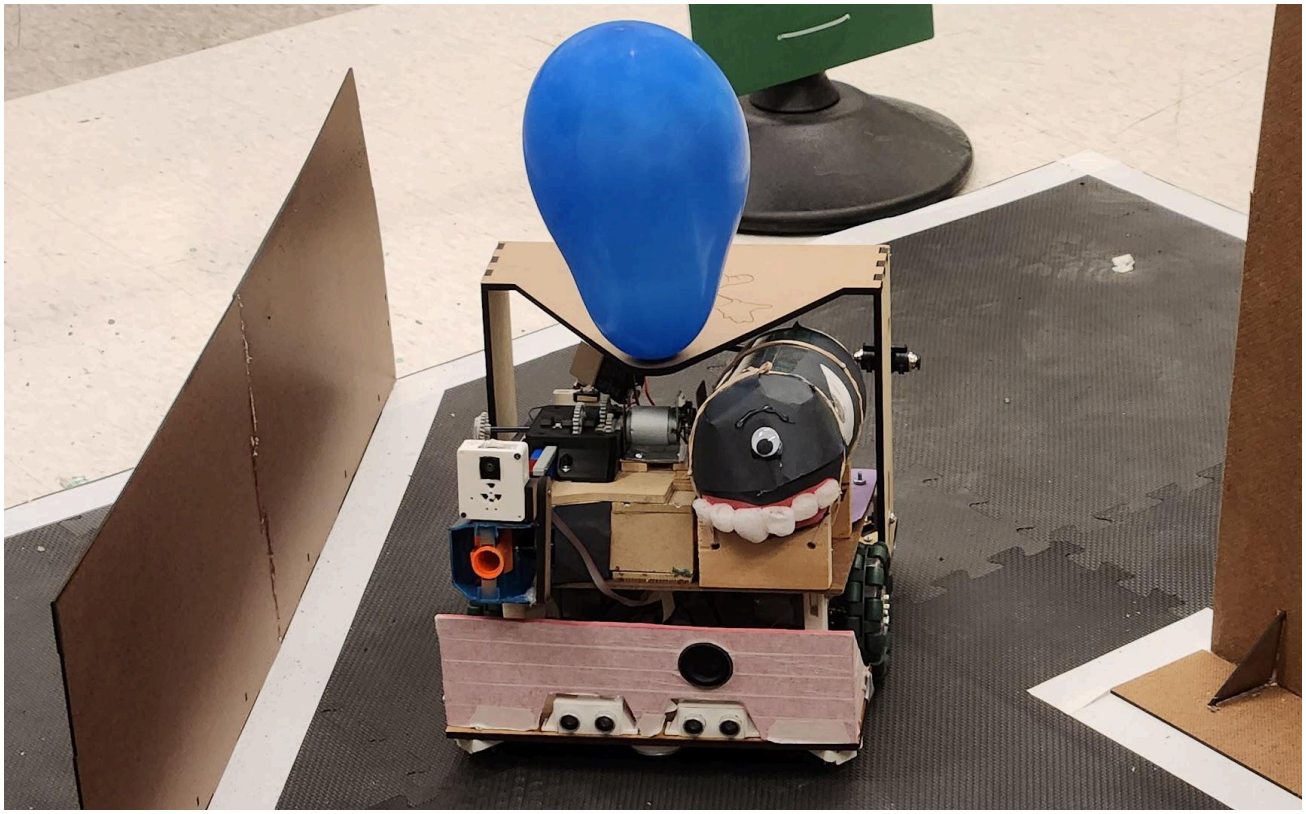


MCEN 4115/5115:

Mechatronics and Robotics I

Team N.E.R.F. Final Project Report

12/12/2022

Palmer Dick-Montez
Jakshil Rakeshkumar Gandhi
Anna Lugthart
Garrett Schmitz
Mangala Sneha Srinivasan
Yugendran Subramani

Table of Contents

Abstract

The purpose of this competition is to create an autonomous robot that can compete in a real life version of the game Mario Kart: Battle Royale!®. Its goal is to fire projectiles to land as many hits on a balloon mounted to the top of an opposing robot without taking hits itself. The robot must be a fully autonomous, stand-alone entity. We are limited to the materials in our kit and an additional budget of $200. The robot must fit within a 12”x14”x12” box, with a balloon mounted at exactly 12” from the ground in the center of the robot. The robot must be able to navigate the course, identify the opposing robot, and fire a projectile with enough accuracy to hit its balloon.

Our robot incorporates vision, distance and color sensors, motor control, and pneumatics to achieve this goal. Using a Pixy2 camera, it identifies features on the course and the opposing robot. Using this information, it drives into position and uses its pneumatic shooter to fire nerf darts at the opposing robot’s balloon. When out of darts, it returns to the reload location. The robot also integrated a graduate mechatronics design project to add a speaker and sound system. This was done to add a fun extra touch to the robot while it was competing in a match. The sound system is not covered in detail because it is extraneous to the robot. Possible additions could include logic to avoid the opposing robot and motion-tracking to land hits on the opposing robot as it moves. The shooting and driving operations could also include PID control in future iterations.

Design Process

Our design process throughout this project has been one of ideation and iteration. Since we’re working with so many systems that are new to our team members, trying to fully plan out their implementation is impractical. At each step of our design, we’ve come up with a plan based on what we know. While this may have come at the detriment of visuals and aesthetics of the design, it did ensure that each sub-system underwent rigorous testing and iteration before it underwent systems integration.

Our first iterations included plans for using certain components and rough layout drawings to work out how our robot may conceptually function. After this, we sourced components, tested them, and planned again. This included things such as ideating and testing how to create a rack and pinion to control the reloading of individual darts into the shooter. During this period we focused on individual systems and ensuring that they would be able to work on their own. Many of the products from this period were very rough as they were created by hand in the woodworking shop. Following this period, we iterated on what we found and transitioned several components to more polished designs with higher tolerances. We also transitioned to finding how the physical components would work with each other, and completing any modifications that were needed. Throughout this process, the core design idea that we focused on was testing. This meant that even in the final product, some of the mountings were not the most aesthetic. This is because we often chose form over function, and many of the early concepts carried over through the designs because they worked well.

If we were to develop our robot further, now knowing how all the components work together, we could create one final, streamlined design. This would include taking aesthetics into account earlier into the design process. While this design process did allow us to have a successful robot at the runoff, it is not without its problems. In some cases, because a bottom-up design approach was taken, necessary interfacing elements were difficult to operate. The best examples of this are working with the Arduino

processors or trying to reload the magazine. The connections for these systems are difficult to reach places, which potentially could have been avoided if a top-down design approach had been used. However, after testing this robot, we may just opt to design another using a different design strategy.

System Architecture

At the beginning of the project, we broke the design of our robot into 3 sections: the drive system, sensor system, and shooter system. All of these are contained within the chassis. The drive system contains all components relating to the robot's movement around the playing field. The sensor system contains all components used to determine the position of our robot, the position of the opposing robot, or anything else on the playing field. Finally, the shooter system contains all components used to load and fire projectiles. This use of subsystems helped to enable our design to be successful and be flexible throughout the design process.

Table 1: System Architecture Decision

System Architecture	Rationale	Option
Microprocessors	Use multiple microprocessors so driving and shooting tasks can be accomplished independently	Arduino Uno Arduino Mega
Shooting Mechanism	Pneumatic launcher for good speed and repeatability	Modified NERF gun with rack and pinion reload system
Movement	Fast turning and driving capabilities to ensure robot can target balloon	12V DC motor VEX Robotics Omni Wheel
Sensors	Determine robot orientation, and location; allow the robot to navigate the environment without exiting the field of play	HC-SR04 Ultrasonic Sensors TCRT5000 Tape Sensors
Vision	Determine the location of opponent balloons and the side of the field	Pixy2 Camera trained for blue (balloon) and red (field side)
Power Source	A power source to drive the motors, solenoid, and the Arduino microcontrollers	11.1V Lithium Polymer Battery Portable power bank
Chassis	Provide a stable platform for vision, shooting, and driving systems	¼" laser cut MDF ⅛" particle board 3D printed supports

This system architecture was implemented into the below high-level system design. The Arduino Mega 2560 was changed into the control center for the robot after repeated failures to establish Serial Communication between a RaspberryPi 3b and the Arduinos.

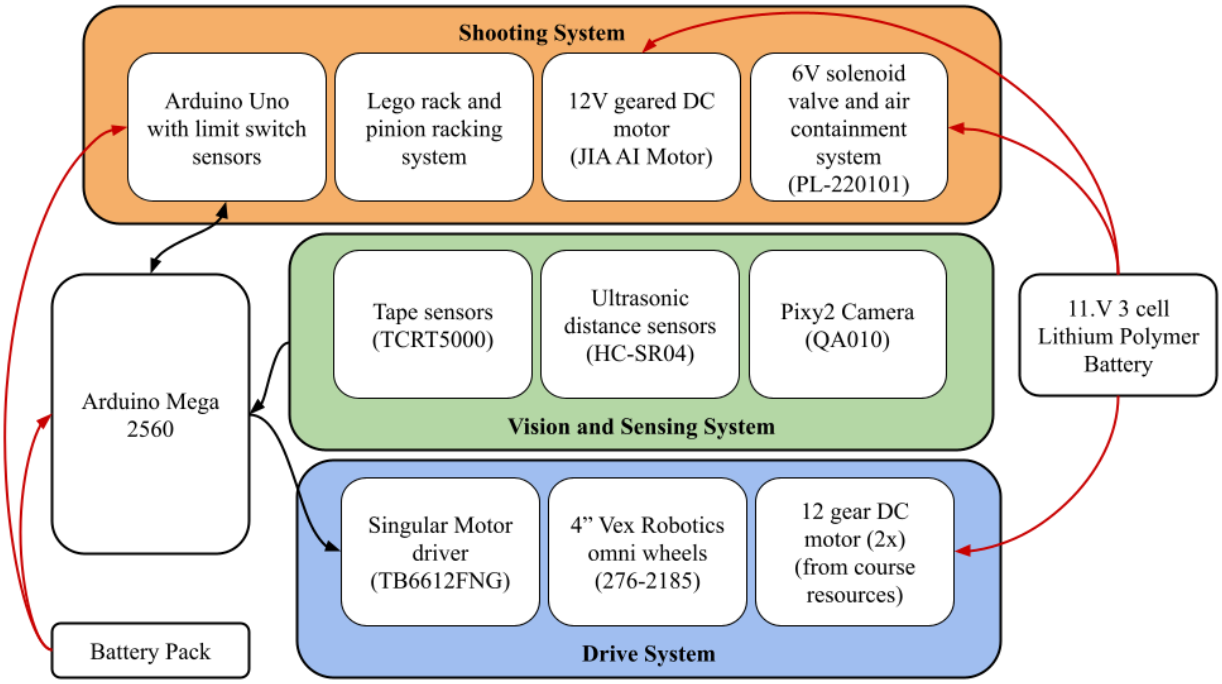


Figure 1: High-Level Systems block diagram

The above block diagram represents the three major subsystems and how they interface. Power is noted by the red arrows, while information is noted by the black arrows. The direction of the arrow indicates how the power or information flows through the system. The Arduino's have bi-direction communication to ensure that commands are not sent from the Arduino Mega to the Arduino Uno while the latter is still trying to execute previous commands. The Arduino Mega takes in information from the Vision and Sensing System. It uses this information to control the Drive System and send commands to the Shooting System. The primary control of the Shooting System is done by the Arduino Uno instead of the Arduino Mega so that the robot can execute multiple functions at the same time (ie. reload and turn).

Chassis

The chassis is made with a quarter inch thick Medium Density Fibreboard (MDF). MDF was chosen due to its cost effectiveness and machinability. The chassis was machined using a laser cutter after being modeled in Solidworks within the given dimensions (Epilog: Fusion Pro). There are two levels in the chassis. The base level (pictured above) houses a Raspberry Pi 3b, an Arduino Mega, two driving motors, two omni wheels, and two stationary guiding wheels. This level also houses the Lipo Battery (the main power source) as well as the sound system. The sound system was integrated into the removable front panel. The two large 3D printed pillars allow for the shooting mechanism (the top level) to be pivoted when combined with the slot guides. These enable easy user adjustment of the shooter angle, while also ensuring that the angle can be secured using a set of bolts and nuts. The side posts connect to

the lower base and the balloon platform by using finger joints. The motors for the drive system were chosen to allow for a high torque with a moderately high speed. This will enable the robot to quickly get into position for the competition. The high torque enabled us to ignore the weight of the robot, and any deformation that the foam of the course may undergo. These motors were driven using a Sparkfun TB6612FNG motor driver. This allows the team to drive the motors using the 11.1 V LiPo battery, while maintaining control from the Arduino Mega by setting individual pins high or low to control the direction of the motor and providing a PWM signal to control the speed of the motor. Motor speeds were set based on experimental testing with live code. This was done to ensure that the robot would be able to react fast enough by starting and stopping the motors.

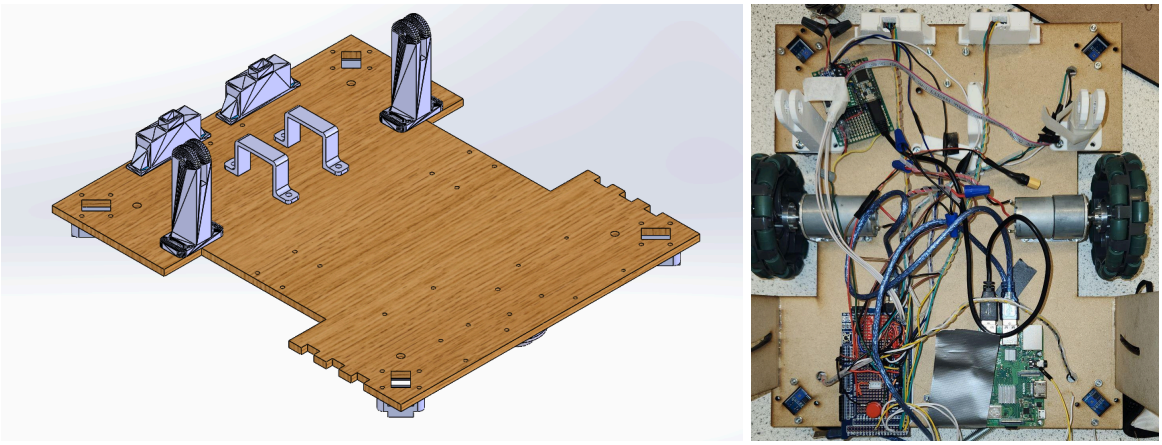


Figure 2: CAD of the lower chassis and components (left) and physical robot (right)

These were chosen because they are easily machinable on a laser cutting system, and will allow for a strong and easy to affix attachment. The finger joints help to ensure the alignment of the side walls during the assembly process. In addition to the driving omni wheels, there is a roller bearing located at the front and rear of the vehicle. These were selected so that the movement of the robot wouldn't be impacted, while allowing for additional stability and support. Since these supports also rotate, they reduce friction when compared with a sliding contact. Larger square alignment features were selected for the distance sensors to ensure that misalignment was reduced as much as possible. Finally, the line detection sensors were affixed to the bottom of the chassis using 3D printed housings. Cutouts above the housing allow for easy adjustment of sensitivity and tolerances. This was especially important for the variable testing conditions as the course became dirty throughout the semester.

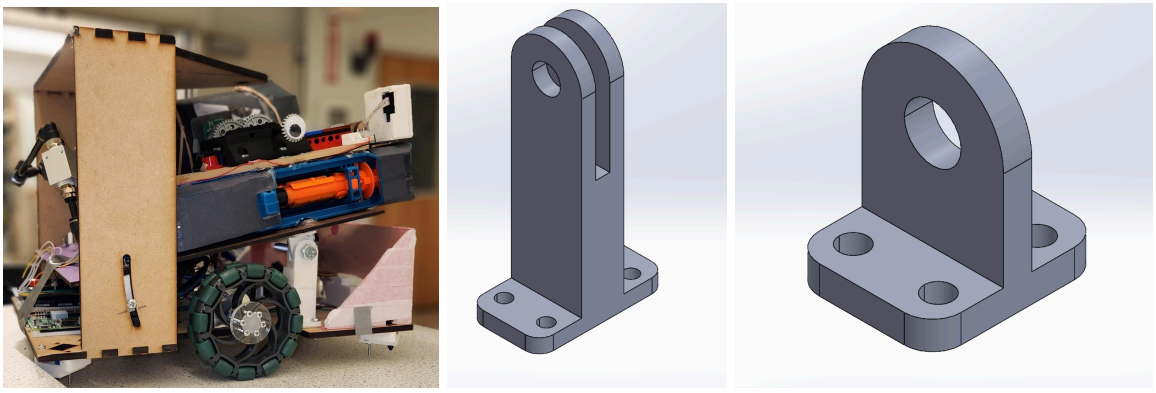


Figure 3: The pivot system used to angle the shooter. The assembled finger jointed side walls (left) are used to anchor the upper shooter plate to the base using 3D printed control joints (middle and right).

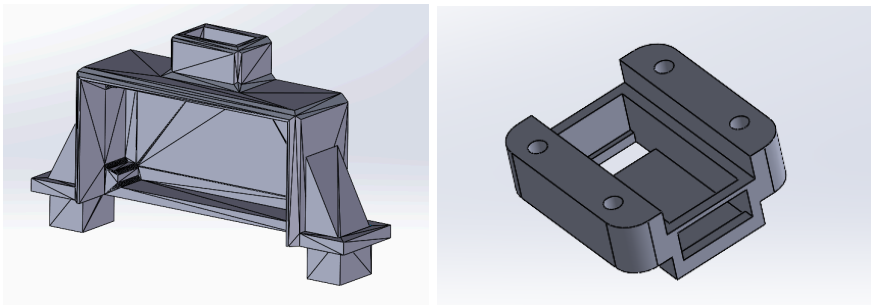


Figure 4: The distance sensor (left) and line detection sensor (right) housings. The distance sensor housing includes a press fit cover (not shown).

The shooting mechanism is located on the upper level, along with a soda bottle that serves as a reservoir for the pneumatic system and an Arduino UNO that controls the pressure valves and the shooter. The balloon is mounted on a platform that is exactly one foot off the ground. The shooter platform is secured using the side panels and the 3D printed control joint. The balloon platform is made with 3 rectangular pieces of MDF and is connected with finger joints. This platform was designed so that the balloon sits above the center of the robot. Since the team focused on shooter iteration, there is no CAD of the shooter platform as it was made by anchoring the shooter to a small piece of particle board. Wire routing holes and shooter attachment points were drilled into the particle board. The air canister was secured to the platform by using two scrap pieces of wood and rubber bands. The rubber bands allow the balloon to expand and contract without risking becoming loose while the wood supports allow the bottle to be positioned above the shooter magazine without causing any interference.

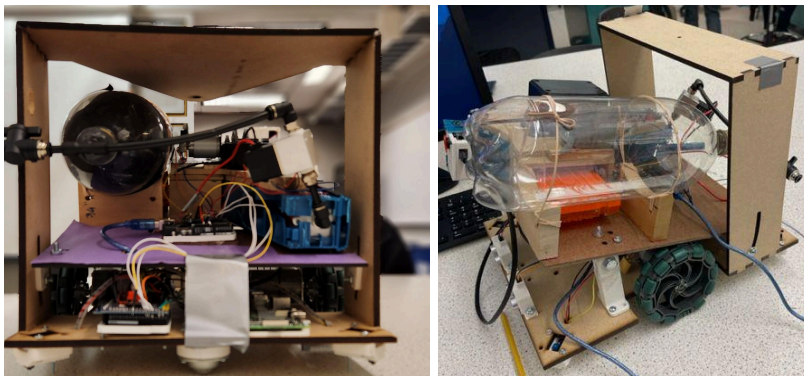


Figure 5: Rear (left) and side (right) views of the shooter platform. The side view shows an earlier iteration of the balloon platform, but the magazine clearance is visible.

Sensors

Being able to sense the environment is vitally important to the robot and for the competition. Based on the competition objective and the course layout we determined that we would need to be able to determine distances, detect lines, and use computer vision. In addition to these basic functions, because of our chosen shooter mechanism, we will need to include sensors that can detect when the rack and pinion is at the far extents of motion in the forward and backward directions.

Line Detection:

The IR line follower sensors were used in our line follower robot to detect white lines and to ensure that the robot stays in the arena. The working of the sensor is such that when infrared rays fall on the white line, it is reflected back and is caught by the photodiode thus resulting in white line detection (Voltage change). When the infrared rays fall on the black mat, a minimal fraction of the rays are reflected back and caught by the photodiode.

Figure 6: IR line detection sensor (left) and ultrasonic distance sensor (right).

Distance Detection:

The ultrasonic distance sensors were used to detect the walls present in the arena and then calculate the distance. By detecting the walls, the robot is then able to orient itself. The working of the sensor involves transmitting a high frequency ultrasonic sound which is then bounced off any immediate line-of-sight object and then received by the receiver. The time difference between transmission and reception is calculated by the control unit. Using time, distance between sensor and object is calculated in code.

Vision and object tracking:

The Pixy2 camera was used to identify the side of the course the robot was located on and the position of the enemy balloon. This was done by training the Pixy2 using the red reloading area flag and sample balloons. This was then integrated into the Arduino Mega using the included cable and the <Pixy2.h> library. This enabled the robot to detect the size and location of the two colors of interest. The Pixy2 camera had to be repeatedly re-trained as it was sensitive to fluctuations in the light level of the environment.

SS-5GL-F-3 Limit Switch:

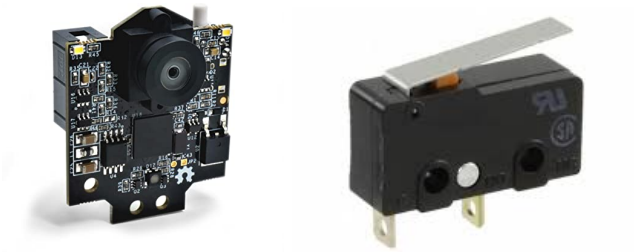


Figure 7: The Pix2 camera (left) and the limit switch used in the shooter (right).

Shooter tracking:

These sensors were used in the shooting mechanism of the robot. One was placed at the forward stop position and another was placed at the backward stop position. These sensors were then used to cut power to the motor to prevent damage to the device, and to allow the controller to know when the device was ready to fire.

Shooter

For the shooter, we decided to use a pneumatic dart launching mechanism. We also considered spinning wheels / fly wheels that would either launch darts or balls, but settled on the pneumatic system because we were more confident it could achieve the accuracy and repeatability required to hit our target on the opposing robot. Additionally, we had concerns about ensuring that the flywheels were spinning at the same speed so that it wouldn't introduce spin onto the projectiles. By using pneumatics with a solenoid controlled release, we are able to ensure a consistent burst of air that can be precisely controlled using the Arduino Uno. For the pressure reservoir we decided to use a 2L soda bottle. This was chosen because they are readily available, can withstand up to 150 psi, and will be able to hold a large volume of air. This will enable us to go through many reloads on one tank of air, without seeing a noticeable drop off in the force of the dart as long as the system is properly sealed.

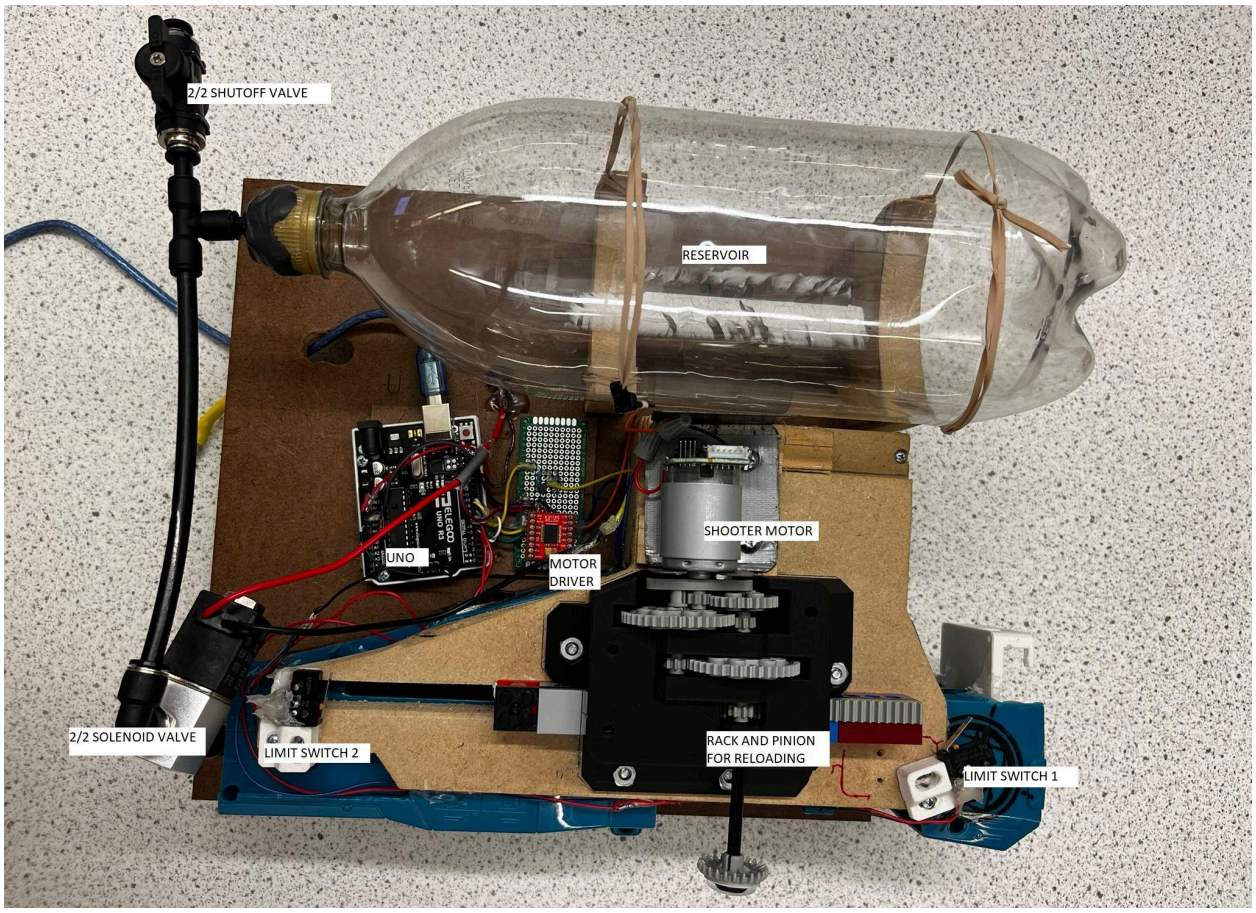


Figure 8: Shooter Mechanism

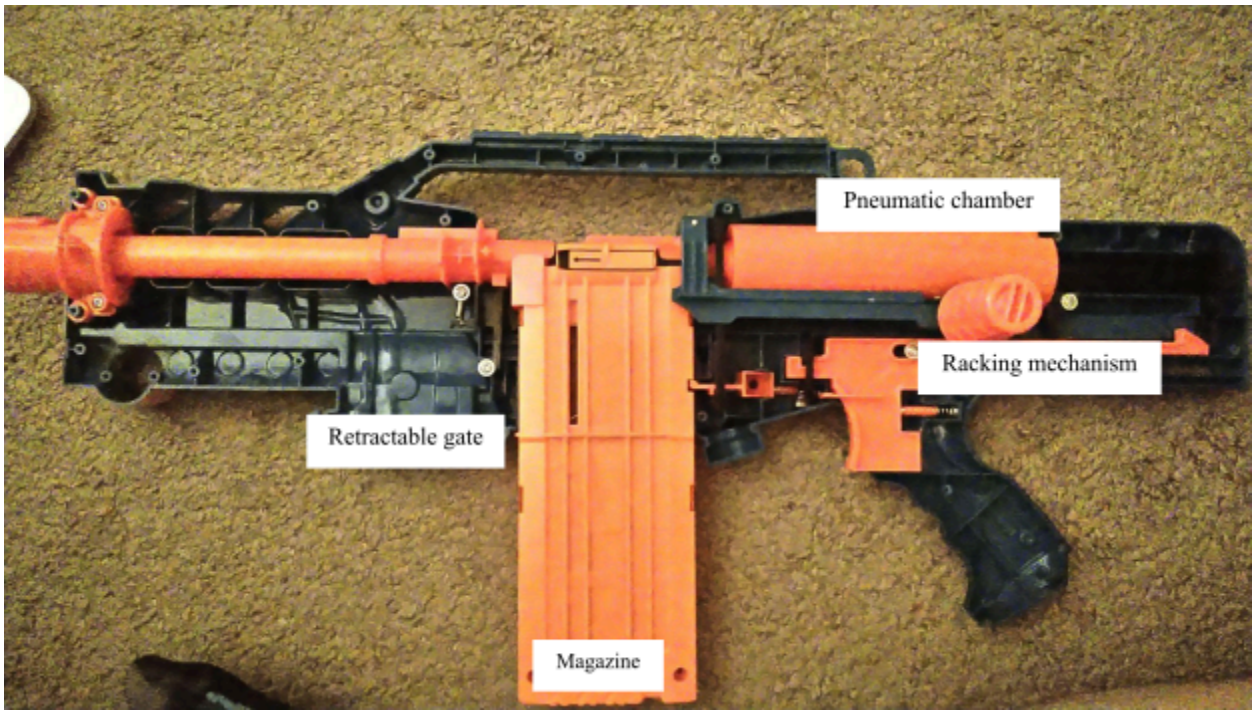


Figure 9: Internal components of the NERF Gun. Includes magazine, barrel, and piston.

Dart Loading System

The majority of our dart loading system comes from a modified NERF rifle we purchased at Goodwill. We removed the unneeded sections and adapted it to be powered by motor instead of by hand. This was accomplished using a lego rack and pinion system using a 75:1 gear ratio to ensure that the darts are able to be loaded. The rack and pinion attaches to the racking mechanism using a lego axle that replaced the normal hand operated slide mechanism. This system was raised by using ¼" MDF so that the gearing system was fully level on top of the injection molded plastic exterior of the NERF rifle. 12 darts are stored in a magazine. In order for a dart to be launched, it is pushed forward by the rack and pinion. This rifle seals the dart inside the barrel so that when it is pneumatically fired, air is directed behind the dart with minimal leakage to launch it from the shooter. The dart does not always reliably seat against the piston, so a retractable gate driven by the loading mechanism pushes it into position, then drops away to allow the dart into the barrel.

Motor and Gearing

In order to drive this mechanism, we used a motor-powered rack and pinion. However, we didn't have a motor with enough torque to drive the mechanism directly. We considered purchasing a stepper motor, but decided we could instead gear a faster motor to have the output drive shaft have a slower rpm, but have more torque. We used LEGO shafts and gears inside of a 3D-printed gearbox. While most of our 3D printed parts were made on the ITLL LulzBots, we used an Ender-3 V2 owned by one of our teammates for the gearbox to achieve a higher accuracy print. This allowed the gear axles to spin freely but with minimal play.

We did later discover that the mechanism required lubrication when one of our LEGO gears experienced catastrophic failure (See Figure X). We also discovered that the pinion gear was subject to

catastrophic failure if the motor overran the stopping location at either the forward or backward extent of motion. After applying some sewing machine oil to the shafts and adjusting the location of limit switches, the mechanism ran far more smoothly. It did, however, also highlight the problem of "double loading." This occurs when the dart in the firing mechanism isn't fired. This causes the next dart to become half-loaded. This greatly increases the force on the pinion gear and may cause it to fail. Additional safety measures wire implemented in the code to ensure that double loading does not occur during the runoff.

Figure 10: Catastrophic Gear Failure

The gearbox drives a moving rack which is attached to the dart loading mechanism by a LEGO shaft. They are easy to replace if broken. This rack activates limit switches at the fully-loaded and fully-unloaded positions. When the limit switches are pressed a signal is sent to the arduino which stops the motor. Once a dart has been loaded into the shooter, it is ready to be fired by the pneumatics system.

Pneumatics

Figure 11: 3D Printed Pneumatic Adapter.

Once a dart is loaded into the barrel, it is fired using our pneumatic system. We used a 2 liter soda bottle as a pressure vessel. Using a 2/2 mechanically actuated valve, we pressurize the bottle to 80 psi using the compressed air available in the ITLL. The bottle is connected to the loading piston via a 2/2 spring return nominally closed solenoid operated valve and 3D-printed fitting. To fire, the solenoid valve is powered with 11.1V for 20 milliseconds. This sends a burst of air into the hole in the back of the dart which launches the dart. The duration of this burst was tested and modified. 20 milliseconds offered good

power without compromising the ability to actuate the solenoid, which can happen if the burst is too short..

The major challenge was to cannibalize the NERF Gun with a different mode of actuation, pneumatics. There was a fear of losing air pressure from leakage during the period of the game. However, by using a custom 3D printed pneumatic adapter along with placing our 2/2 solenoid valve close to the shooter, we were able to minimize the amount of air required per shot. We were also able to minimize this by using teflon tape at the air connections. This also improved the robot's ability to shoot as it reduced the pressure required to fire darts as well as the loss of air velocity due to frictional forces in the line. The only detectable leak within the system was at the junction between the air reservoir and the pneumatic tubing. This leak was deemed minimal, and did not appear to have a large impact on the functionality of the shooting mechanism.

Microcontrollers and Communication

The team initially planned on using an Arduino Mega, an Arduino Uno, and a RaspberryPi 3b to control the robot. The RaspberryPi was going to aggregate data from the two Arduino boards and send commands to them that they would be able to execute. The goal of this was to reduce "waiting times" where one system has to halt for operations in the other system. The plan to communicate between the three controllers was to use Serial communication through the USB data bus. However, after nearly a week of testing, there were unworkable communication delays between the devices. This resulted in the robot losing track of where it was in the program, or commands being missed entirely. The team then decided to remove the RaspberryPi from the communication system. The final implementation of the robot was to use the Arduino Mega to control the motors and contain the bulk of the programming and logic while the Arduino Uno only controlled the shooting mechanism and associated checks to reduce risk of damage. Since the change had to be made rapidly, pin to pin communication was chosen. Each Arduino monitors a set of pins which correlate to different potential commands and a completion verification. This method was chosen because it is fast and reliable. In addition to the Arduino units, a teensy microcontroller was added to control the speaker and sound system. This was integrated from a graduate project and does not have a bearing on the overall robot or project. It was added to include extra personality and flair to the robot during the runoff. The circuit diagrams, with included pin locations, are provided within the Appendix. The RaspberryPi 3b was left attached to the robot as a way to easily distribute power to the Arduino Mega and the Arduino Uno. It did not serve any purpose other than power distribution.

Control Algorithms

To enable easier troubleshooting, the code was broken into several distinct algorithms. Any extraneous tasks, such as initializing variables and checking to ensure the robot is in a ready-to-fire state at the start of a run is taken care of by the initialization state. Since this state is simple, it is not included in the algorithms. This state simply waits with the shooter in the backwards most position until the button on the Arduino Mega 2560 shield has been pressed. Once the button has been pressed the shooter will move to the forward ready to fire position and the code will resume with the next algorithm.

The first actual algorithm employed by the code is the "orientation" algorithm. This is a fairly simple program that ensures the robot faces the correct direction before continuing on with the rest of the

code. Since the robot can start in a random orientation, a pair of distance sensors are used to align the front of the robot to face and be parallel to the wall at the starting location. The robot will rotate until it receives two consecutive sensor readings that are within a specified minimum distance. In this case, the minimum distance was chosen experimentally to be 0.5 in. If only one sensor reading was employed, instead of two consecutive, the code would often falsely identify the robot as being straight. While the other programs in the code could make up for this discrepancy, it could potentially lead to the robot exiting the course during the positioning segment of the code if it was too far off from actually being straight. After the first sensor reading is taken from the distance sensors, a direction is chosen. If one reading is less than the other, that direction is chosen, otherwise the robot will default to rotating clockwise. This is important because the distance sensors were limited in code to avoid any unnecessary delays between pings. If it takes too long to return a distance value, the returned value is instead -2 (to differentiate between when the distance sensors are active and when they are inactive as -1 is returned if they are inactive). This helps to improve reliability and speed of code execution. Once the robot has been oriented, the code transitions to the "positioning" algorithm.

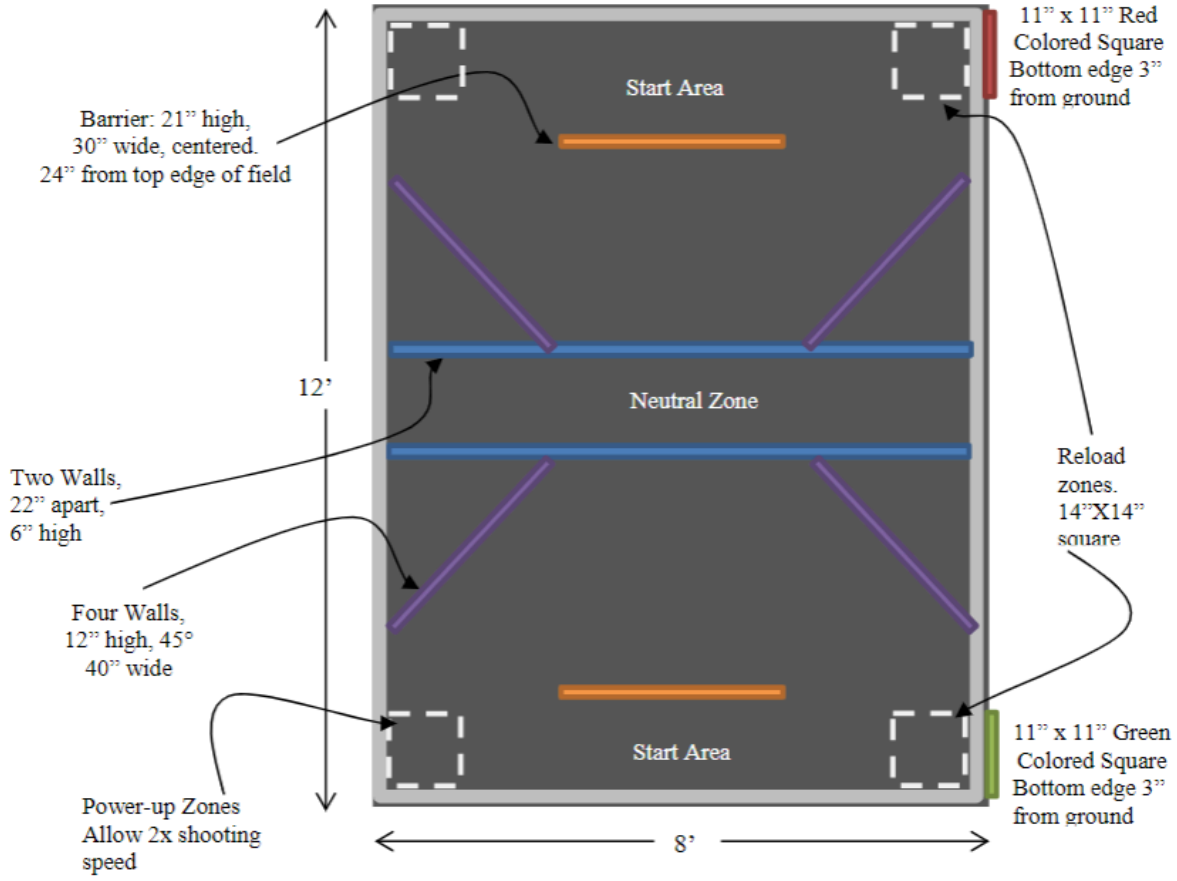


Figure 12: The arena layout provided for clarity

The positioning algorithm is designed to direct the robot to the reload corner of the respective side that it was placed on. This was chosen so that the robot can be reloaded when necessary without the need to move. Additionally, since the shooter has the power to cover the entire course, and there is relatively minimal height dropoff of a given dart if the air canister is above 50 psi, this allows a fast reload

option that other teams may not have had access to. To reach the respective reload corner the robot must first reverse from the start area until it reaches the backmost line with both of the line detection sensors. By requiring both, this minimizes any false positives that may occur. From here, the robot turns counterclockwise until the front left line detector is triggered. This assumes that the robot is on the red side. This was chosen as the red color was significantly easier to detect than the green color. From here, the robot determines if it is on the red side or if it is on the green side. As the robot rotated to face the red colored square (assuming it was on the red side), it monitors the Pixy2 camera channel for red input. If any red was detected during the rotation, the robot will execute the red side code. If not, it will execute the green side code. The code principals are the same for both but many of the directions are reversed. For the red side, the robot will use the front left line sensor to track the backline to the reload corner. This is done by having the robot drive the left wheel forward when the sensor reads white and drive the right wheel forward when it detects black. This is continued until the front right line detection sensor is tripped, at which point the robot rotates clockwise until the rear right line sensor is triggered. This will have the robot face down the length of the course. From the green side, the conceptual procedure is the same but the motor directions and line sensors are changed. The robot follows the line in reverse until it reaches the green reload corner before it turns to face toward the red reload corner. It does this by driving the wheels in reverse in a similar manner to how they were previously driven forward, and monitoring the back left line detection sensor. The corner is detected when the back right line detection sensor is triggered, and the robot proceeds to the "targeting" code when the front right line detection sensor is triggered. This ensures that the robot always moves to the right side of the course and faces the opposing side.

The targeting code seeks to find and shoot enemy balloons. This is done in a series of stages. First the robot checks to see if there is a balloon detected within the visual field of the Pixy2 camera. If no balloon is detected the robot will begin a "sweeping" procedure. The direction of this sweep is dependent on the side that the robot was placed on. If the robot is on the red side, the initial sweep is in the clockwise direction, while if the robot is on the green side, the initial sweep is in the counterclockwise direction. The robot sweeps for 2.5 seconds before switching the direction of the sweep. If a balloon is detected, the robot begins to track the balloon. A "targeting window" was defined which gives an ideal targeting center location and an acceptable radius. This was implemented to avoid the robot constantly "hunting" for the balloon if there is any error in the Pixy2 camera's detection of the balloon. The speed at this point in the program is reduced so that the robot does not overshoot the target and miss. If the balloon is within the defined firing window, the robot will monitor the balloon 0.1 second before it fires. This delay is to ensure that the robot does not misfire or waste a shot if the Pixy2 camera has uncertainty in the balloon detection. If the balloon exits the targeting window during this time, the robot will continue to track it. Otherwise, the robot will fire at the balloon and rack back the shooter. If there are bullets left in the magazine, the robot will then rack the magazine forward before it returns to the beginning of the targeting algorithm. If all of the bullets have been shot, the robot will instead wait with the shooter in the backwards position until it has been reloaded.

User Interfacing

Since the robot was designed to be autonomous, user interfacing was kept to a minimum. Variables such as the targeting window and speeds could be adjusted through the Arduino IDE by directly interfacing with one of the arduino boards. While interfacing, these boards output the sensor data and current state so that problems can be easily troubleshot. The primary means of user interface beyond this

was the single button attached to the solderboard of the Arduino Mega 2560. This button is used to signal both the start of the program and when a reload has been completed. After initial startup procedures, pressing the button will rack the shooter forward so that it is in a ready to fire position, as well as transitioning the code into the “orientation” section.

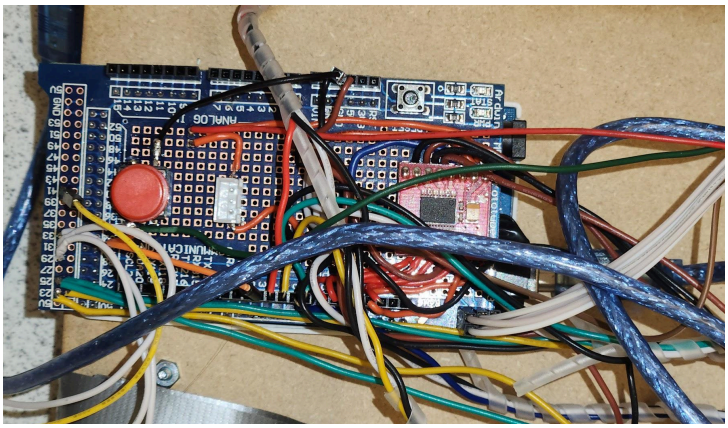


Figure 13: The Arduino Mega 2560 shield with button for user interface.

Once the robot has been positioned and all 12 shots have been fired, another press of the button will signal that a reload has been completed. Prior to this, the robot will wait with the racking mechanism in the backwards position so that the magazine may be removed and reloaded again. When the button is pressed during the reloading state, the robot will rack the shooter into the forward position and return to the targeting algorithm.

System Integration

Once we had each system working separately, we faced several challenges for integration. First, we needed to physically connect all the components together. This proved difficult as we had multiple large components that all needed to fit within the size limitations of the competition. We also needed to wire everything securely. Many of our initial setups included breadboards and jumper wires but with all the systems put together, these became difficult to understand and prone to disconnection. This issue was solved by using a solder board and having all of the connections being securely soldered. Integrating software proved the most difficult task of all. Using three microcontrollers meant we needed to set up communication protocols for them to interact. We encountered issues with data speed, timing, missed data, garbled data, and many other hiccups. Having great difficulty with Serial communication, we switched to using two Arduino units. This meant re-writing the programming base in a new language. However, once this change was made, the integration and communication between the various systems became very smooth. The easiest component to integrate was the shooter, after showing that the shooter worked individually, there was little to no additional work required to integrate with all of the other systems. The only change that needed to be made in relation to shooting was ensuring that the center of the barrel aligned with the targeting area of the Pixy2 camera.

Final Design

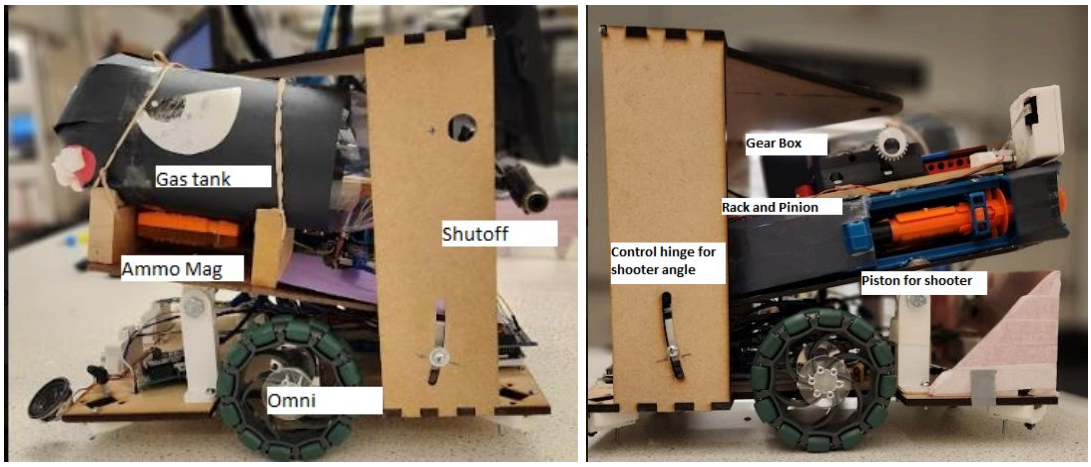


Figure 14: Robot side views. The left view (left) and the right view (right).

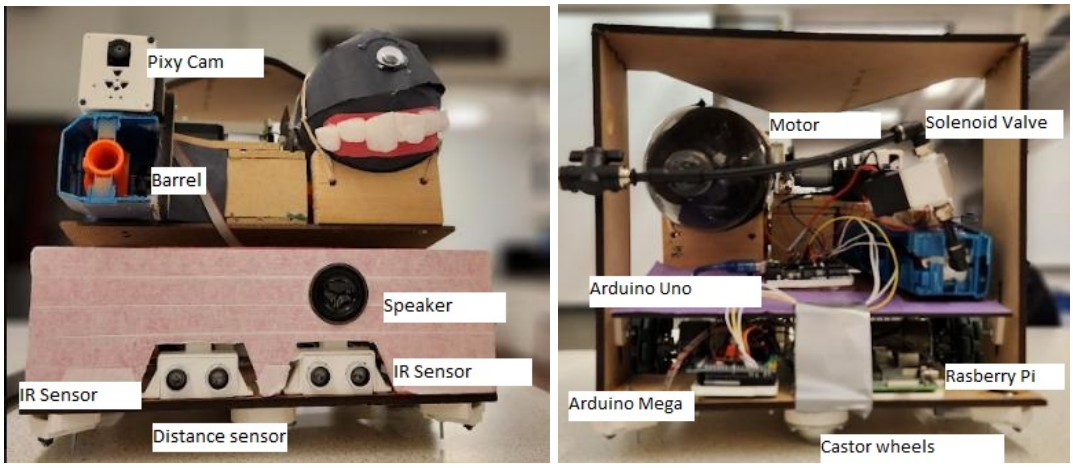


Figure 15: Front (left) and rear (right) views of the robot.

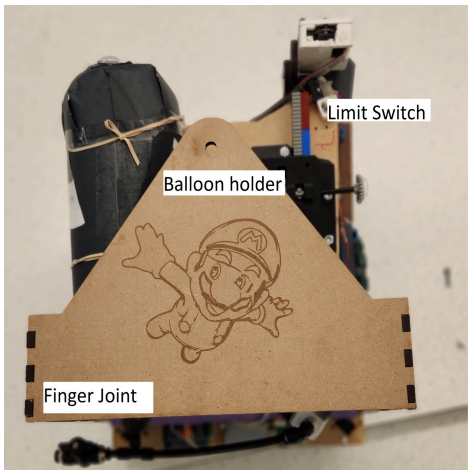


Figure 16: Top view of the robot.

Bill of Materials

Part	Part #	Qty.	Price	Source
Chassis				
MDF Board	-	1	$14	Home Depot
Caster Wheels	Unknown	2	$10	McGuckin Hardware
Drive System				
12V DC Motors	Unknown	2	-	Course resources
Motor Brackets	Unknown	2	-	Course resources
Omni-Wheels	276-2185	2	-	Course resources
Motor Driver	TB6612FNG	1	-	Course resources
Sensors				
PIXY 2.1 Camera	QA010	1	-	Kit
Ultrasonic Sensors	HC-SR04	2	-	Kit
Tape Sensors	TCRT5000	4	$9.99	Amazon
Shooter				
NERF Zombie Strike Longshot CS-12 Blaster	A9546	1	$7.99	Goodwill
DC 6-12V Encoder Motor	Unknown (JIA AI Motor)	1	-	Team Member Owned
Motor Driver	TB6612FNG	1	-	Kit
Limit Switches	SS-5GL-F-3	2	-	Team Member Owned
Transistor	2n222a	1	-	Kit
Solder Board	EL-CP-021	1	-	Team Member Owned
Solenoid Valve	PL-220101	1		$10
2 Liter Soda Bottle	-	1	-	Team Member Owned
LEGOs	-	-	-	ITLL
Fasteners	-	-	-	ITLL

Darts	Little Valentine 100 - Dart Refill Pack		$8.99	Amazon
¼”x¼” Valve	VHK2-06F-06F	1	$11.69	Amazon
¼”x¼” Elbow fittings	6579 56 14WP2	2	$2	Amazon
¼”x¼” Straight Fitting	6505 56 14WP2	2	$2	Amazon
¼”x¼” T Fitting	6304 56 00WP2	1	-	ITLL
Microcontrollers & Communications				
Elegoo Uno	A000066	1	-	Kit
Arduino Mega	EL-CB-003	1	-	Kit
Miscellaneous				
3D Printer Filament	-	1	$25	ITLL
Various Wood Pieces	-	-	-	ITLL
Misc Small Components	-	-	-	Kit/ITLL
11.1V High Discharge Li-Po Battery	9067000399-0	1	-	Course resources
Portable Battery	-	1	-	Team Member Owned
10-24x3/4” Machine Screws	Unknown	10	$5	McGuckin Hardware
4-40x1¼”	Unknown	6	$4	McGuckin Hardware
4-40x3/4”	Unknown	16	$8	McGuckin Hardware
22 Gauge Solid core Wire	Unknown	-	$5	Amazon
Total			$113.67	

Conclusion

Our team was able to create a working autonomous robot that was able to navigate to the reload station and successfully detect, aim at, and hit enemy balloons. We were able to effectively utilize each team members’ strengths to work on subsystems and components that they worked well at. We also allowed for team members to try out tasks that they are unfamiliar with, but were interested in being able

to do. This team dynamic allowed us to have success with the aspects of our robot and allowed for individuals to grow and improve their skills.

Each system had its fair share of difficulties. The chassis system had issues with layout, sizing, and integrating the different parts together. The shooter required a lot of iteration in order to shoot darts, shoot darts with great enough speed, maintain air pressure, and utilize the air pressure effectively. The communication system had issues with sending the right information that each controller could properly read and had issues with two way communication being unacceptably slow. However, through all of these difficulties and issues, the team was able to work together to overcome and solve all of the problems thrown at us which resulted in a successful design that we are proud of. In the end, we had an Arduino Mega controlling movement, sensors, vision, and sending signals to the Arduino Uno. The Arduino Uno controlled the racking and loaded the shooter using information from the limit sensors. While limited communication was used between Arduinos, we were able to have fast and effective communication due to the lack of delay. By focusing initially on having working subsystems the team was able to effectively work through difficulties established later in the project. This included the communication issues between the initially chosen Arduino to Raspberry Pi communication system. This was, however, limited by the bottom-up design approach the team took. This meant that some design problems, such as the location of the magazine release button, were discovered too late to be changed. Some of these problems cascaded into the runoff because seemingly independent systems, such as the shooter angle, could be impacted by the ease of access to cables or the power system.

At the start of the project, the team had some idea of how to create a robot that could autonomously compete in Mario Kart: Battle Royale!®, but through research, iteration, and trial-and-error, we were able to learn a great deal about pneumatics, motion planning, control, and automation all of which came together to create our robot. From all that was learned this semester, we all would be able to utilize what we learned to create a better autonomous robot in the future. Given the chance to reattempt the project, we would be able to build on our newfound knowledge to be able to greatly improve the design. This could include improved systems integration and control algorithms, as they were not defined until the end of the project.

Lessons Learned

One of the major lessons that we learned was to have team-members working on different portions of the project even if they are not immediately necessary. There were several times through the semester where one sub-system was prioritized over another. This meant that potentially not every member of the team had work that could be completed at that time, and that another sub-system may require additional work as well. A more even distribution of labor to each sub-system so that steady progress is made would make some of the necessary project check-in's go more smoothly. Another big lesson that we learned was that communication between different microprocessor systems can be difficult to say the least. Even though we had set up communication protocols earlier in the semester, and seemed to have bi-directional multi-channel communication working, when it came to integration we discovered many problems. We discovered a lack of consistency in the behavior of the data transfer as well as a relative lack of solution options. This was especially true when the problem seemed to change depending on how it was being evaluated, even in simpler code algorithms. Going forward, and in future projects, this integration should have been tested much sooner. Had we discovered potential problems sooner, even if the code was rudimentary, we would have been more able to shift the design and communication

protocols without needing to redo a large amount of work. Another lesson learned was that it is very helpful to save things from previous projects even if they don't immediately have a need. For example, one of our members was able to help supply many of the necessary components for the robot because she already had a large collection of sensors, microchips, and tools. This allowed the team to begin testing and iterating on the design without having to wait for lead times or parts to be delivered.

Future Work

One fundamental change that would be done in future works is improvement on the shooting/aiming system. The robot was able to successfully fire in the general vicinity of the enemy balloon, but it would often be slightly off and miss. There were two causes for this, one was the angle of the shooter was slightly too great such that the dart would fly just over the balloon, and the other was the center of the pixy vision targeting area did not directly match with the center of the NERF Gun barrel. With our current design, more work needed to be done fine-tuning the angle of the second platform and adjusting the pixy targeting area to be more directly in line with the barrel. However, for future work, an even better improvement would be shot tracking and adjustment using PID. After firing a dart, the camera could detect where the dart went in relation to the expected location of where the dart was supposed to go. The code would use this data and adjust the targeting window to account for misalignment of the dart's path. By using feedback, the robot would be able to further improve the accuracy of the shooter. This could extend further to faster target acquisition by including a ramping motor speed based on the distance the target is from the targeting window.

Another future work would be to improve the scrapped two-way communication code. The issue our team ran into was the controllers would not always receive the right signal from the other controller and there were huge communication delays when integrated with the entire robot. Future works could improve upon this code to be more robust with fewer errors and be able to remain functional when errors occur and improving the two-way communication such that the delay between a signal being sent and the controller acting on the received signal is minimal.

A third future work for this project would be improving the motion planning. Currently, the robot moves to the reload station and stays there for the duration of the battle. While this setup works, and it is possible to hit balloons over walls, our robot is stationary and proves to be an easy target to hit. This could be improved by having the robot move around and use cover more effectively. Using cover would allow for more protection of the balloon and make it harder for the enemy robot to track our balloon. This would reduce multiple, repeated shots hitting our balloon in a short period of time.

Appendices

Functional Videos

Link to runoff videos

Circuit Diagrams

Figure 17 : Wheel Drive system (Arduino Mega)

Description: Mega Digital PWM Pins: 13,12,11,8,9,10,6,7,4,5,3; Mega Digital Pins: 24,22,25,23

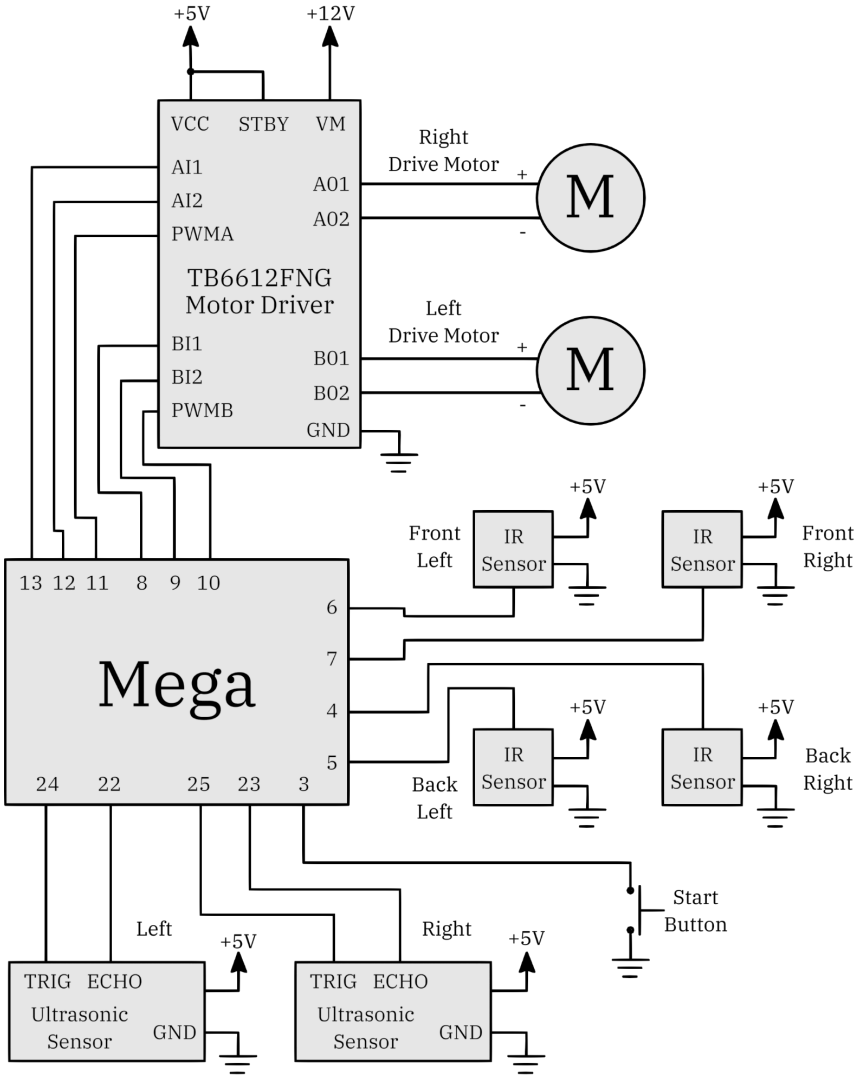


Figure 18 : Vision (Arduino Mega)

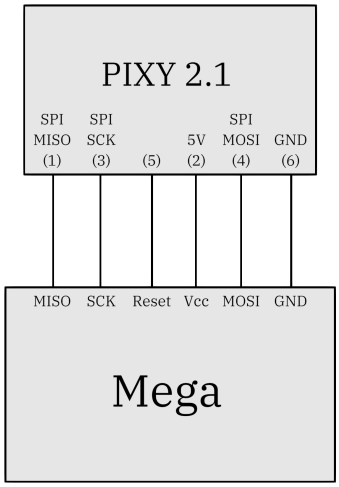


Figure 19 : Shooter (Arduino Uno)

Description: Uno Digital Pins: 8,7; Uno Digital PWM Pins: 9,10,6,5

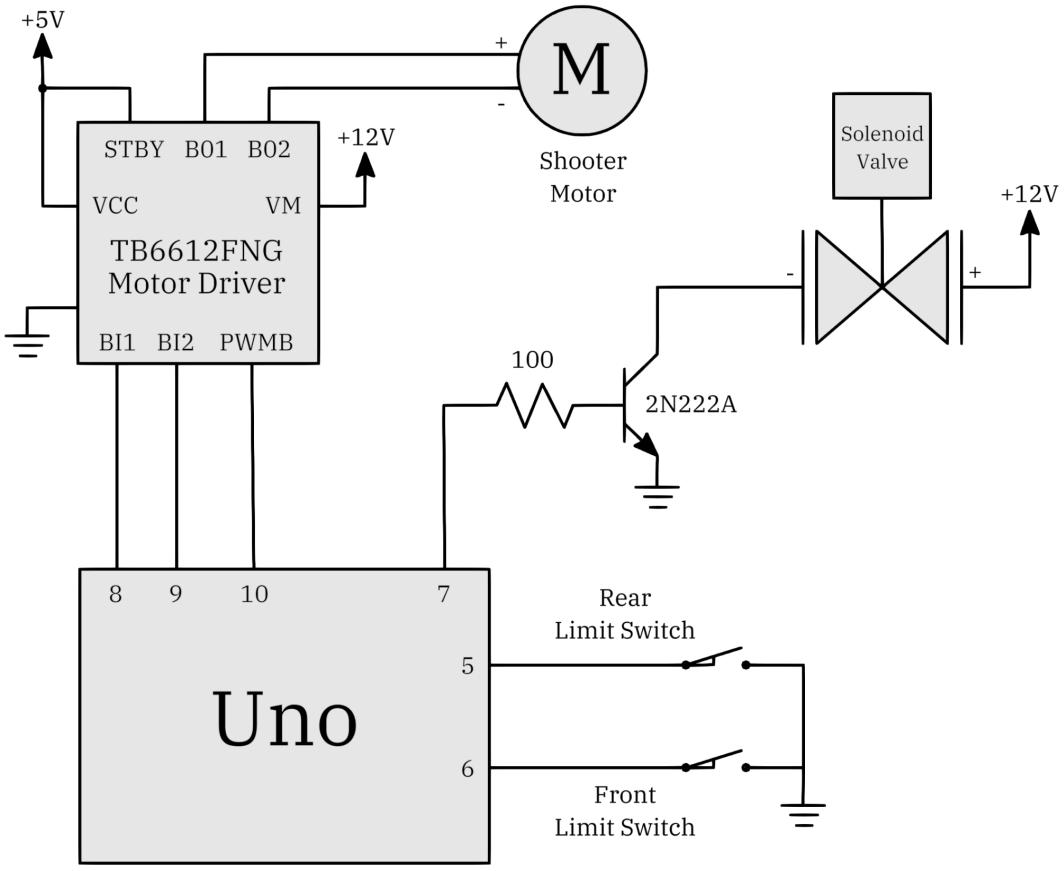


Figure 20 : Audio (Teensy 3.2)

Description: Teensy Digital Pins: 11,12,13,4,23,9,22 ; Teensy RX/ Digital Pin: 2

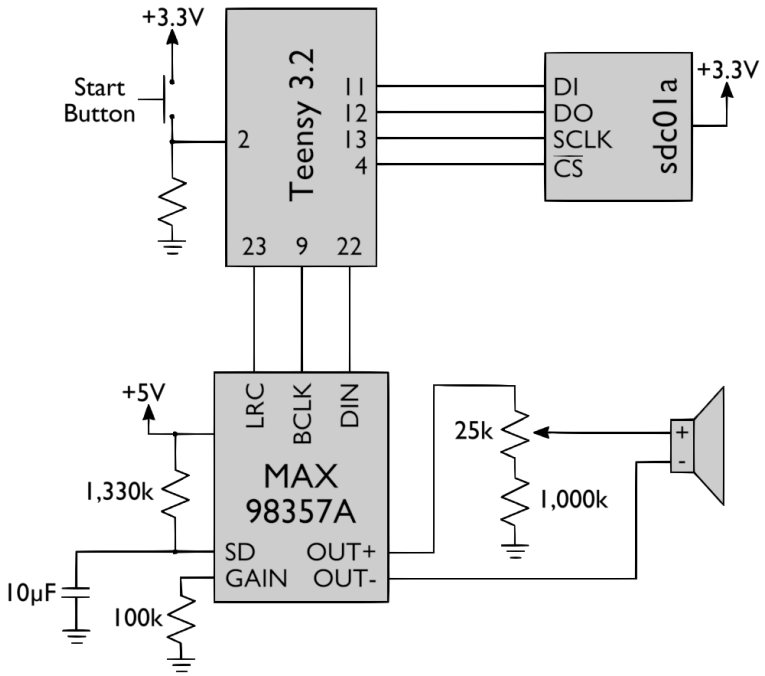


Figure 21 : Microcontroller Connections (Arduino Uno, Arduino Mega, Teensy 3.2)

Description: Uno Digital Pins: 4, 2, 12; Uno Digital PWM Pins: 3; Mega Digital Pins: 29, 31, 33, 41; Mega Digital PWM Pin: 3; Teensy Digital Pin: 6

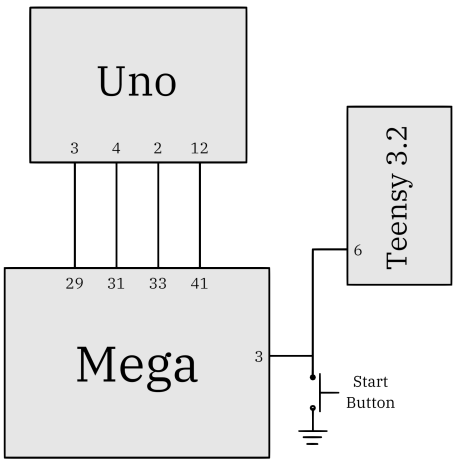
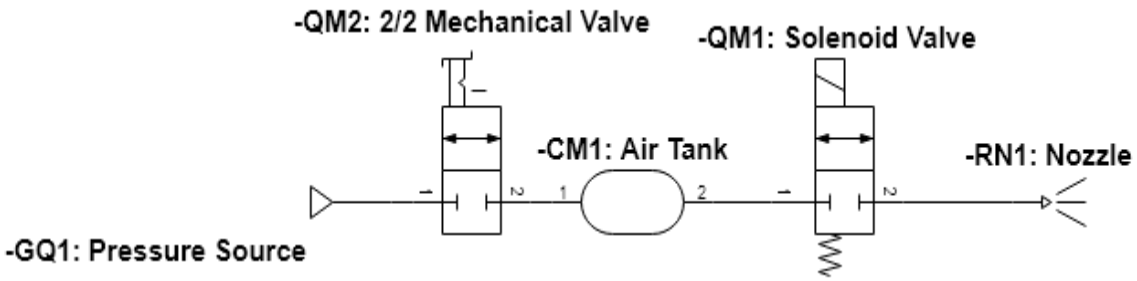


Figure 22: Pneumatic Diagram



#	Part-Number	Description	Quantity
-QM1	Solenoid Valve	Solenoid Valve	1
-CM1	Air Tank	PET Bottle	1
-GQ1	Pressure Source	Pressure Source	1
-QM2	2/2 Mechanical Valve	2/2 Mechanical Valve	1
-RN1	Nozzle	Nerf Inlet	1

Code

Github repository: https://github.com/PdickMontez/MCEN4115/tree/main