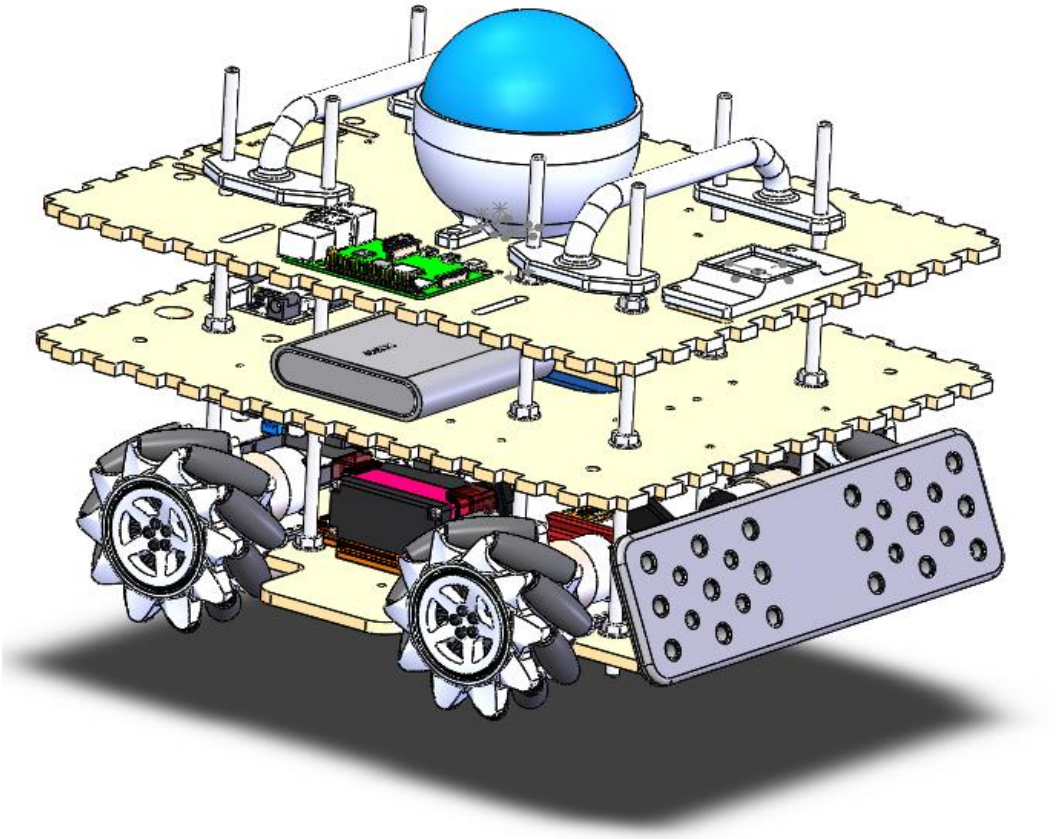


MCEN 5115: Mechatronics and Robotics Robot Project Final Report

CU Boulder, December 9, 2021

Authors:
Kirollos Gerges
Maxwell Anderson
Nick Hudson
Monique Reid
Sai Samineni

Table of Contents

ABSTRACT

The need for autonomous tracking and sensing mobile robotic systems guides the concept for the Rocket League robot. Many applications from climate monitoring to updating renewable energy hardware need autonomous systems that can function in difficult environments. The safety and risks towards manual tracking and sensing by human beings can be mitigated by developing autonomous robots that can utilize mobility, vision, and sensor subsystems to track objects of interest, and then execute decisions encoded by human authors with the information gathered.

The purpose of this project is to design and build an autonomous machine that can compete in the robot version of the popular video game *Rocket League*. The project goals were to navigate the Rocket League field and attempt to move the ball into the opposing goal. The major requirements for the design and build were a $200 budget, a maximum size of 12’ by 14’ for the robot, and a designation of “Blue” or “Green” robot while the opponent represents the opposite color. Additionally, the autonomous actions have to be safe for all robots on the playing field and not cause any damage to the robots.

The autonomous robot integrates movement, sensing, vision, localization, and tracking to find the ball and score a goal into the correct goalpost. The omnidirectional wheel design of the Mecanum wheels allows the robot to move in any direction with the necessary friction and stability to generate a reliable speed. The ball is identified through the most recent location provided by the field’s radio GPS coordination as well as the vision tracking to confirm the ball is available to attempt a goal. Once the ball’s location is confirmed, the robot calculates the necessary rotation and direction to go to the ball. Using this information, the robot travels to the ball and attempts to use the linear arm actuator to shoot it into the goal. Future work could include PID control, coupling a servo to the camera for full swivel computer vision, and improving the speed and power of the actuator.

SYSTEM ARCHITECTURE

According to Kortenkamp (2016), architecture refers to how a system is divided into subsystems and how those subsystems interact. The key parts of the robot system architecture are the body, control system, manipulators, and drive train. Some of the goals when designing effective architecture include:

- Capacity to control sensors and actuators in real-time
- Account for uncertainty or noise when encoding diverse decisions and outcomes
- Ensure all functions are able to work simultaneously and asynchronously

Further, there is no single architecture that is optimal for all outcomes. Therefore, we must use different subsystems and application integration for the best possible outcome. The following provides the architecture system breakdown to achieve the goals of the robot design:

Table 1: System Architecture Decision

Architectural decision	Rationale	Option(s)
SBC (Single Board Computers)	Simplify adding PC based technology and provide pre-built functions	Raspberry Pi 4 Arduino Mega
Actuator	Hitting system with necessary voltage to punch the ball	3D designed actuator (refer to Appendix link)
Movement	Driving system with necessary stability for movement on the field	Mecanum wheels (4)
Perception Sensor(s)	Inform localization and supported tracking for the robot ; Sense objects and facilitates tracking Provide coordinates on the field and object location	Inertial Measurement Unit (IMU) USB Camera Radio / Antenna
Display and Remote View	Ability to improve object tracking and update camera HSV inputs based on lighting changes	Trackbar (hue, value, sat) VNC
Power Source	A power source to drive the robot autonomously and to turn on the SBCs	Lithium Polymer Battery Portable Charger
Controller	A mechanism to input external communication and reset the robot	Buttons (Reset, Team Color, etc)
Firmware and Hardware	Signal processing; Control the voltage to the actuator; Between Radio and RFM69	Motor drivers (4) Voltage regulator Level shifter
Chassis	3D designed layers to hold all components and mount the parts	3D Printed and Laser Cut Frame; Brackets and Molds; Mounts; MDF
Peripheral Protocol	Wires and master/slave communication protocol to inform modules	I2C x 12 SPI

Form and Functional Breakdown

The functional breakdown is based on the frequency of information processing and the flow of information from upstream to downstream of the initial inputs.

The frequency of information processing ranges from lower frequencies such as the Radio providing the localization coordinates at ~two frames / second and compared to the Radio, the information processed at higher frequencies such as the actuator and IMU. Further, the order of information processing also impacts the decision-making process with the SBC and Radio functioning upstream, while sensing and vision are more downstream in their function from the initial inputs. The table describes this breakdown and the impact of the process components of the Rocket League robot.

Table 2: Impact of Process Components

	Upstream	Downstream
Lower Frequency	Radio Remote Control and I2C	Firmware
Higher Frequency	SBC (RPi, Arduino) Camera Inertial Measurement Unit (IMU)	Actuator Protocol

The figure below illustrates a high-level schematic of our system. The red arrows indicate power and the black arrows represent an exchange of data or information.

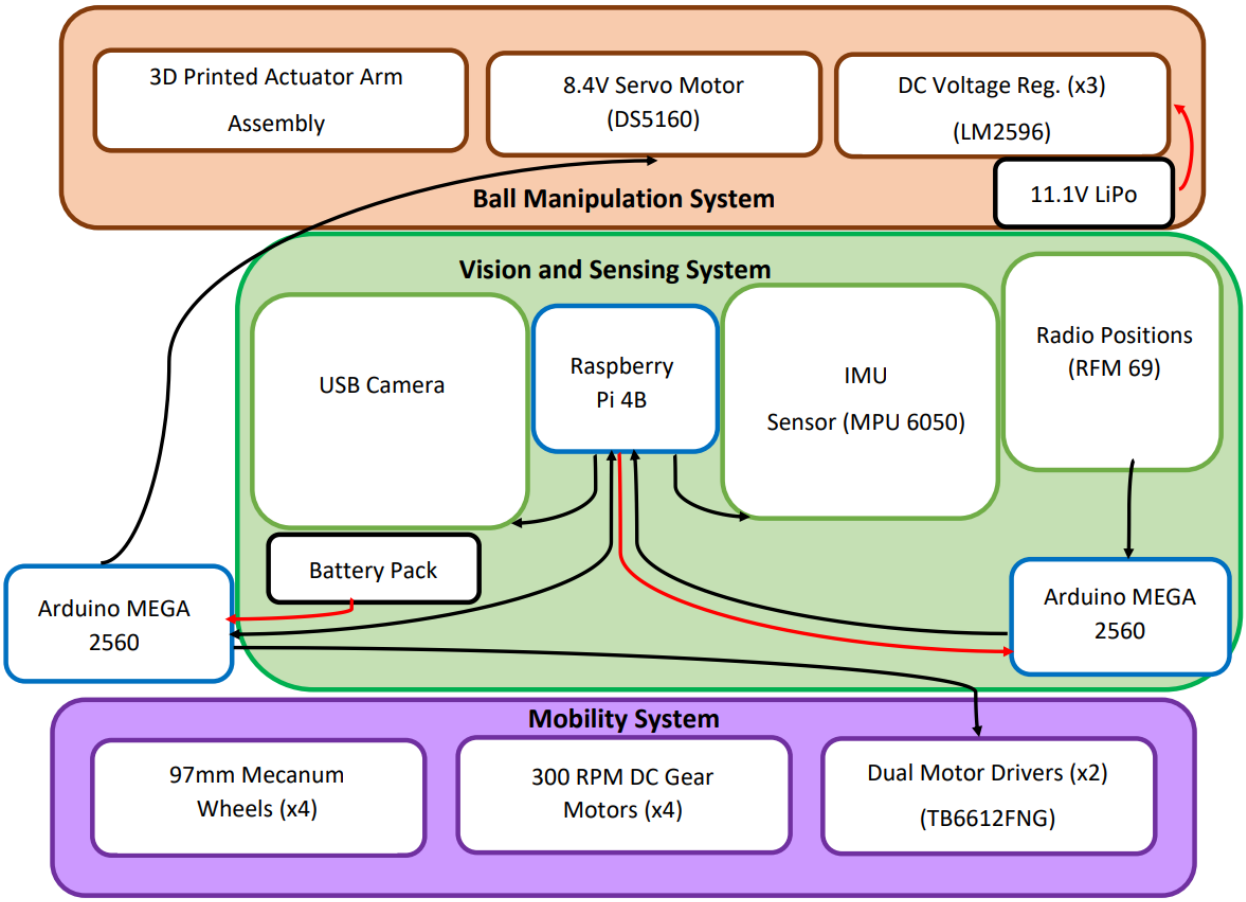


Figure 1: High-Level Systems Design

CHASSIS

Frame Design

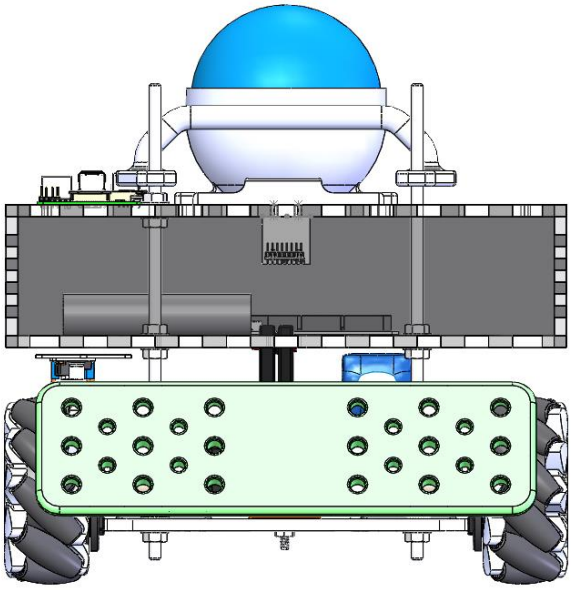


Figure 2: Front View of Robot, Frame Design

The chassis encapsulates all major sub-components of the robot and mounts for all subsystem elements. Two motor drivers are coupled to four mecanum wheels for mobility and stability. Other major components include two Arduino Mega, one Raspberry Pi, four DC motors with motor caps, one inertial measurement unit, and one USB camera. Initial plans included using a Pi camera for ball detection and tracking. However, challenges with the RPi's cv2 library with the Pi camera led to us utilizing a USB camera with a better frame rate. Overall, the motor drivers in the chassis inform the direction and acceleration commands from the Arduino Mega based on measurements from the IMU and the USB camera detection.

The chassis frame was designed with a number of goals: sturdiness, low cost, ease of laser cutting, and ease of additional modifications or repair. Ultimately, the team decided to laser-cut the frame out of 0.25" thick MDF and used long, threaded rods with locking brackets to assemble the layers.

Bumper Design

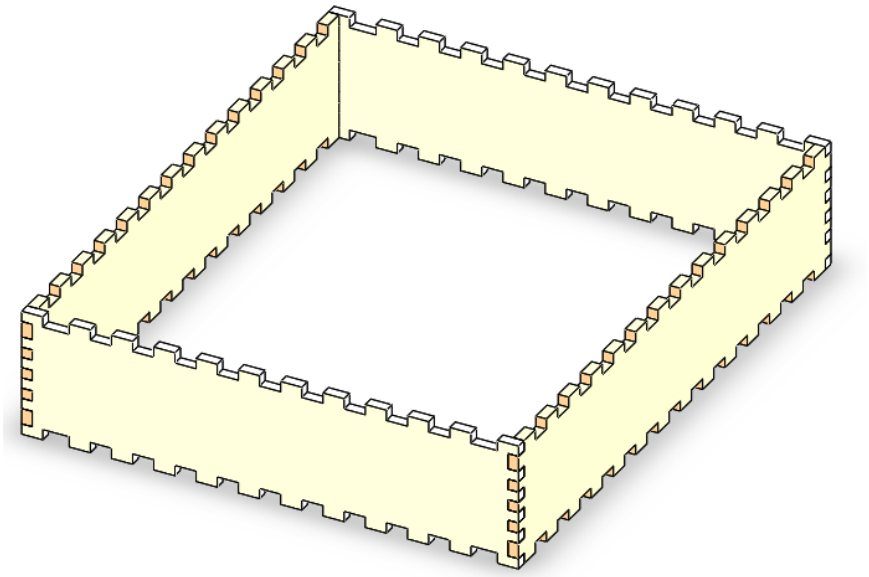


Figure 3: Bumper using Finger Joints Design

The bumper design features two identical pieces, front/rear and left/right. This design was made using 0.25” thick MDF wood on a laser cutter. The finger joint mechanism is commonly used because of its strength, stability, straightness, and consistency. This design ensured that our robot would be protected during the game from both the ball and other robots. Finger Joints are also cost-effective because less wood is wasted during manufacturing. An adhesive glue was applied to create a stronger, more durable connection.

MOBILITY SUBSYSTEM

The mobility subsystem includes four DC motors, two motor drivers, an Arduino Mega, a set of four mecanum wheels, and draws power from the 11.1V lithium polymer battery. Our team knew very early on that we wanted to invest in a robust mobility system to attempt to be faster and more agile than our opponents. This was the main deciding factor for using mecanum wheels. These wheels allow for omni-directional movement and can provide both rotational and translational motion at the same time when driven correctly, this will be discussed in more detail later. Along this vein, we invested in powerful motors so that we could move more quickly than our competition. At the beginning of the project, we made some assumptions about our final system's weight and used that, along with the diameter of our wheels, to find a set of motors with desirable specifications. We finally settled on four, 12 volt, 300 RPM motors made by Geartisan.

Rotational and Translational Direction

A Mecanum wheel drive for a robot allows for rotational and translational movement at the same time. A mecanum wheel is a special type of wheel that has rollers faced at a diagonal on the exterior wheel surface. These rollers mean that each individual wheel outputs a diagonal force based on it's handedness and the direction in which it is being driven. When used in a four wheel pattern and driven simultaneously, these diagonal force vectors cancel each other out and output a net force in one direction. This means that depending on how one drives each wheel, one can output a force vector in a wide range of magnitudes and directions. The following diagram shows a couple of examples of how one can drive a robot with a mecanum wheel drive. This diagram, and several other ones like it, informed our team's approach on how to set up the wheels on our robot.

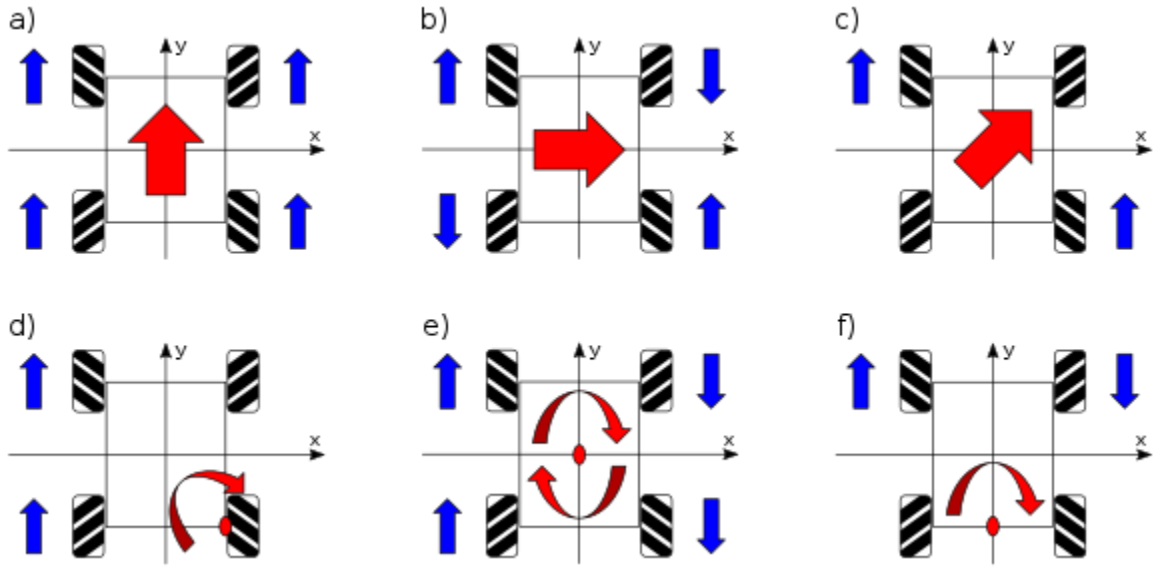


Figure 4: Mecanum Wheel Motion Patterns

Source: https://en.wikipedia.org/wiki/Mecanum_wheel

Being able to get our robot to drive in "cardinal directions" and diagonals was a useful first step, but we wished to be able to give our robot any vector derived from coordinates sourced from the pseudo GPS system and have it drive along that vector. In order to do this, we needed to implement a driving method that goes by a couple of different names, but we called it a "variable speed drive". First, one must calculate the angle of the desired vector of movement and a reference unit vector that indicates the orientation of the robot and its motors. Next, we calculate the magnitude of our desired vector and the angle that we wished to rotate our robot.

Equation 1: Movement Vector Angle and Magnitude

$$S = sin(\omega_V \pm \frac{\pi}{4}) \cdot \left|\left|V\right|\right| + \omega_R$$

Where S is the motor speed, ω_V is the angle between the reference vector and our desired vector, ||V|| is the magnitude of our desired vector, and ω_R is our desired rotation angle.

The reason there is a plus-minus symbol is that the orientation of each individual wheel determines the phase of the sine function that is needed to output the desired force. One thing that is tricky about using this drive is that when you add ω_R, the S for each motor will be different, and could be larger than the largest possible value that our Arduino could output with a PWM signal. To fix this, after we computed each motor speed, we found the largest motor speed to set the maximum value, and divided each of the motor speeds by that maximum value. This gave us S values: $-1 \leq S \leq 1$, which we could then scale to be within the range of -255 to 255, the maximum range that our Arduino Mega could provide. These calculations were found online and are source number 3, in the “Links” section of the appendix.

Motor Drivers

Our team needed to be able to drive each of our wheels at different speeds at the same time. To achieve this, we used two motor drivers from Sparkfun. We used two TB6612FNG motor drivers to control all four of our motors. These chips take in two PWM signals, four digital inputs, logic voltage, and motor voltage. Our motor voltage was sourced from the Lipo battery and the logic voltage was sourced from the Arduino. These chips can take up a total of 1.2 amps of current and can take a range of logic voltages from 3.3 V to 5 V. We used a benchtop power supply to test the amount of current that our motors would draw, and found that in total the four motors would draw 560 mA. After this test, we concluded that the TB6612FNG motor drivers would be more than adequate for controlling our four motors.

Thinking ahead and planning for our electronics dying during testing, we made sure to make the board that housed our motor drivers to be as modular as possible. Every component is connected into header pins that are soldered into a solderable breadboard The rest of the circuit was built up off of the same breadboard.

We, unfortunately, made a few mistakes and had to recreate our circuit a couple of times, however, in the end, we were grateful for that as we learned something new each time we remade the board. In the end, we all had better intuitions for organizing circuit boards, properly spacing electrical components and all of us got much better at soldering.

Angle and Speed of Movement

Finding the appropriate motors for our robot was another important part of our design process. We had the thought to invest in our mobility so that we could outmaneuver our competition, but we knew we needed to balance speed and power. Once we had settled on the wheels we wanted we could begin calculations for the speed. Unfortunately at the beginning of the process when we first ordered our parts we had no idea what the final weight of our robot would be. So based on the components we did have we estimated our final weight would be around two (2kg) kilograms. This was, in our mind, the bare minimum that our motors should be able to move easily. With the diameter of our wheels decided, 97 mm, we used this dimension to figure out the maximum speed we would be able to drive.

Equation 2: Maximum Drive Speed

$$V = \left(\frac{x\ rot}{1\ min}\right) \cdot \left(\frac{1\ min}{60\ s}\right) \cdot \left(\frac{2\ \pi\ Rad}{1\ rot}\right) \cdot \left(\frac{48.5\ mm}{1\ Rad}\right) \cdot \left(\frac{1\ m}{1000\ mm}\right)$$
$$= 0.0051\ x$$

In order to have a leg up on our competition, we knew we wanted our robot to have the capability of moving faster than one (1 m/sec) meter per second, but also have enough torque to easily move the 2 kg we estimated our robot would weigh. These constraints brought us to buying motors rated from 250 rpm to 500 rpm. After some deliberation, we decided that having a little more torque to move our robot was more important to us, and so we chose motors on the lower end of that range at 300 RPM. This gives us a theoretical maximum speed of 1.53 m/s. Unfortunately, we were unable to verify that our calculation was correct but in the future, we would like to test our robot's maximum speed and develop a velocity-current curve to analyze how fast our robot is compared to the current draw of the motors. This curve would be really helpful in the future for developing more refined controls and analyzing the robot's performance.

Mobility Subsystem Pinout

Part / Connection	Input	Output - Arduino
Rear Left Wheel	PWMA	7
Rear Left Wheel	AIN2	8
Rear Left Wheel	AIN1	9
Rear Motor Driver	Standby	10

Rear Right Wheel	BN1	11
Rear Right Wheel	BN2	12
Rear Right Wheel	PWMB	13
Front Left Wheel	PWMA	6
Front Left Wheel	AIN2	37
Front Left Wheel	AIN1	33
Front Motor Driver	Standby	5
Front Right Wheel	BN1	29
Front Right Wheel	BN2	25
Front Right Wheel	PWMB	4

ACTUATORS

3D-Printed Arm Assembly

The actuator that we decided to move forward with was a simple linkage-based linear actuator. We wanted to use this actuator to manipulate the ball to move faster than simply just ramming the ball as most of our competition was planning on doing.

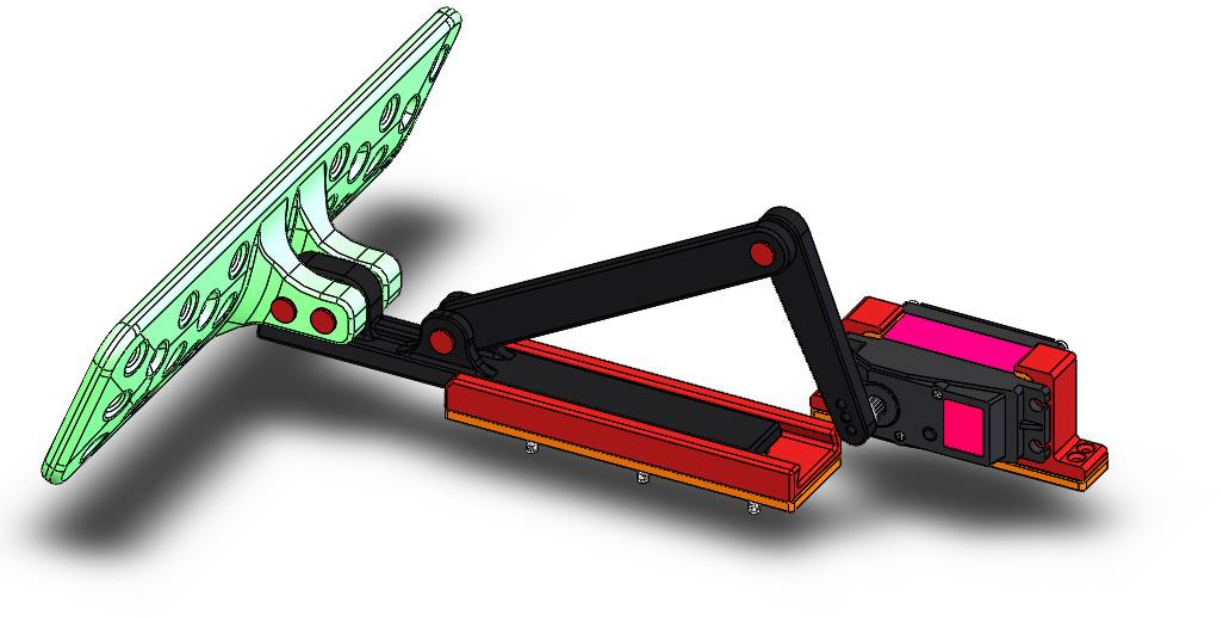


Figure 5: Linear Actuator

As one can see in the above CAD diagram, we used a strong servo motor, attached to a linkage that turns the rotational motion of the motor into linear motion. The linkage also makes use of some mechanical advantage to increase the force output. Overall, we were aiming to be able to accelerate the ball at twice the acceleration due to gravity. We believed this number would be simultaneously a reasonable challenge to attain and move the ball fast enough that it would be difficult for our competition's robot to react to. Every piece in the assembly was printed on a LULzbot workhorse except for the servo motor mount. The motor mount was printed in a compliant material that allowed the motor to be held with some force without the use of fasteners. Everything else was then bolted to the chassis.

Servo Motors

We used an 8.4V High Voltage Digital Servo to move our linkage. We used a servo motor because we wanted to be able to specify the angle to which our motor actuates. This allows a higher fidelity of control than a regular DC motor would give us

Voltage Regulators

The servo motor that we used to actuate the pusher, ran on 8.4 Volts, This was much lower than what the lipo battery provides but higher than what our Arduino mega could provide. To get around this we used three voltage regulators in series to drop our battery voltage from 11.1 V to the required voltage to run the servo at an acceptable speed.

Actuator Subsystem Pinout

Part / Connection	Input	Output - Arduino
Actuator Angle	PMW	2
Actuator Battery	PWR	11.1V => 8V
GND	GND	GND

SENSORS & GUIDANCE

Radio Positions

The purpose of establishing this wireless communication is to set up a pathway to allow the robot to receive positional data for the ball and the two robots during the match, every half a second. On our robot, we used the RFM69 radio module to receive the data. The RFM69 chip is wired to a level-shifter because the board will not tolerate 5V logic. From there, we were able to transmit the data from the module to an Arduino Mega 2560 in order to inform our vector logic by giving us updated coordinates to approach the ball, to ensure our robot does not go out of bounds, and only stay in the goal box for up to 20 seconds.

Radio Subsystem Pinout

RFM 69 Module Label (Left Side)	RPi Pin Label and Pin #
O/ MISO	MISO (21)
I/ MOSI	MOSI (19)
C/ SCK	SCLK (23)
S/ NSS	CE0 (24)

R (RESET)	GPIO 5 (29)
0 (DIO0)	GPIO 6 (31)
G (GND)	GND
3.3V (3.3V)	3.3V

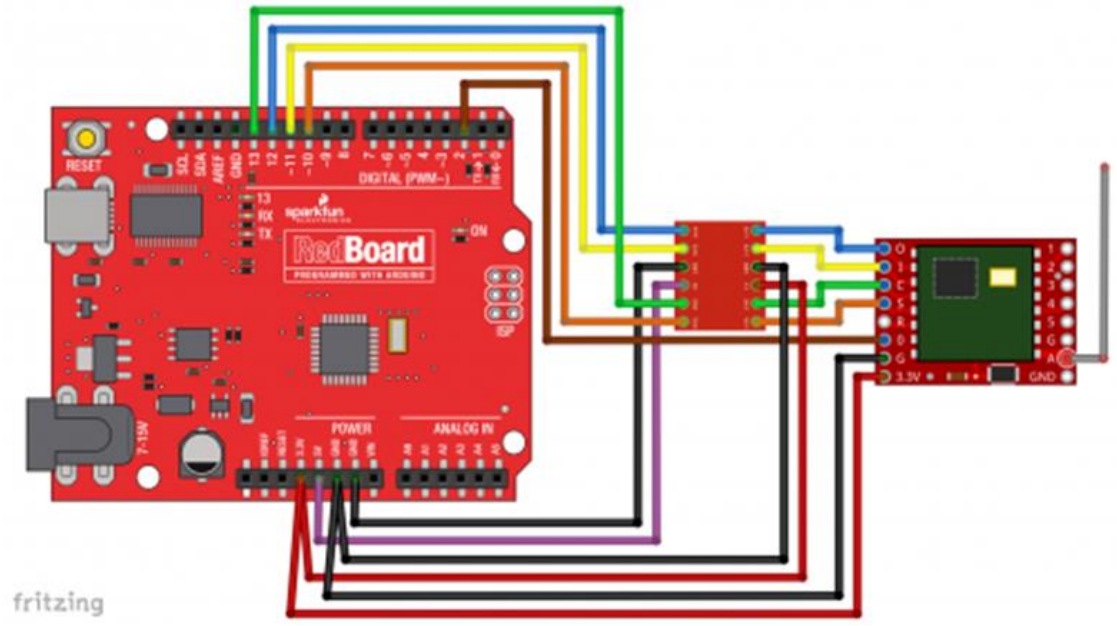


Figure 6: RFM 69 Module with Level Shifter Wiring

Source: [RFM69HCW Hookup Guide - learn.sparkfun.com]

Inertial Measurement Unit (IMU)

Initially, our team purchased an ADXL 345 3-axis accelerometer. We thought this would give us enough information to inform our position on the field and be able to correct our orientation. After setting up the ADXL with the Raspberry Pi, it was discovered there was a better way to inform that decision with more data using a 6-axis IMU. In the final prototype, we pursued an MPU 6050 6-axis inertial measurement unit (IMU) with the Raspberry Pi 4B including an accelerometer and gyroscope to give changes in XYZ. The MPU 6050 uses i2c bus, SCL, and SDA pins, on the Raspberry Pi.

IMU Subsystem Pinout

Part / Connection	Input	Output - RPi
Serial Data (I2C Connection)	SDA	SDA
Serial Clock (I2C Connection)	SCL	SCL
GND	GND	GND
VCC	VCC	5V

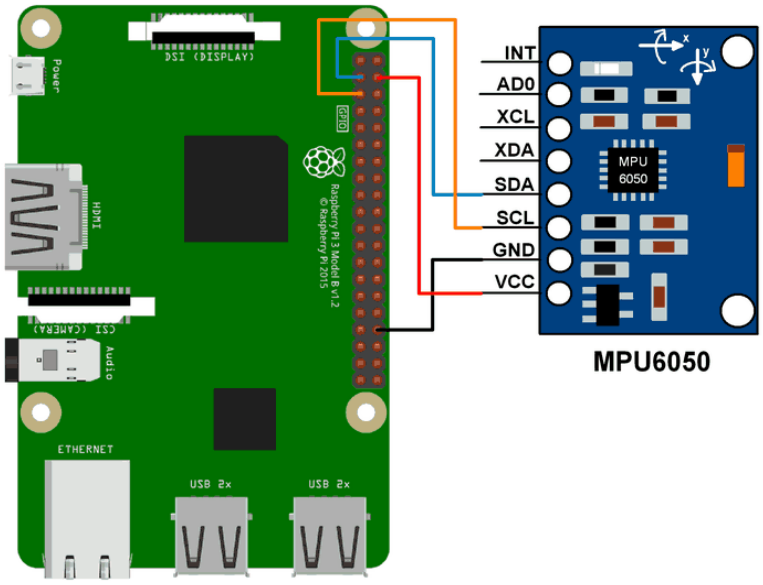


Figure 7: MPU 6050 Wiring to the Raspberry Pi

Source: [Raspberry Pi Mpu6050 Accelerometergyroscope Interfacing With Rasp... (electronicwings.com)]

VISION

Pi Camera

The Pi Camera was initially selected to develop the vision subsystem. However, after spending six weeks developing object tracking and detection with the Pi Camera, the difficulty of integrating the Pi Camera with the OpenCV library, delayed frame rate and poor color sensing ability of the Pi camera led us to select an alternative USB Web Camera for the vision subsystem.

USB Camera

Once we transitioned to the higher resolution USB web camera, we were able to improve the color detection and object tracking built on OpenCV to identify the ball and use object tracking to retrieve coordinates from the center of the ball. The program flow with the OpenCV library is as follows: 1) Capture frame and pass into variable to store for processing ; 2) Use trackbars to set the HSV values for low and high range to select the desired color for the ball ; 3) Perform color detection ; 4) To track the object matched by color detection, return coordinates for the center of the ball in the frame.

Color Detection

The color detection was performed using built in functions from the OpenCV such as masking, cv.imshow, and filtering based on inputted values for hue, saturation and value with low and high values. By creating track bars that allowed us to set the low and high limits for the hue, saturation and value, we were able to only sense the color of the ball.

Object Tracking

Once the color detection successfully sensed only the ball color and omitted other objects in the frame, we implemented object tracking using contour and tracking coordinates centered on the ball. These functions created a trace on the ball and provided coordinates from the frame to continuously tracking the ball as long as it was visible to the USB camera lens.

Unfortunately, after we had tested and implemented the vision subsystem to track the ball, the SD card with the OpenCV packages and ball tracking vision code was corrupted due to a conflict between different Python versions during transfer from the development RPi to the RPi onboard the robot. We were unable to reimplement the corrupted file and use the vision subsystem with the robot for the Rocket League runoff. The code developed for the vision subsystem and the integration tests are included in our project documentation links and repository.

DECISION MAKING

The robot's decision making was made primarily by a raspberry pi 4. We made our microcontroller set up so that it acted like a pipeline. The raspberry pi receives information from the arduino mega (that parses information from the radio), the camera, and the IMU and then sends commands to the arduino mega that controls the DC and servo motors. Essentially communication never travels “upstream”. We implemented this “pipeline” so that we never had any commands from the motor arduino that interrupted the decision making that the rpi4 was performing. This will be discussed further later in the document.

Radio Communication

The radio as discussed above, receives coordinates from the pseudo GPS for both robots on the fields and the ball. These coordinates are what we used to mostly inform our robot it's position relative to the outer bounds of the playing field and its position between itself and the ball. We set up a set of boundaries inside of the robot's decision making process to make sure that it would never travel outside of the playing field bounds, and be able to figure out how to enter back into the playing field if it accidentally drifts outside.

CONTROL

Control Algorithm

Our robot had a pretty simple control algorithm. First, we needed to overcome the issue that our robot would start the game with a random orientation. If our robot did not know its own orientation relative to the playing field then it would be almost impossible for the robot's decisions, and execution of them to be successful. To get around this, once the robot turned on and the game was started, the robot would drive forward. We would then take the resulting change in position vector and calculate the angle between the position vector and a reference unit vector that is defined by the team our robot is playing on. For the Green team this was a vector that pointed from west to east, for the blue team it was the opposite. Once this angle was calculated the robot then will rotate so that it can be oriented along the reference vector. We taught the robot to perform this maneuver by timing how much time it takes the robot to turn 360°. Then by taking the angle we wish to rotate and finding the fraction of that time that it would take the robot to rotate the desired amount.

Now that our robot is properly oriented, the IMU can be used to help determine orientation from now on. Now using the coordinates received from the radio, we can start to play the game. First, our pi calculates a vector from its current position to a position just in front of the ball. This is calculated with respect to the boundaries of the playing field and may be checked first. Once we have found a valid location around the ball to drive to, we calculate the corresponding vector. This vector is then fed into the code described in the mobility system's section and the four motor speeds needed to drive the robot in that direction are calculated. These speeds are then fed to the Arduino Mega which handles driving the motors. The rpi4 then loops through this decision making process again until the robot is close enough to the ball to hit it with the pusher, about 4 inches away. Once the robot has determined it is close enough to the ball, it then rotates so that it is pointing to the goal and activates the pusher. The process of this is very similar to the initial orientation calibration. A vector is calculated from the robot's current position and the enemy goal. The angle between this vector and its current orientation vector (calculated from the IMU) is calculated and then the robot rotates until this angle is found to be zero. Once it is, the pusher is activated and the robot is set back to its original state.

If a goal is not scored and the match stopped the robot loops back through the decision making process described above until it is.

INTEGRATION

The approach to integration started with constructing the chassis design and testing the mobility subsystem. Once the chassis was robust and could remain stable under the applied torques from the motors and the movement of the wheels, we worked on including the actuators and sensors. The vision subsystem was left for final integration, however this led to challenges with coordinating the positions for inputs and outputs between the Arduino and RPi. We had to review the pinout diagram and refine the wiring in order to cohesively integrate all of the subsystems. Once the team came together and understood the needs of each subsystem more completely, integrating the electronics of each subsystem was much more simple.

COMMUNICATION

As discussed in the “Decision Making” section of our report, the on-board microcontrollers communicated in a sort of data “pipeline”. Each microcontroller was only ever receiving data from input sources and outputting data to output sources. This effectively made a one way chain of communication where each microcontroller did not know what was happening further down the chain. We did this to not run the risk of garbaling the signal between two boards especially since we used two separate methods of communication. The Arduino Mega that receives and parses data from the radio sends those coordinates down the chain to the Raspberry Pi via a serial USB connection. The Pi then calculates four motor speeds from the vectors that it calculates based on those coordinates. Then, via an I^2C communication protocol, the Pi sends these motor speeds to the second Arduino Mega which executes the command. Our communication protocol used I^2C with the Pi as the master and the Arudino as the slave. This I^2C connection allows for serial data exchange between the two devices. We have established this connection using the SDA and SCL pins on both devices. Seeing that the motors must all drive at different speeds at the same time, the second arduino must make use of pulse width modulation to appropriately drive the motors. This capability is built into the digital pins of the arduino mega, but nevertheless, was invaluable to our design. Without it our “variable speed drive” described above would not function. The Arduino does this outside of the control of the “Master Pi”, which as the Arduino is executing the drive command is sourcing data from the radio, IMU and the camera to make it’s next decision.

USER INTERFACE

Robot Push Buttons

In order to best prepare for game day, our team wired several push buttons on the top of the robot as a user input.

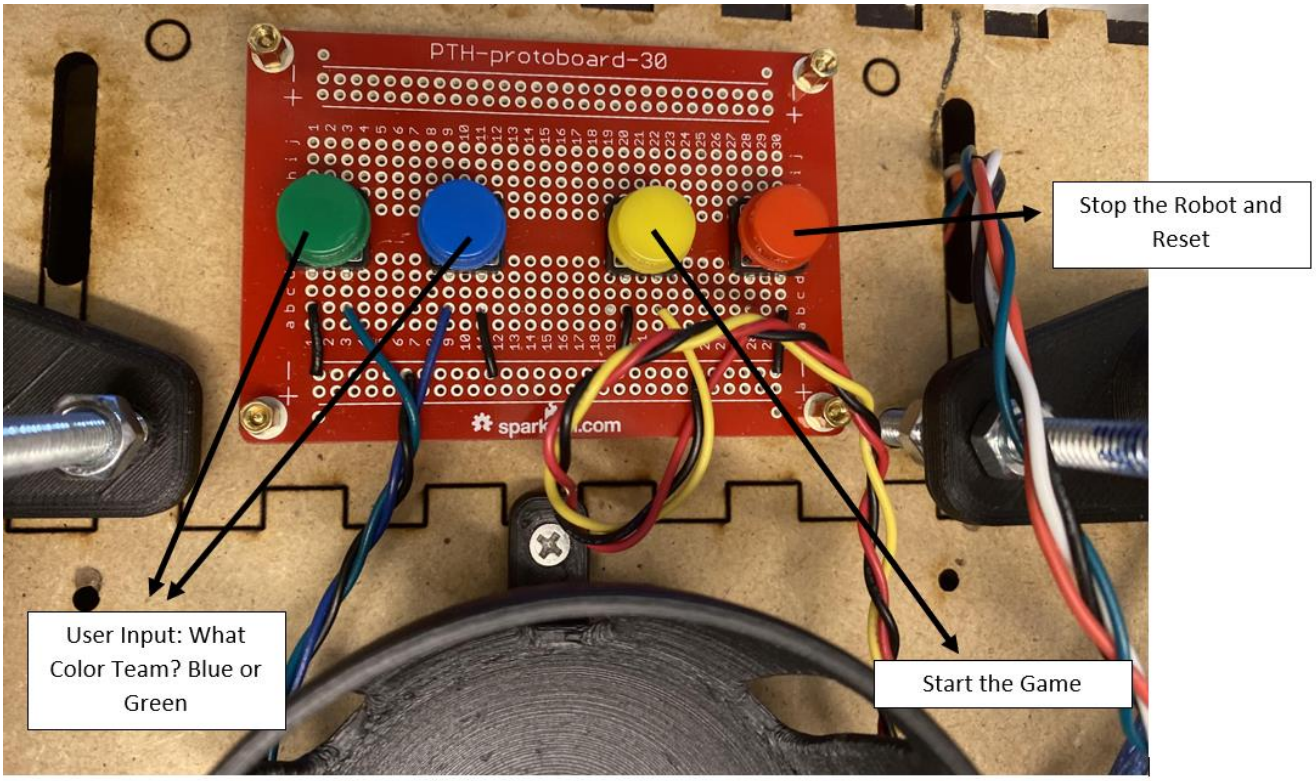


Figure 8: Push Buttons on the Robot

VNC Connection from RPi

In order for our robot to be autonomous, our team prepared for wireless communication via RealVNC. The purpose of this application is to allow remote access from a laptop to the robot. Several members of the team installed this application on their computer to prepare for wireless communication to the Raspberry Pi 4B device.

Environmental Light Dynamic Update on Vision System

The final user interface component of the design is in the vision subsystem. Our robot uses ball tracking with OpenCV to detect the presence of the colored ball using computer vision techniques and track the ball as it moves by drawing its position. Objects can look different in varying light conditions which would impact the exact color boundaries of the ball. In order to account for this, our team developed an adjustable dial for hue, saturation, and value, to change the settings on the camera. The user is able to adjust this as conditions change prior to running the code on the Raspberry Pi serial window as shown in Figure 9.

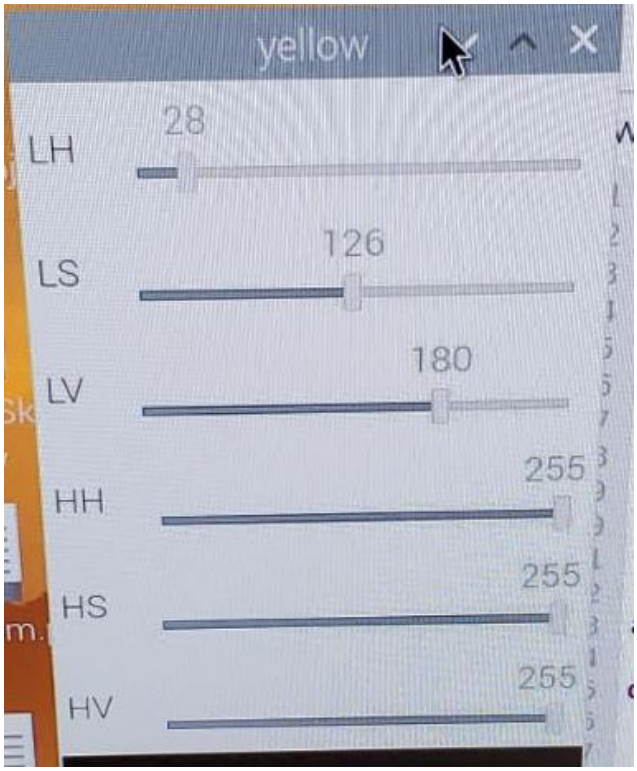


Figure 9: Adjustable Environment Dials for Vision System

ROBOT OPERATING SYSTEM

The Raspberry Pi 4B and Arduino MEGA 2560 are set up as serial communication via USB. This is the simplest hardware connection between the boards, as shown in Figure 10.

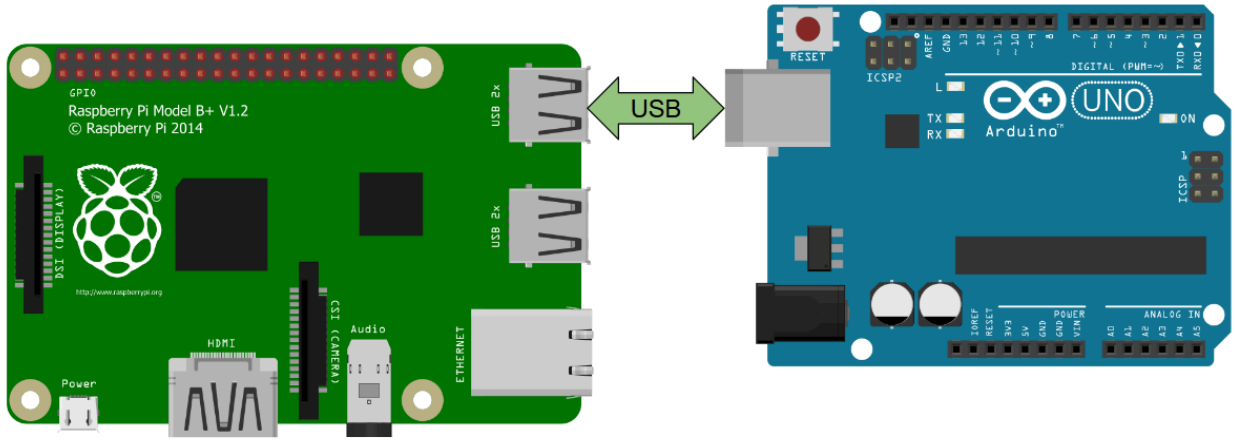


Figure 10: Arduino and Raspberry Pi Communication

Source: [Raspberry Pi Arduino Serial Communication - Everything You Need To Know - The Robotics Back-End (roboticsbackend.com)]

The software setup was fairly simple and straightforward. Our team used a basic Arduino I2C slave program script to establish the address. The Wire.h package was downloaded in order to use commands such as Wire.begin, Wire.onReceive, and Wire.onRequest to prepare the Arduino to receive and send data. In a loop, the Arduino had commands to receive bytes and read it and send data back to the Raspberry Pi.

The Raspberry Pi detects the Arduino board on the I2C bus. In Python, several packages were installed including iostream and wiringPiI2C.h. The master code first sets up the I2C communication with the Arduino and returns a message to the serial monitor when successful. Then, the Raspberry Pi has a command to either send data to the Arduino or receive.

FINAL CAD DESIGN & MATERIALS

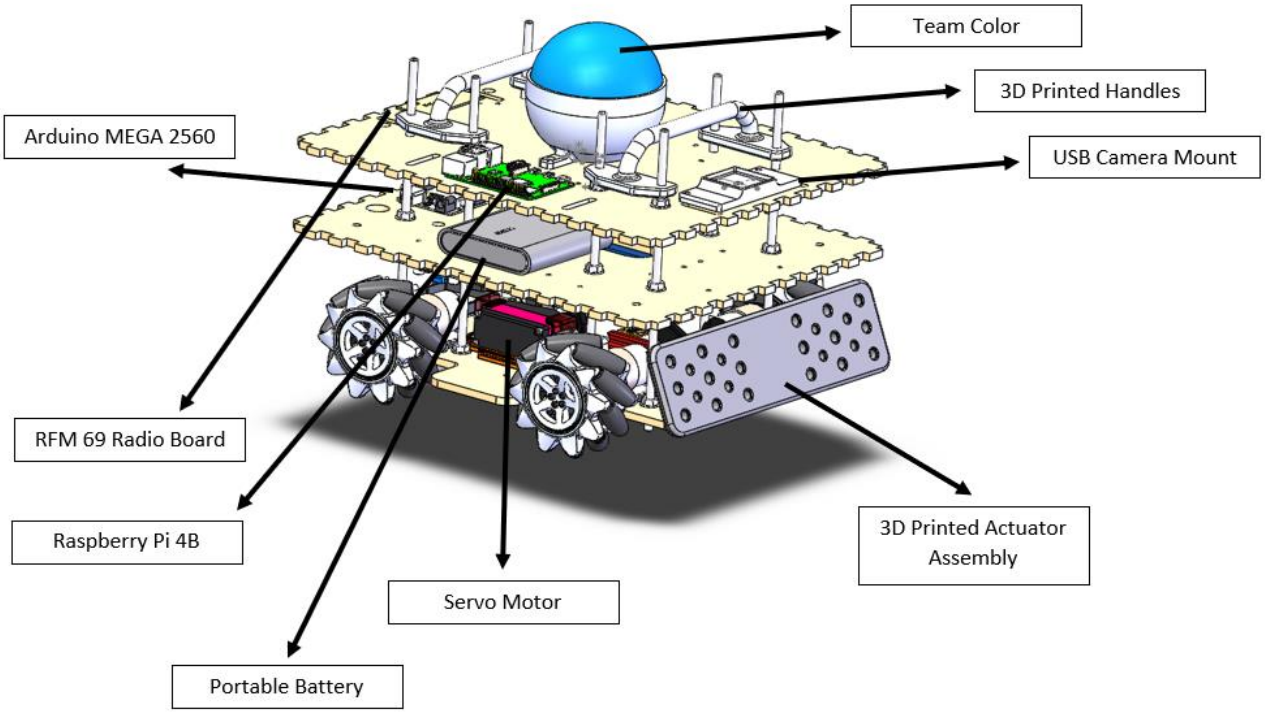


Figure 11: Isometric View of Final CAD Design

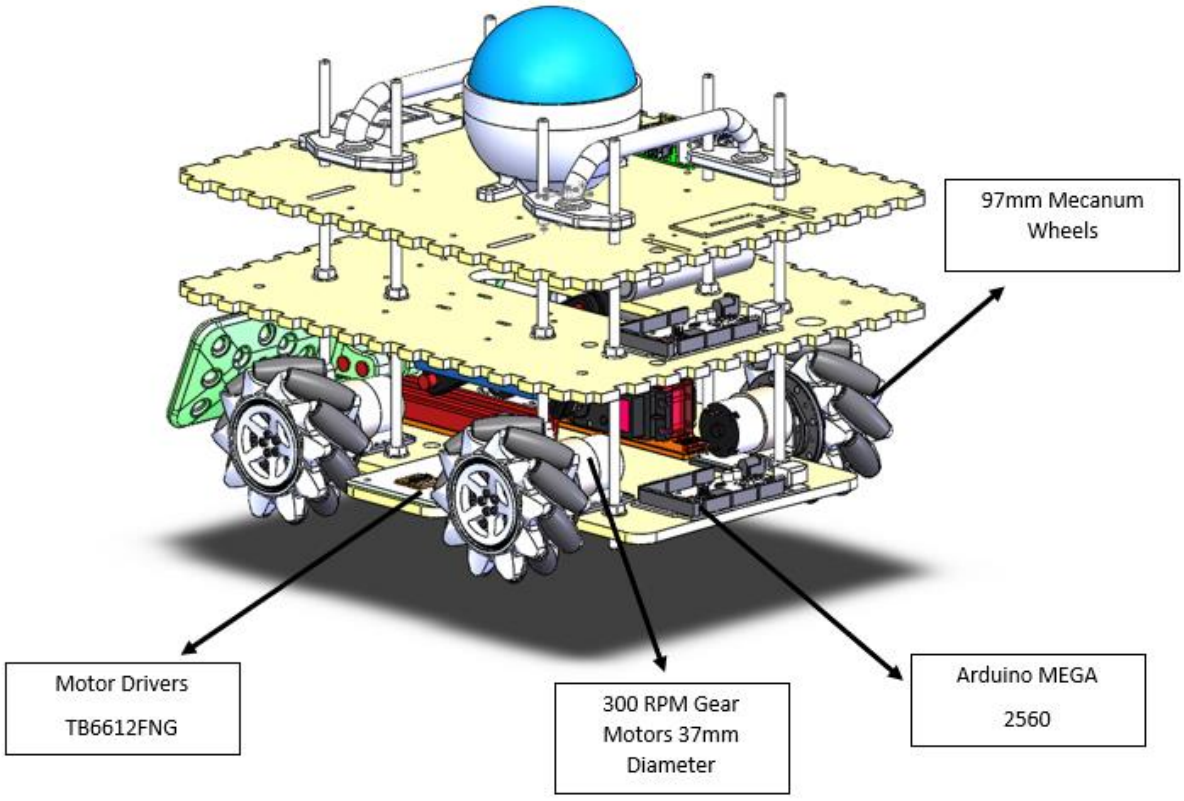


Figure 12: Mobility System on Final CAD Design

Bill of Materials (BOM)

Part	Part Number	Quantity	Price	Link
Mobility Subsystem				
Mecanum Wheels	-	1	$34	Link
Gear Motors	-	4	$48	Link
Motor Drivers	TB6612FNG	2	$10	Link
Arduino MEGA 2560	-	1	-	Arduino Kit #1
Motor Caps	-	4	$1.60	Amazon
Vision and Sensing				
Raspberry Pi 4B	-	1	-	Raspberry Pi Kit
IMU Sensor	MPU 6050	1	$3.50	Link
Raspberry Pi Camera	-	1	-	Raspberry Pi Kit
RFM 69 Radio	-	1	-	Link
Ball Manipulation				
Servo Motor	DS5160	1	$23	Link
Voltage Regulator	LM2596	3	$4.80	Link
Arduino MEGA 2560*	-	1	-	Arduino Kit #2
11.1V LiPo Battery	-	1	-	Professor Given
Miscellaneous				
1/4 in.x 2ft. x 4 ft. MDF Wood for Chassis	-	1	$14	Link
Portable Battery Pack	-	1	$30	BestBuy
Hardware	-	-	$8	-
3D Printing Filament	-	1	$21	Link
Total	$198			

CONCLUSION

Our team was able to demonstrate a variety of strengths and weaknesses this semester through this project. The strengths and accomplishments we achieved propelled us forward and provided motivation for us to continue to work hard. Some of these strengths include the functionality of our individual subsystems. Our team decided to work on perfecting three subsystems: mobility, vision, and ball manipulation. Each of these systems had their challenges, but it allowed us to practice working together and implement our individual strengths. In particular, we are proud of our design, chassis, wiring, and sensor mounts. Our hardware design decisions are proven to be robust, compact, and stable. This was very valuable to us, as Rocket League can be a very intensive game. Our team hoped that putting a lot of effort in the front-end would allow us to focus on the software and algorithm with low-probability of hardware failure. Unfortunately, this strength can also be considered a weakness. We spent many weeks on the hardware portion of the project and less time on integration of systems and software. This resulted in leaving our robot's ability to make decisions not robustly characterized. Our robot has logic, but does not utilize some of the control feedback decision-making methods such as PID control. A PID-controlled robot could use input from the radio system, IMU sensor, and USB camera at once, to make a targeted, accurate decision, under changing conditions in the field. Some of the benefits of PID control are it can be used to control DC motor speeds and reduce the time constant, making the robot faster at being able to respond to changes, much like a human- brain.The use of PID control was not approached this semester due to time constraints and our team's skills set.

The limitations our group faced impacted and influenced our interpretation of the project. Our group was mainly limited by time and lack of understanding on how we will ensure the robot is completely autonomous. This includes understanding the logic of the functions prior to implementation. This knowledge would help inform the guidance of the robot and use of the subsystems appropriately on the playing field. Secondly, this lack of foresight resulted in lack of well defined action items to accompany our project's milestones. If given the opportunity to reattempt the project, I believe we would be better equipped to plan and implement complex systems.

LESSONS LEARNED AND TIPS

Here are some tidbits of advice for the next MCEN Robotics I cohort.

1) Plan early and have at least one person working proactively on something that you will need in the future
2) Ask for help when you get stuck, you waste more time struggling
3) Divide and conquer is not always the best method, especially when everyone is doing something new. It will become evident when you begin integrating your systems, everyone might be able to work independently alongside each other but no one can work alone.

4) Save all your lab work-- those same techniques will help you with your team and personal projects
5) Have a backup plan on game day
6) Plan the logic before purchasing the hardware

FUTURE WORK

Our primary future work will include implementing the PID control. The PID control system would allow for the robot to have multiple inputs from sensors and position and a controlled output for our DC motor speeds. Another opportunity for future work includes placing the camera on a swivel with a servo motor to facilitate 360 degree information and ball tracking. We already practiced the method of implementing opencv on the USB camera including finding and tracking the center of the ball. This would allow for finding the position of the ball in real time, dynamic object capture and fast vision tracking. Finally, in lieu of a servo to support the linear actuator, we could potentially utilize a rack and pinion method with a DC motor, which would lead to increased power and speed. Our servo robotic arm is a great solution for precise, “jab” like actuation, but not necessarily ideal for the game of soccer.

APPENDIX

Milestones Chart

Milestone	Due Date	Instructor Assignment

M1	9/22/2021	Prototypes and design plans due
Draft Initial Requirements from Project Document		TRUE
Determine materials and manufacturing methods		TRUE
Basic CAD Design		TRUE
M2	10/29/2021	Demonstration of Independent System Functionality
Mechanical Subsystem Running Independently		TRUE
Radio Subsystem Running Independently		TRUE
Vision Subsystem Running Independently		TRUE
Ball Mechanism Subsystem Running Independently		TRUE
M3	11/12/2021	Demonstrate Complete Functionality
Successful Hardware System Integration		TRUE
Successful Software System Integration (Arduino and RPi code)		TRUE
Secure outer enclosure with bumper and handles		TRUE
Navigate a Course (Stay in bounds, move autonomously)		FALSE
Interact with ball and goals		FALSE
M4	11/29/2021	Successful Practice Run for Robot
Robot can maneuver around field successfully		FALSE
Robot can score around field successfully		FALSE
M5	12/6/2021	Project Runoff
Robot can successfully participate in the game of Rocket League		FALSE
Robot can be randomly placed anywhere/on any team and play		FALSE
M6	12/9/2021	Project Report due
Detailed outline of each of our subsystems		TRUE
Schematics, mechanical drawings, labeled digital photos		TRUE
Clear, complete and concise outline of goals		TRUE
Section on design process		TRUE
Well commented code for Arduino and Rpi		TRUE
Summary and Lessons Learned Section		TRUE

Additional Links

Raspberry Pi (master) Arduino (slave) I2C communication with WiringPi - The Robotics Back-End (roboticsbackend.com)

Raspberry Pi Arduino Serial Communication - Everything You Need To Know - The Robotics Back-End (roboticsbackend.com)

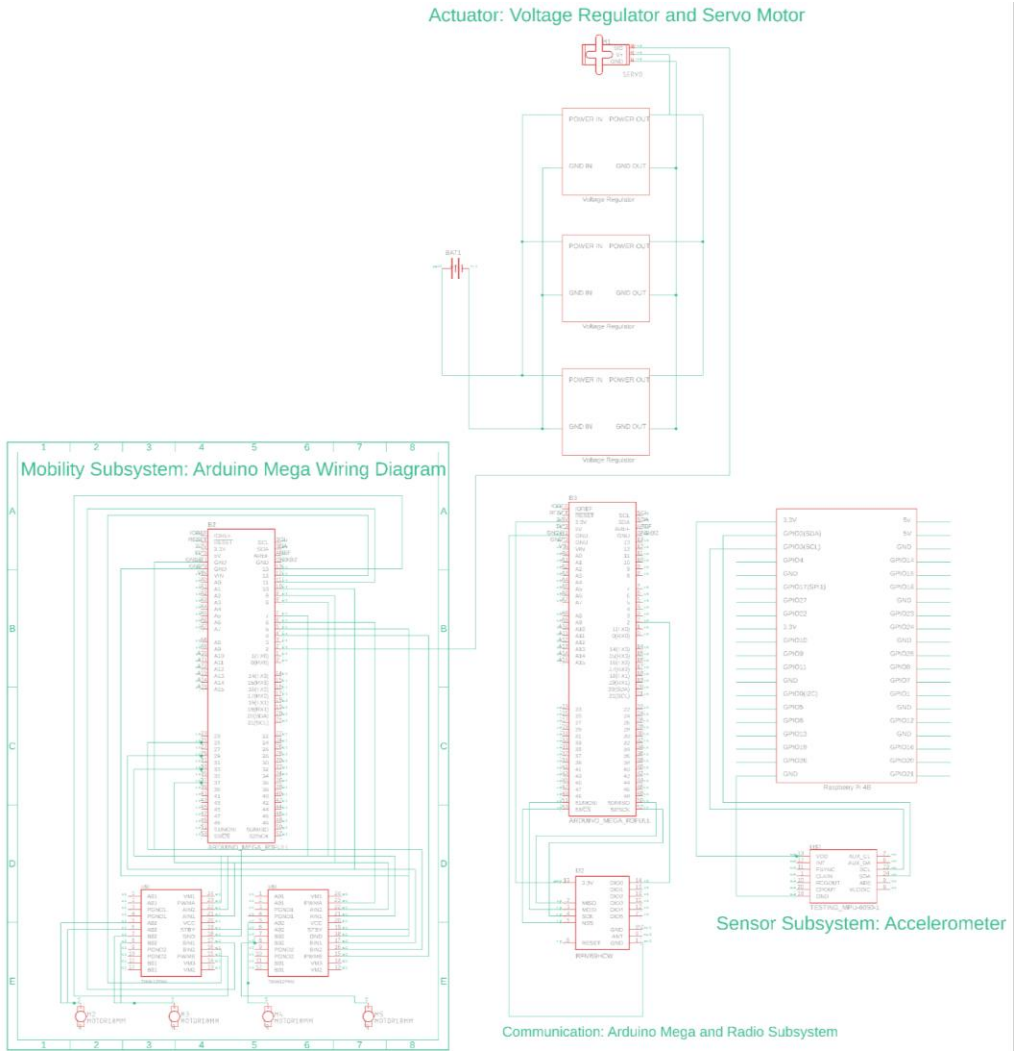
https://seamonsters-2605.github.io/archive/mecanum/

Git Repository

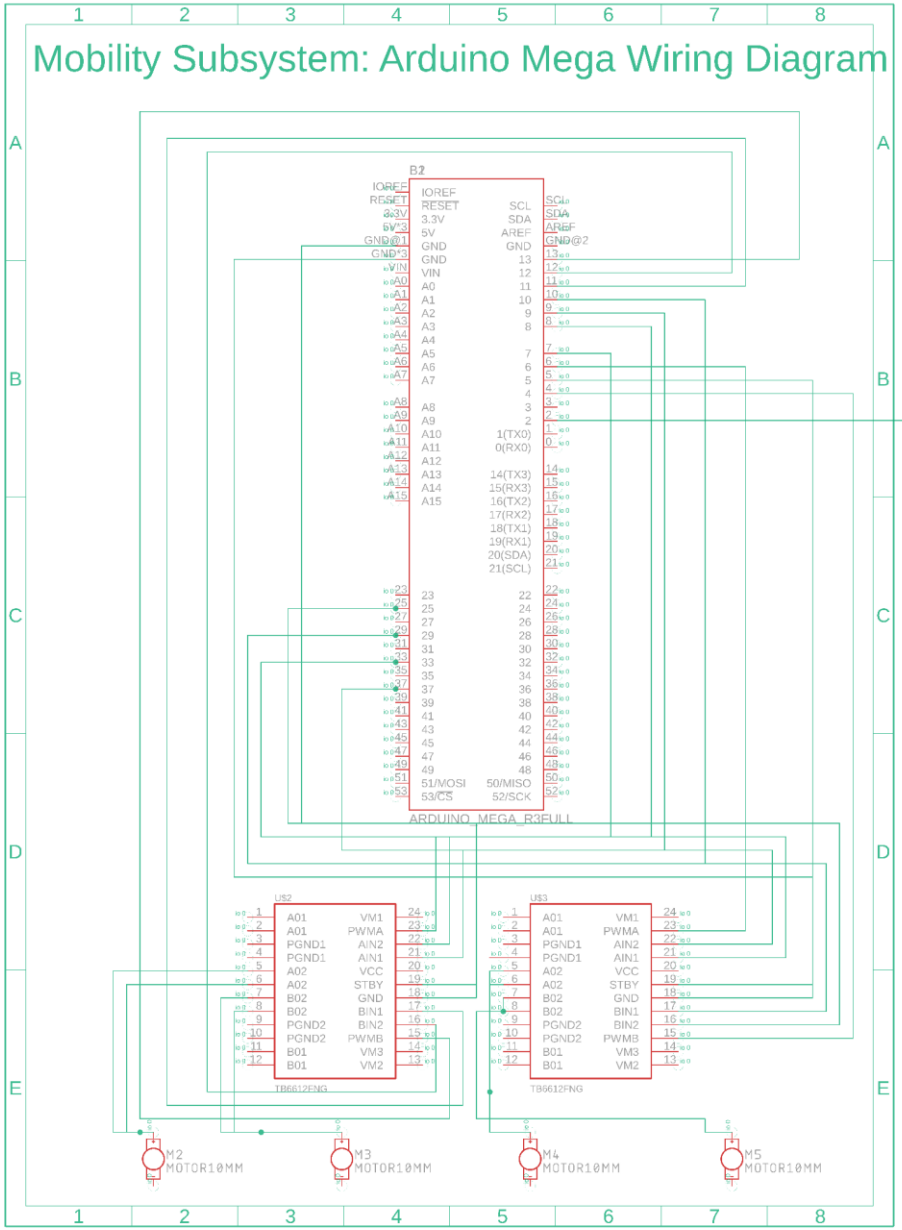
https://github.com/Maxwell-P-Anderson/MCEN-5115-Team-Dwayne-J.git

Final Robotic Schematics

Schematic 1: Comprehensive Robot Subsystems

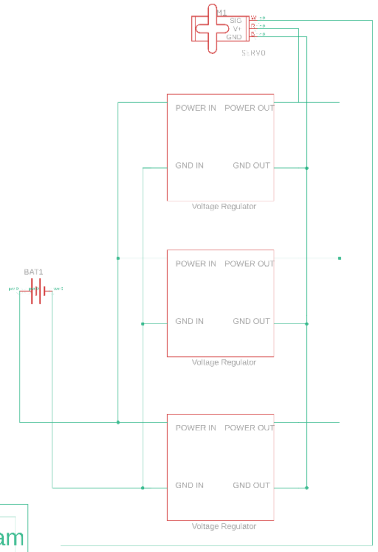


Schematic 2: Mobility Subsystem

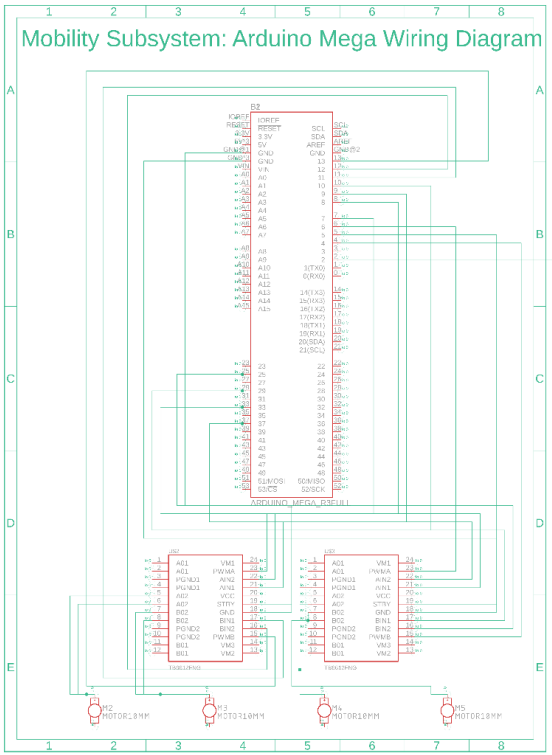


Schematic 3: Actuator Subsystem

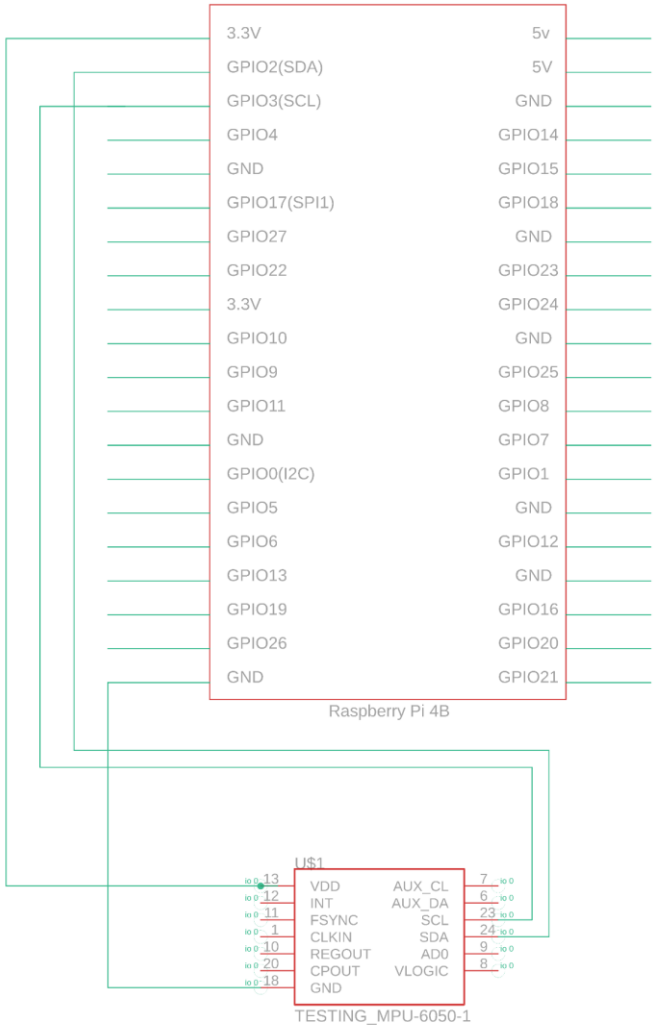
Actuator: Voltage Regulator and Servo Motor



Mobility Subsystem: Arduino Mega Wiring Diagram

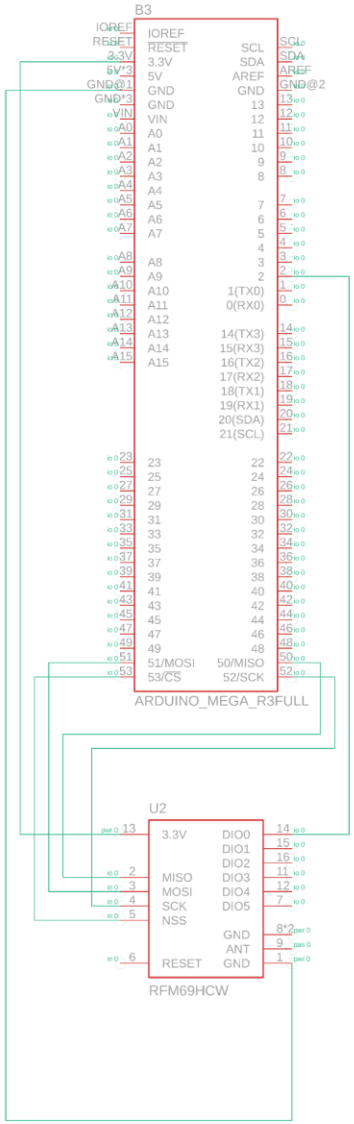


Schematic 4: Sensor Subsystem



Sensor Subsystem: Accelerometer

Schematic 5: Communication Subsystem



Communication: Arduino Mega and Radio Subsystem