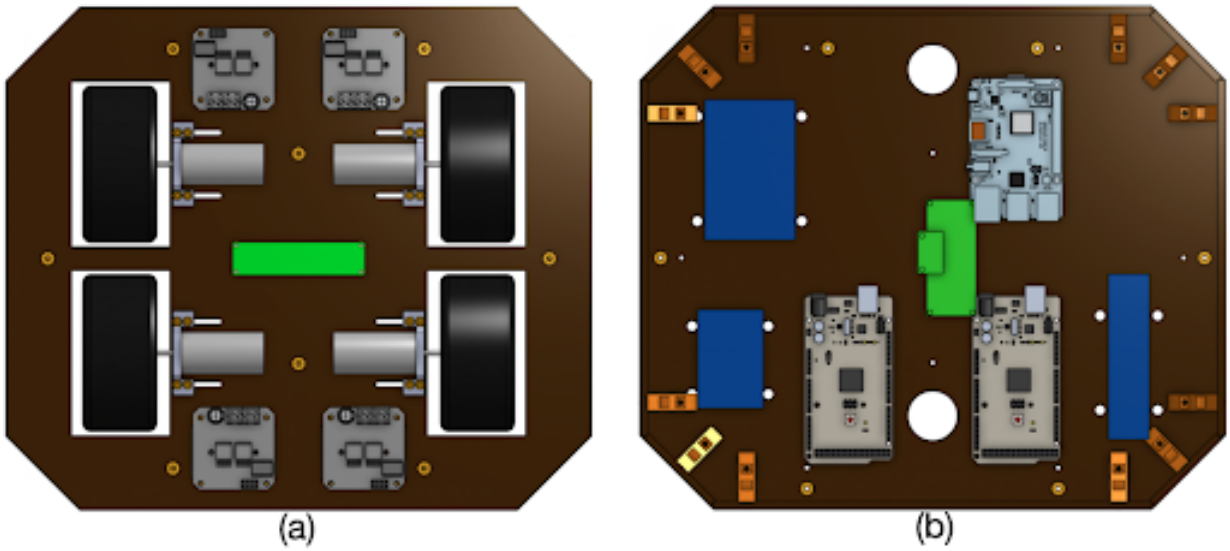

Figure 3: Top view of Level 1 (a) and Level 2 (b) of the robot CAD model

3.2 Camera Mount

As shown in Figure 2, a camera mount mechanism was mounted on the top level of the robot to support the Pixy2 camera. The camera, discussed further in Section 5.4, was intended to track the ball's position in the playing field, but was not used in the final design.

The camera mount system consisted of a rotating circular platform with side and top walls to protect the camera. The platform was mounted on a "lazy Susan" ball bearing turntable attached to the robot's top level, with a servo housed underneath the top level board to control the platform's rotation. A second servo was housed on the rotating platform and was attached to the camera to tilt it up and down. As a whole, the camera mount mechanism was designed to let the camera pan across a 270° field of vision, as well as tilt up to 20° up and down to track the ball both near and far from the robot.

The camera mount's platform, lid, and servo mounts were all designed using CAD and 3D-printed. The completed mechanism was assembled using machine screws and nuts. The top of the enclosed camera platform was 10.87" off the ground, just under the 11" height limit. Originally, the blue or green ball required by the overhead field camera during game play was going to be attached to the top of the camera enclosure, but was subsequently mounted inside the enclosure without the lid after the camera was removed.

3.3 Manufacturing

Several different manufacturing processes were used to create parts for the robot. The three 14" x 12" chassis plates as well as the bumpers were designed in CAD to contain all necessary holes and features. They were then laser-cut from 3/16" Eucaboard. Other holes in the Eucaboard chassis plates were drilled later with a hand drill, as the design and placement of components was adjusted over the course of the project.

A variety of parts were also designed in CAD and then 3D-printed. These included the motor mounts, bumper mounts, camera servo mounts, and the rotating platform and protective housing for the camera. Both 3D printing and laser cutting were low-cost and accessible forms of manufacturing that allowed the team to iterate its component designs when necessary at minimal added cost.

To secure electrical wires in place as much as possible for the competition, breadboards used for testing were eventually replaced by printed circuit boards with soldered wires. Four different PCB circuits were built for the motor wiring, accelerometer, electronic receiver, and the robot's on/off and team switching buttons.

3.4 Bill of Materials

Category	Part Name	Manufacturer	PN / Model	Quantity	Cost
Chassis	Eucaboard (4' x 8' sheeet)			1	20.67
	M3 Brass Standoff - 20 mm			32	15.22
	M3 Brass Standoff - 15 mm			30	(Total
	M3 Brass Standoff - 10 mm			18	for M3
	M3 Brass Standoff - 5 mm			6	Standoffs)
	M2 Brass Standoff - 10 mm			38	5.06
Drive System	97 mm Mecanum Wheel & Coupling	SZDoit		4	40.32
	485 RPM Econ Gear Motor	ServoCity	638356	4	75.07
	Motor Driver	Songhe	BTS7960	4	30.45
	Silicone Tubing	Quickun		4 x 0.5"	0.44
	M3 Machine Screw - 40 mm			4	3.54
	Brass Eye Hook Screw			4	1.96
	11.1 V Lithium Polymer Battery			1	Provided
Electronics	Raspberry Pi	CanaKit		1	Provided
	Arduino Mega Microcontroller	Elegoo	Mega 2560	2	Provided
	Accelerometer	ACEIRMC	MPU9250	1	14.02
	Transceiver	SparkFun	RFM69	1	Provided
	Portable USB Power Bank			1	Provided
	Jumper Wires (Various)	Elegoo		Many	Provided
	Printed Circuit Board	Elegoo		4	11.96
Camera System (Not used in final design)	Pixy2 Camera	PixyCam	Pixy2	1	Provided
	180° Servo	Deegoo-FVP	MG995	1	5.44
	270° Servo	DS SERVO	DS3218MG	1	17.00
	Camera Mount Assembly	3D Printed		1	Provided
	4" Lazy Susan Turntable			1	8.05
Total					246.20

4 Drive System

The robot's drive system, shown in Figure 4, consisted of four Mecanum wheels, each driven by a separate motor and motor driver. This configuration enabled the robot to move in any direction as well as rotate in place, allowing for a wide range of motion planning algorithms to be implemented. The motors were powered by a 11.1 V lithium polymer battery and controlled using an Arduino Mega 2560 microcontroller. Details of each component of the drive system are described in Sections 4.1 - 4.3 below.

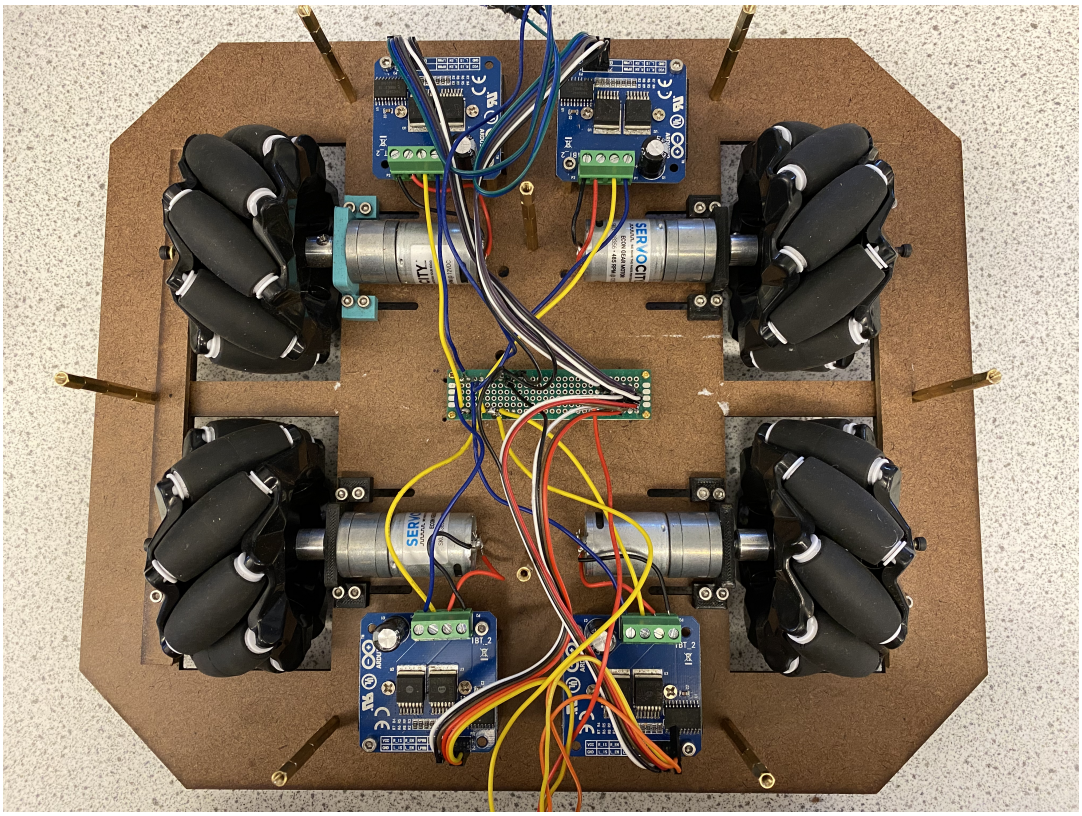

Figure 4: Bottom level of robot containing drive system

4.1 Motors and Drivers

The team chose to spend a significant portion of its budget on the drive system with the goal of creating a fast robot that could outmaneuver competing robots in the playing field. Additionally, rather than spend time and money on a mechanism specifically to kick the ball, a faster and more powerful drive system would allow the robot to hit the ball with force from any side.

With these design goals in mind, the drive motors chosen were four 12 V brushed DC Econ gear motors, each with a nominal no-load speed of 485 rpm and a stall torque of 106 oz-in. These motors provided the best combination of torque and speed given the robot's weight and design goals, within a reasonable price range for the team's budget. Each motor was mounted to the chassis via a 3D-printed motor mount, which was attached to both the chassis and the motor using machine screws.

To accommodate the current drawn by the motors, four Songhe BTS7960 H-bridge motor drivers were used—one for each motor. Although each driver's maximum rated current—43 A—was much greater than the 3.8 A maximum current of the motors, these were nonetheless the most affordable motor drivers found that met the current requirement. Each driver was wired to its motor and the Arduino Mega 2560 as shown in Figure 5; the only difference between the four drivers in terms of wiring was the digital pins used on the Arduino. The 11.1 V lithium polymer battery used to power the motors was connected to the motor drivers through the Arduino. Pulse-width-modulated signals from the Arduino's digital pins were used to control the speed and direction of each motor, as described further in Section 4.3.

4.2 Wheels

The four 97 mm Mecanum wheels used to drive the robot were mounted to the chassis as shown in Figure 4. Mecanum wheels are omnidirectional wheels with rollers mounted around the circumference at 45° from the wheel's axis of rotation. With four independently-powered Mecanum wheels arranged as shown in Figure

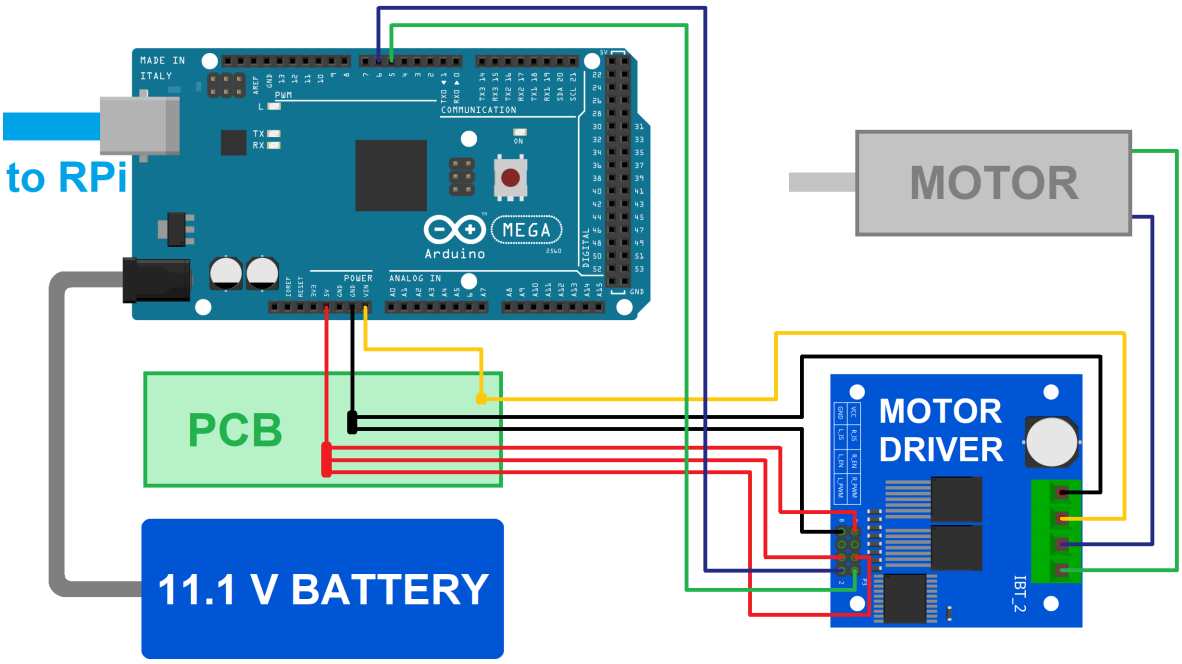

Figure 5: Wiring diagram for robot's drive system

4, the robot can move perpendicular, parallel, or diagonal to the wheels' axes of rotation as well as spin in place, depending on which wheels are powered in which direction.

Mounting the wheels onto the motor shafts required some additional design work, since the wheel couplings had an inner diameter of 6 mm while the motor shaft diameter was 4 mm. To snugly fit each wheel to its shaft, silicone tubing with a 4 mm ID and 6 mm OD was pressed onto the shaft before inserting into the wheel coupling. The set screw was then driven through the coupling and the tubing until it contacted the flat part of the D-shaped shaft.

Although this solution provided a snug fit and successfully avoided slip between the motor shafts and wheels, the flexibility of the silicone tubing allowed the wheels to bend slightly out of alignment with the motor shafts when under the weight of the robot. Additionally, the weight of the robot was carried entirely by the four 3D-printed motor mounts, causing one of the mounts to crack and need replacement. To solve these problems, the outer side of each wheel was also mounted to the chassis as shown in Figure 4. A 40 mm machine screw was screwed into each wheel opposite the motor shaft, passing through an eye hook that was screwed into the chassis. By supporting each wheel on both sides, bending of the wheels was eliminated and the robot's weight was supported at four additional points, taking stress off the motor mounts.

4.3 Arduino Motor Code

The code used on the Arduino Mega 2560 to control the motors is shown in Appendix A, and the wiring diagram for the Arduino is shown in Figure 5. The Arduino was connected via USB cable to the Raspberry Pi, described in Section 5.5, and received motion commands from the RPi through its serial port. Each command was a string containing a directional command in the form of a letter character, and a numerical time delay command indicating the time in milliseconds for which the motors should be powered in the specified direction.

When a motion command was received through the Arduino's serial port, the code would call one of eleven directional functions defined for each direction of motion. Each function powered each motor with the appropriate direction, speed, and time duration, given the requested speed and time. This was accomplished by sending pulse-width-modulated signals from the digital output pins on the Arduino to the motor drivers. Each driver was connected to two different Arduino pins to allow each motor to be commanded in the forward

or backward direction, while the motor speed was controlled by the pulse width of the signal.

Additionally, the Arduino class RobotMotors.h, shown in Appendix A, was defined to simplify the code in the main Arduino program. A RobotMotors object was defined given the eight motor control pin numbers, which it set as output pins. The single RobotMotors object could then be used as an input to the eleven motor command functions in place of the eight pin numbers individually.

5 Electronics and Communication

5.1 Receiver

For receiving the coordinate data transmitted from the SkyPixy overhead camera, the team used a RFM69 transceiver module, as shown in Figure 6. The transceiver received coordinate data for the positions of both robots as well as the ball in the field of play. An Arduino Mega 2560 was used to parse the data and then send it to the Raspberry Pi for use in motion planning.

Initially, the PCB with the transceiver was mounted on the top layer of the robot. However, its position interferred with the pan-tilt camera's angle of view, and the green color of the PCB had the potential to be confused for a green ball by the SkyPixy. The team thus decided to move the transceiver to the middle level, with only the antenna pointing out.

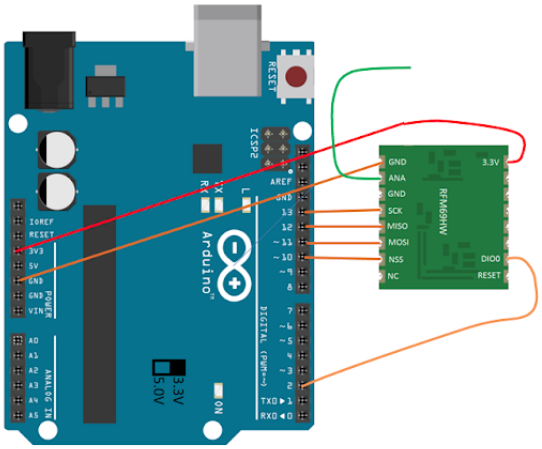

Figure 6: RFM69 to Arduino wiring diagram [1]

5.2 Accelerometer

For finding the orientation of the robot the playing field, the team used a MPU9250 9-Axis 9 degree-of-freedom (DoF) Gyroscope Acceleration Magnetic Sensor (IMU), shown in Figure 7. Initially a MPU6250 6-DoF sensor was tried, but considerable drift was found in the yaw reading over time. For accurate readings, the IMU had to be mounted in the geometrical center of the robot on the second level. The IMU data was sent to the same Arduino board as the transceiver module's data. The yaw was calculated using quaternions and a Madgewick filter and sent to the RPi for motion planning.

Due to the nature of the magnetic sensor, the magnetic field interference from the motors and drivers introduced error in the yaw. To account for this, the team determined that the yaw values could be considered reliable when the motors are halted. Thus, the robot was programmed to perodically halt, receive and process the IMU data, reorient itself, and then continue with its motion.

The Arduino code for receiving and interpreting the data from both the transceiver and the accelerometer, and passing it to the RPi, is shown in Appendix B.

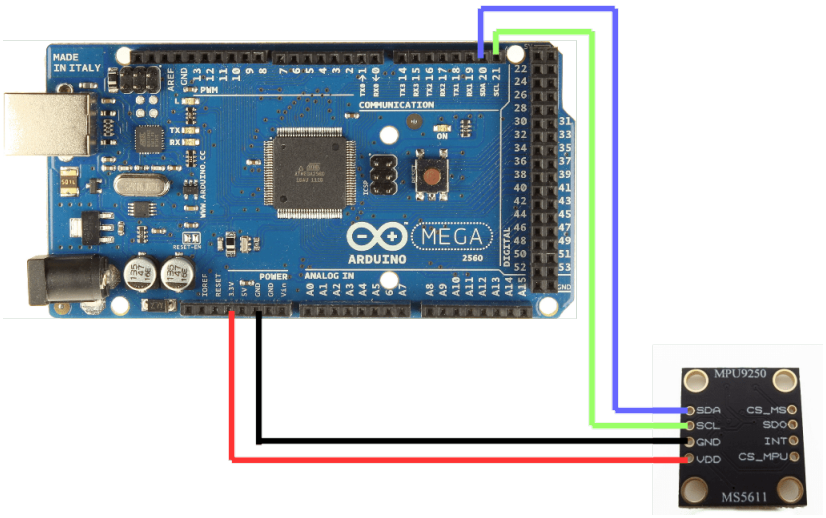

Figure 7: MPU9250 to Arduino wiring diagram [2]

5.3 Buttons

While fully autonomous during game play, the robot included two buttons to be activated by team members as necessary outside of play time. The first was an on/off button used to turn the robot on at the start of each match and after field resets, and turn it off whenever play was stopped. The second button switched the robot's assigned color in the RPi program between blue and green, so that the robot would properly identify which SkyPixy coordinates were its own position and which were for the opposing robot. The buttons were connected to the RPi as shown in Figure 8. The functions that are executed upon pressing the buttons are shown in RPi code in Appendix C. While the team planned to mount the circuit board containing the buttons on the top level of the robot, it was left on the second level due to time constraints, and had to be accessed through a large hole in the top level board.

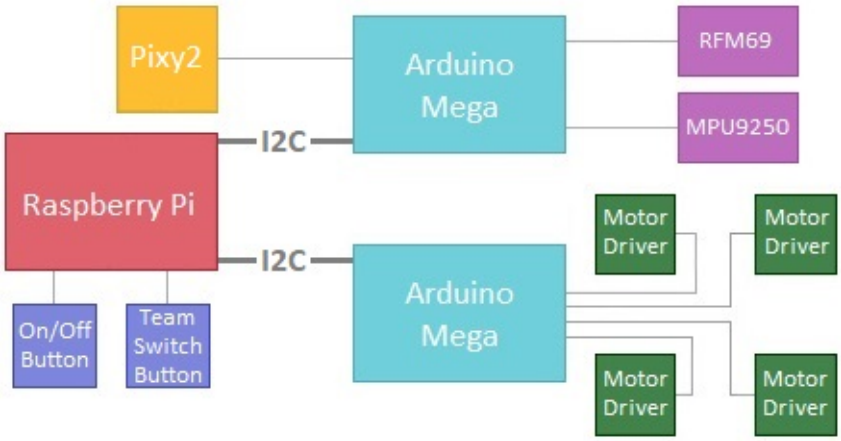

Figure 8: Electrical component connections

5.4 Camera

The original design of the robot used a Pixy2 camera to track the ball's position. The camera was mounted on the camera mount system described in Section 3.2, which used two servos to allow the camera to pan across a 270° field of vision and tilt up to 20° up and down. The camera was connected to the same Arduino

receiving data from the accelerometer and transceiver, with the intention of the camera being the primary device for tracking the ball's position. Ball position data received by the transceiver would be used if the ball went outside the camera's 270° field of vision.

Despite the effort put into developing the camera system, implementation of the camera was eventually abandoned for several reasons. While the team was able to program the camera system to accurately track the ball using PixyMon software on a computer, the tracking failed as soon as the camera was disconnected from the computer or the PixyMon application was closed. Additionally, although the pan servo was supposed to rotate up to 270°, its actual usable rotation range was only about 180°.

Moreover, the SkyPixy overhead field camera communicated the ball and both robots' positions to the robot's receiver several times per second, which was sufficiently fast for motion planning. This made the camera data unnecessary, while adding another source of positional data would only complicate the Arduino code. Thus, rather than spend additional time getting the camera to work, the team chose to remove the camera and focus on achieving functionality for the other components.

5.5 Microcontrollers and Integration

The team used three microcontroller boards for the robot—two Arduino Mega microcontrollers and one Raspberry Pi single-board computer. One Arduino was connected to the receiver and accelerometer to receive positional data, which was then sent to the RPi. The RPi acted as the brain of the robot, running a motion planning algorithm to determine how the robot should move based on the positional data from the two sensors. The RPi then sent motion commands to the second Arduino, which was connected to the motor drivers and the 11.1 V battery. The second Arduino processed the motion commands from the RPi and sent PWM signals as well as current from the battery to each of the motor drivers to power each motor appropriately.

All three boards were mounted on the second level of the robot using standoffs, as shown in Figure 9. Communication between the boards occurred using I2C connections over USB cables. As previously mentioned, the push buttons discussed in Section 5.3 were also connected to the RPi, while the later-abandoned Pixy2 camera discussed in Section 5.4 was connected to the first Arduino. A simplified schematic of all the electrical components and connections is shown in Figure 8.

6 Motion Planning

6.1 Mapping and State Space

The robot's motion planning algorithm used positional coordinates for itself, the ball, and the opposing robot to decide how to move in the field of play. The algorithm represented the robot as a point mass to simplify calculations. To account for this reduction in the robot's size, an additional 7 inches were added to the walls and the other robot.

The playing field was represented in the algorithm as a discretized grid to reduce computation costs. Every 2 inches was a square, making the entire 8' x 12' field a 48 x 72 grid. The nearest neighbor method was used to determine what square the robot was in at any time. The coordinates of the field as given by the overhead camera system were:

Top Left: (35,10), Top Right: (515, 15), Bottom Right: (515, 370), Bottom Left: (25, 350).

These can be converted to a discretized grid by creating the equations below using the boundary coordinates of the desired grid and the boundary coordinates of the field. Equations 1 and 2 are the boundaries for the y-axis and equations 3 and 4 are the boundaries for the x-axis.

$$72x + y = 515 \tag{1}$$

$$0x + y = 25 \tag{2}$$

$$48x + y = 370 \tag{3}$$

$$0x + y = 10 \tag{4}$$

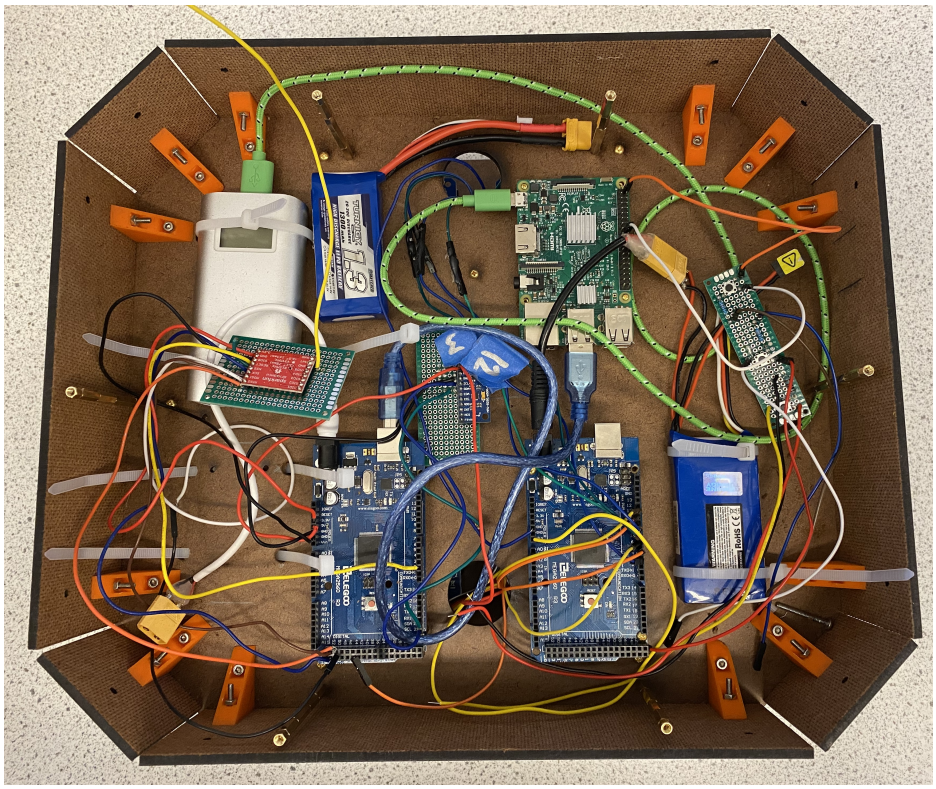

Figure 9: Second level of robot containing electrical components

Figure 10 shows how the state space looked. 1's were used to show any space that was occupied and blank spaces were used to show free space the robot could traverse. The opposing robot is pictured in the top left of Figure 10(a). It is represented as double the maximum allowable robot size to avoid any collision. The S is the starting position of the robot and the $'s are the path mapped by the algorithm to get to the ball.

In Figure 10(b) the ball is shown as a G, while the center of the goal is shown at the bottom as a Q. A circular "barrier" was created around the ball in the algorithm, with a 30° cutout opposite the direction of the goal. This would allow the algorithm's planned path to approach the ball only from an angle that would give the robot a good chance of scoring. If the team's assigned color was changed, the position of the opposing goal in the state space and the ball's barrier with respect to it were updated.

6.2 A* Algorithm

The team wanted to choose an algorithm that would be low computational cost and give a simple and concise path. After consideration of several algorithms, A* was the clear favorite to achieve the team's goals. The A* algorithm found the shortest available path from the robot's current position to the ball, within the constraints of the robot's eight programmed directions of motion. The heuristic used for the algorithm was euclidean distance. The algorithm then outputted an ordered list of coordinates the robot had to traverse to get to the ball. The Python code used to implement the A* algorithm on the Raspberry Pi is shown in Appendix C.

6.3 Motion Vectors

Making the robot follow the planned path required knowing how far it would move in a given direction when the motors were powered at a given speed for a given amount of time. Thus, it was necessary to perform tests on the robot's motion in each direction. In doing so, the team decided to limit the robot's speed in

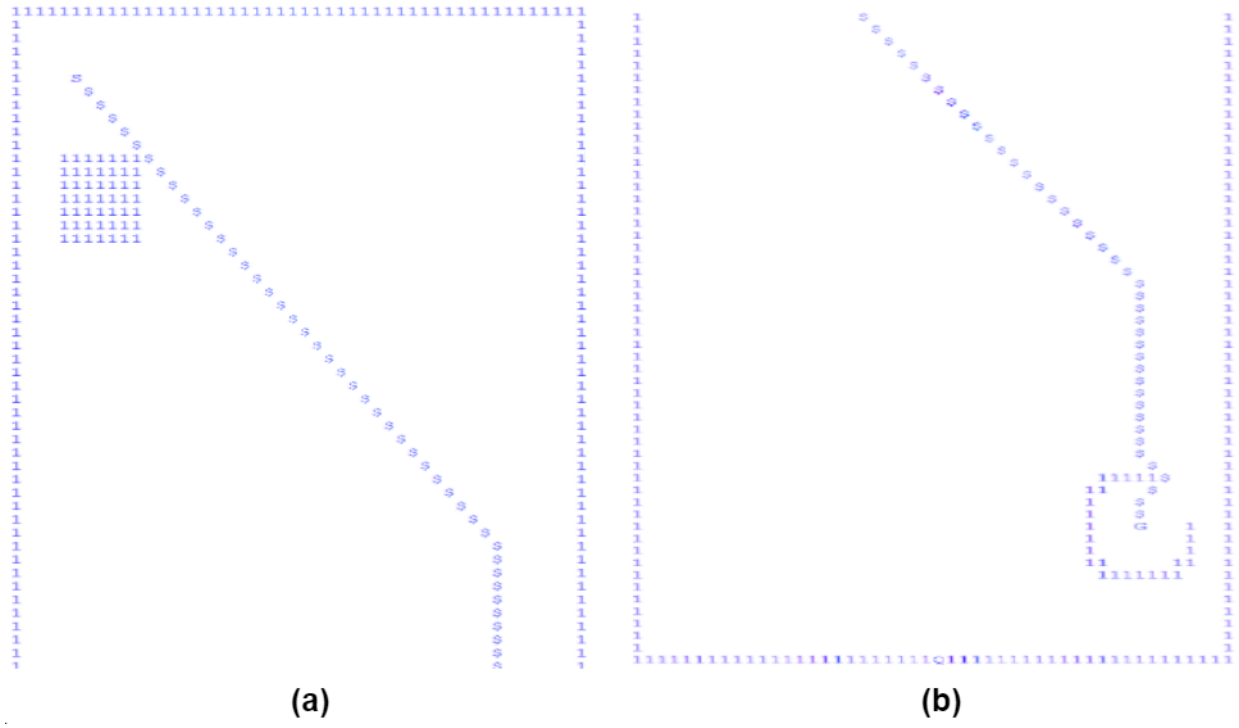

Figure 10: Robot path in field determined by motion planning algorithm

each direction to about a third of the maximum possible speed, as it was concluded that the robot may not be able to accurately plan and follow a path at higher speeds.

Speed testing revealed that for a constant PWM motor command of 85 (out of 255 maximum), the robot traveled forward and backward at 25 inches per second, left and right at 17 inches per second, diagonally at 13 inches per second, and rotated in place at 210 degrees per second. These results were consistent across multiple tests and time durations, so they were determined to be accurate enough for motion estimation. With velocity constant for each direction, a desired distance of travel could then be achieved in a given direction by powering the motors for the appropriate time duration, as found using the equation

$$Velocity = Displacement/Time \tag{5}$$

To power the motors to follow the planned path, each action from the A* algorithm was thus sent to the motor control Arduino as a command containing a direction and a time duration. The eleven directional commands were represented as letters from "a" to "k", and the time durations were given in milliseconds. In the action sequence shown in Figure 11, for example, "a" represents forward, "b" represents diagonal forward right, and "c" represents diagonal forward left, with the time duration commands following each letter. Once an action sequence reaches a total time of over 500 milliseconds, the RPi stopped sending that action sequence to the Arduino and reran the A* star algorithm with the updated coordinates to determine an updated sequence of actions.

```
['c 1980', 'a 560', 'c 113', 'b 113', 'a 80']
```

Figure 11: Example action sequence for robot motion sent from RPi to Arduino

7 Results

7.1 System Performance

Aside from the Pixy2 camera system which was removed from the final design, the team was able to get all subsystems on the robot functioning independently. However, the team was unable to integrate all systems and get the robot completely functional before the competition. In particular, both the accelerometer and the transceiver gave correct positional data when operating independently, but seemed to produce faulty data when operating as part of the completed robot, perhaps due to electromagnetic interference from the motors. Since the positional data from both sensors was required for the robot's path planning algorithm to be accurate, faulty data caused undesired paths of motion, often leading the robot into the arena walls.

The motion planning algorithm itself also had an unforeseen issue in that the algorithm would not output any path of motion if it could not find a clear path to move the ball to the goal. This meant that the robot would not start moving until the opposing robot moved out of its penalty box clear of the goal. In some cases, the opposing robot did not move at all or got stuck in front of its goal, causing this team's robot to stand still instead of moving toward the ball.

Additional issues encountered during the competition were wires coming loose from the Arduinos, and one of the wheels slipping off its motor shaft. These issues caused the robot to veer off the desired path due to a non-functioning wheel, or in one case not move at all. To make the robot more robust for competition, more secure electrical and mechanical connections would be necessary.

Overall, the team did not perform as well as hoped during the competition. The robot won its first match on a penalty kick, as it was able to find a clear path to the ball and goal without the opposing robot in the field. However, the robot malfunctioned most of the time during regular play, resulting in two subsequent losses and elimination from the tournament.

7.2 Lessons Learned

The team learned a variety of useful lessons over the course of working on this project. The first lesson is that it is worth referring back to the "lessons learned" from previous project teams, not just once at the start of the project but continually throughout. In compiling a list of lessons learned, the team found overlap with much of the advice provided from previous teams.

Beyond that, the team learned that it is never too early to start working on any stage of the project. Every day in the lab, every group member should be able to find something to work on to advance the project forward. Furthermore, it is crucial that team members effectively communicate their work, methods, and thought processes, so that other teammates can help with or take over their work if necessary. This includes commenting code.

It is also advisable to keep the end working environment of the robot in mind during design and assembly. If the robot is colliding with other objects, it needs to be robust and have secure electrical and mechanical connections. If the battery needs to be recharged between matches or a team member needs to press a button to start and stop the robot, those should be easily accessible. Although the team knew all this from the start, it was easy for some of these details of end usability to get lost in the building process.

Finally, for a project of this scale with only a couple months to complete it, it is important to keep ambitions in check and keep the design reasonably simple. It is tempting to pursue a highly complicated and sophisticated robot, but such a design is difficult to implement successfully during the time duration of this project. This is a lesson this team learned the hard way in sinking time into a camera and supporting mechanism that turned out not to be necessary for the basic design goal. The team also pursued a sophisticated motion planning algorithm which required data from several other components, and was unable to successfully integrated them by the time of the competition. The end result was a robot that was largely inept in the field of play and was defeated by robots with seemingly much simpler algorithms. While the team still believes pursuing a challenge is worthwhile, a better design strategy would be to start with basic functionally, get that working effectively, and then expand capability from there.

References

[1] Arduino Forum. *Arduino uno + RFM69HW*, January 2016.

[2] Philippe Lucidarme. *MPU-9250 and Arduino (9-Axis IMU)*, February 2021.

Appendix A: Arduino Motor Control Code

See the link below for the Arduino code used to control the motors based on motion commands received from the Raspberry Pi.

`https://drive.google.com/file/d/1CamS9kxTVr-_mdM-hC1ETM8JNqX1yzW9/view?usp=sharing`

See the link below for the RobotMotors.h library used by the Arduino motor code linked above.

`https://drive.google.com/file/d/1vabQcUx8GxFP7Ws9HB1psG0o4-XFNOMp/view?usp=sharing`

Appendix B: Arduino Sensor Code

See the link below for the Arduino code used to receive position data from the transceiver and accelerometer and send it Raspberry Pi.

`https://drive.google.com/file/d/1_WSjIiqt6-PWHDslGCOy5dD5MGEt9xth/view?usp=sharing`

Appendix C: Raspberry Pi Code

See the link below for the Raspberry Pi Python code used implement the robot's motion planning algorithm and output motion commands based on position data received.

`https://drive.google.com/file/d/1EJVrUh7USgGJfCqYZlAh_K2bVg8lEj64/view?usp=sharing`