Mechanical Engineering
University of Colorado Boulder

MCEN 4115/5115 | Mechatronics & Robotics

Tank Robot

The Phantom Adjudicators

Akash Chinthamanipeta | Loren Peterson | Lowell Glovsky
Allister James Sequeira | Sylvester Francis | Elyjah Toledo

Table of Contents

Table of Contents....2
Section 1: Introduction.... 3
Figure 1: Our robot and silo (right) and the CAD of our robot (left)....3
Section 2: Drivetrain and Chassis....4
Design Process....4
Mechanical Design.... 5
Figure 2: Drive Train Level Chassis....5
Electrical Design....5
Figure 3: Drive Train Level....6
Section 3: Turret and Computer Vision....6
Design Process....6
Mechanical Design.... 7
Figure 4: Servo Motor and Second Level Plate.... 8
Section 4: Launcher....8
Design Process....8
Mechanical Design.... 10
Figure 5: Ball Launching Mechanism....10
Figure 6: Spiral Loading Mechanism.... 10
Electrical Design....10
Figure 7: Solenoid shooter schematic....11
Section 5: Ball Silo....11
Design Process....11
Mechanical Design....11
Figure 8: Completed Silo (left) and CAD mockup of silo structure (right).... 12
Section 6: Results....12
Round 1 - Performance Analysis.... 12
Round 2 - Technical Resilience and Advancement....12
Round 3 - Successful Advancement to Playoffs.... 13
Round 4 - Quarter Finals: Triumph Amidst Technical Challenges.... 13
Section 7: Lessons Learned and Advice.... 14
Appendix: Code.... 15
Special Thanks.... 27

Mechanical Engineering
University of Colorado Boulder

MCEN 4115/5115 *The Phantom Adjudicators* Fall 2023

Section 1: Introduction

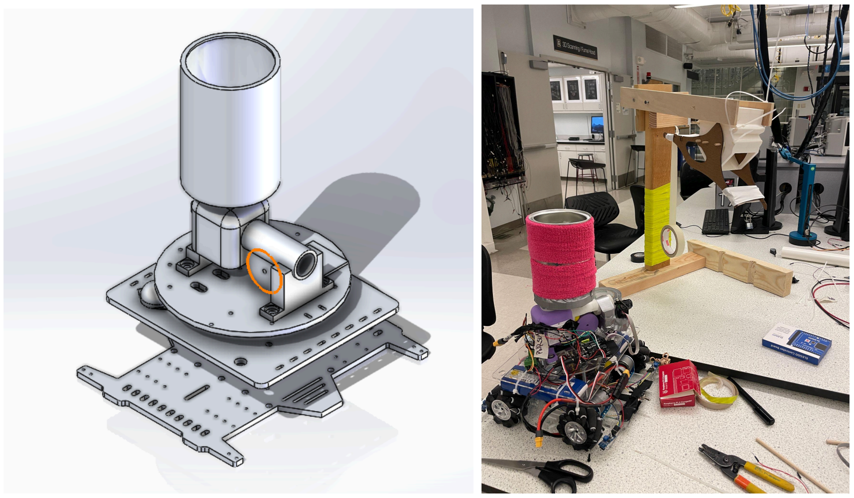


Figure 1: Our robot and silo (right) and the CAD of our robot (left)

The Phantom Adjudicators team developed a fully autonomous robot with a ball launcher and a turret to compete head-to-head with other robots in MCEN 4115/5115. We aimed to create a robot that could easily maneuver the arena while simultaneously targeting and shooting opponents. We achieved our goal after several iterations of our shooting system, diagnosing serial communications failures and a few broken parts. Our final design included a solenoid-actuated single flywheel shooter, a turret rotated by a servo motor, and a Mecanum wheel drive train. We also developed a force-activated ball silo for autonomous reloading. All of these features together aided in our third-place finish.

Section 2: Drivetrain and Chassis

Design Process

Early in the design process, we utilized Mecanum wheels for optimal maneuvering. With the ability to change direction instantaneously, we hoped to have less likelihood of getting stuck in corners. However, initially, we designed the robot with two wheels attached to their respective motors and a caster wheel to guide the back of the drive system. As we developed the system for line following, we realized issues within our drive train system. The drive train was successful in line following but was restrictive and made for a complicated design. If we committed to motors and normal wheels, it restricted our mobility within the competition.

Therefore, we realized that the Mecanum wheels made the robot very responsive and adjusted to boundaries fluidly. There were slight rotations to the robot's movement due to each of the four motors not being pulse-modulated at the same mean voltage. We solved this problem by calibrating each wheel and adding counter spins to the Mecanum wheels each time they reached a white line on the arena. Information was communicated to each wheel using 4 IR sensors.

In the early stages of installing the Mecanum wheels, we assumed that the robot drifted due to unstable connections between the coupler and the wheel itself. To solve this problem, we used hot glue between the coupler and the motor shaft to reduce play. This was very effective; the movement and deflection changed the robot's course meaningfully.

Our chassis was initially designed to maintain structural stability due to the shooting mechanism we originally intended to use. The first platform was large enough to reduce the momentum caused by our planned rack and pinion shooter. The dimensions of the largest portion of the chassis were 9’’x11’’. We also increased the size for mounting, L brackets for the wheels, an Arduino Mega, and a lithium battery. As a result, most of the space used was to navigate and route wires. We would opt for a smaller chassis for future iterations to make it more compact and lighter.

Multiple slots were made within the upper and lower platforms for potential mounting solutions for the batteries and wires. Our next iterative process would route each wire in Solidworks for a more effective and intentional design. In general, our next iteration should consider the placement of each component regardless of criticality for a more aesthetically pleasing design and ergonomics.

Mechanical Design

The base of the chassis housed the complete drivetrain assembly, and the electrical system responsible for its working. The base included mounting plates for the 4 Mecanum wheel drive motors and the four IR sensors responsible for line tracking and autonomous movement in the arena. The chassis had a number of slots, which helped in efficient wiring and mounting of subsystems. The Arduino, being the system's brain, was mounted in the center with the IR sensor circuit board mounted alongside. We mounted the power switch to the chassis base. There were 4 mounting bolts that extended from the base level to the upper level securely mounting the upper level and its assemblies.

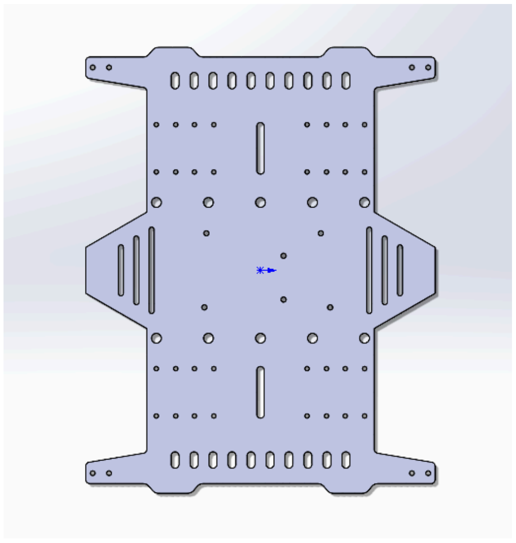


Figure 2: Drivetrain Level Chassis

Electrical Design

The chassis's base housed the drive train's electrical system. An Arduino mega worked as the brain of the drive system. We used an Arduino shield to mount two motor drives for efficient wiring. The motor power came directly from the 12V LiPo battery through the motor drivers. We used IR sensors to detect white line and autonomous movement in the arena. The power supply to the IR sensors came directly through the Arduino 5V and a custom circuit board—the signal wires from the sensors connected to the appropriate pins of the Arduino. The shield was made for a compact design, which routed the motor signal and PWM pins directly within its enclosure. The 12V LiPo was also responsible for powering the Arduino. We added a power switch that directly cutoff and on power to the Arduino and motors when needed.

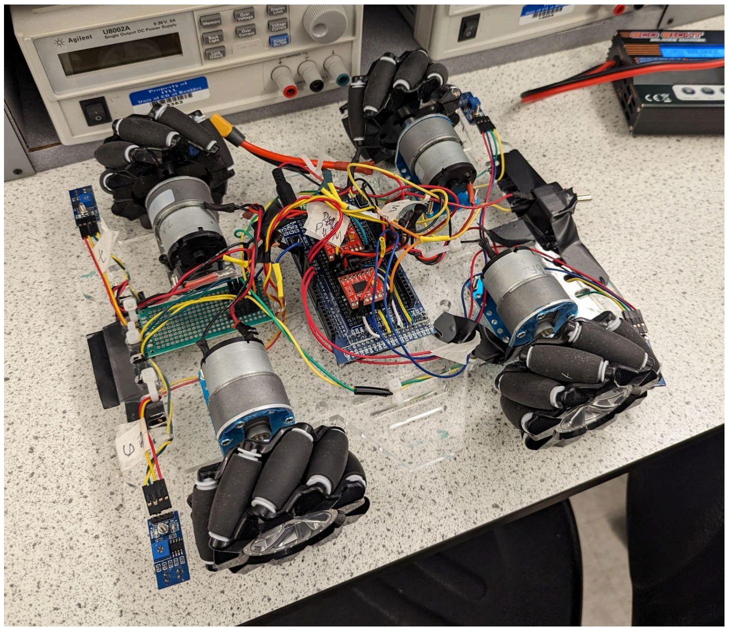


Figure 3: Drive Train Level

Section 3: Turret and Computer Vision

Design Process

Design Choices:

The upper platform of our robot contained a shooting mechanism combining both the solenoid and a flywheel. Servo motors controlled this platform and supplied information to alter the position based on the color signatures read from the pixy. The pixy was mounted beneath the shooting mechanism.

Our initial goal with the turret was to have the shooting mechanism independent of the drive system. This would allow us to have separate microcontrollers for each system, allowing for simpler code.

Many communication issues between the servo motor and Pixy camera took a long time to fix. Changing our design so that the drive train would handle aiming was considered; however, due to our resources and planning our time efficiently, we thought implementing the servo would be the best design for our robot.

Design Challenges:

One issue we had was stuttering in the servo motor caused by not grounding the servo to the microcontroller. The pins used for the Pixy directly used pins 10-13 on the Arduino. However, we used these pins for the servo motor, causing an excess amount of current to be supplied to the pixy and, as a result, bricking 4 pixies in totaldue to signal interruptions and too much power being drawn from the batteries. Once this was fixed, the turret was highly responsive, and the pixy worked without fail.

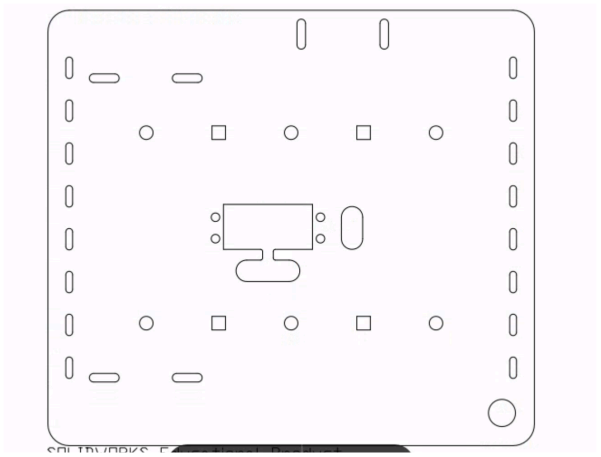
Another issue was the vibrations from the flywheel motor, causing screws attaching the turret plate to the servo motor to become loose. We wanted to replace the nuts with nylon locking nuts, but getting tools positioned to work on this connection was challenging once the robot was assembled. This resulted in our turret wobbling severely after several runs.

Also, the 3D-printed mount for the Pixy camera was far too fragile and would snap easily. Because of this, we needed to tape the camera to the mount, often causing interference with the second level or the ball launcher if not placed perfectly.

Additionally, after bricking several Pixy2 models, we only had the original Pixy available. We rewrote our code for this model and had it function decently; however, the color isolation of this model could have been better. It was far more difficult to calibrate the original Pixy to consistently recognize pink without also recognizing red and even skin tones. Because of this, our robot fired at red cables, shoes, shirts, and even skin in the wrong light.

Finally, we found that our turret often trailed moving targets when firing. This led to a lot of shots barely missing. With more time for calibration and code adjustment, we would have been able to add logic to account for object motion.

Mechanical Design



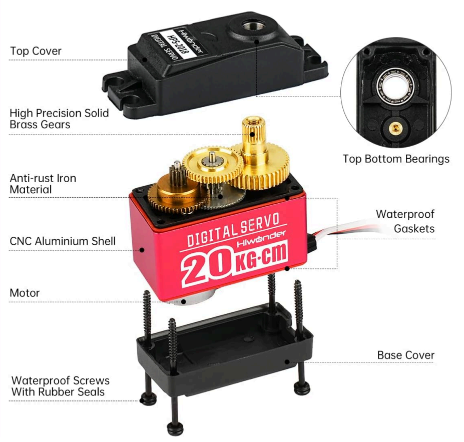


Figure 4: Servo Motor and Second Level Plate

Section 4: Launcher

Design Process

The launcher underwent various iterations and methodologies. We made design progress on pneumatic, spring, and flywheel launchers. Although we were initially resistant to the flywheel method, its simplicity, reliability, and power eventually made it the best option.

We initially considered a pneumatic piston using 2-liter soda bottles for compressed air storage and a solenoid for changing airflow direction; however, the idea was quickly abandoned due to complexity, durability, and space concerns.

The spring-powered piston was the next concept, which got a lot further in the process. The first prototype of this piston used a DC motor with a press-fit 3D-printed pinion gear with a lobed cam. The DC motor was attached to a free platform secured to a rigid platform with springs. The pinion gear on the DC motor interfaced with a geared rack serving as a piston attached to a spring attached to the fixed platform. As the DC motor turned the pinion gear, it pulled the geared rack backward, stretching the spring. The cam's lobe attached to the pinion gear would interact with the fixed platform, lifting the pinion gear out of the rack. This would release the rack, launch it through guides (initially nails in the fixed platform), and smash it into a ping-pong ball. A video of this prototype in action is sitting at 3.8 million views on Instagram.

This design was very promising, and we continued to iterate it. To reduce jerk from the system, we replaced the cam by removing teeth from the pinion gear, allowing the rack to slip underneath. We found that it was far more challenging to re-mesh the rack and pinion without the springs tensioning down the pinion; however, we solved this by making the tooth size larger. Next, we 3D-printed a guide interface for the rack-piston to mount, which integrated a ball holder.

Ultimately, the team decided to abandon the spring piston design. Although it could launch a ball 8 feet (the end-to-end distance of the arena) before falling to launch height, it barely had the power to do so. When we attempted to use stiffer springs to increase the launch power, the inner hub of our 3D-printed pinion gear could not handle the torque and would deform until the motor shaft spun in place. Although a more robust material for the pinion gear might have solved this problem, additional deficiencies in the design made us rethink pursuing it further. The team was concerned about long reload times due to the rack needing to be pulled back by a high torque motor and the high jerk on the rest of the robot caused by the piston stopping. All of these issues led us to quickly pivot.

The first version of our fly-wheel shooting mechanism used two sections of 1-inch inner diameter PVC pipe (a barrel and a feeding tube), a PVC elbow, a motor, and a 3D-printed wheel. The 3D-printed wheel was shaped to cup the radius of the ball, and semi-circular nurlings were included for more ball grip. A slot was cut out of the barrel for the wheel to be placed into, and the wheel was attached to the motor. For our first test, we powered the motor with an adjustable power supply and held the barrel in place with our hands. It did not work well this way, but balls shot out quickly as they were fed through, proving our design. We quickly 3D-printed mounts to secure the barrel for further testing.

The next iteration was similar to the final product. We 3D printed a housing to replace the elbow and join the feeding tube with the barrel. This housing featured a 90-degree interface with a dimple on the bottom to keep the ball from rolling into the shooting tube during robot movement. It also housed a solenoid in the back designed to push the ball into the flywheel as a triggering mechanism. We made minor tweaks to this design for solenoid placement (to avoid jamming) and the pressure the flywheel put on the ball. For the final design, the barrel was superglued to the front support.

Since the shooter was gravity-fed, the shooting-frequency limit was a matter of how quickly the next ball could drop once the ball below it was shot. Determining the shooting frequency limit took some trial and error—our strategy was to maximize shooting speed. Still, if the frequency were too short, the solenoid would hit the bottom of the ball before it had a clear path to the flywheel, jamming the system.

We also carefully considered the distance between the solenoid's piston and the ball chamber. We learned that a solenoid’s force increases as a function of its extension, i.e. the solenoid did not have the force to push the ball if the piston was positioned too close to the ball. To find the sweet spot, we struck a compromise between the piston hitting the ball toward the end of its travel while having enough remaining travel to clear the ball from the chamber. Once the solenoid was dialed in, it was extremely reliable and did not jam.

The flywheel had tremendous potential to shoot the ball with high velocity and consistent placement. We had the benefit of only calibrating one flywheel instead of most groups’ dual-flywheel design requiring the calibration of two. While the ball shot straight, we had to contend with the arc a single flywheel model creates. As our flywheel was on the bottom of the tube, the flywheel imparted a backspin. We thought this would be an advantage, with the backspin causing the ball to rise, counteracting gravity, and ultimately delivering a shot parallel to the ground. In the end, the backspin force was too much, giving the ball a trajectory with a distinct arc that peaked higher than its release point. With more time, we would experiment with factors like the barrel length, the flywheel-tube clearance, the RPM of the motor, and a slight downward angle of the barrel.

During the run-off, we observed that the balls passed through the opposing robot's plane too high, making the ball more prone to whizzing by the sides of the robot's narrower top and missing the meat of the robot below. We turned the shooting motor down some, which sent the ball more toward the body of the robot and increased our hit rate, but the ball was in the drooping section of its arc once reaching the other robot, making longer, cross-court shots more difficult as the ball was likely to hit the ground first.

Mechanical Design

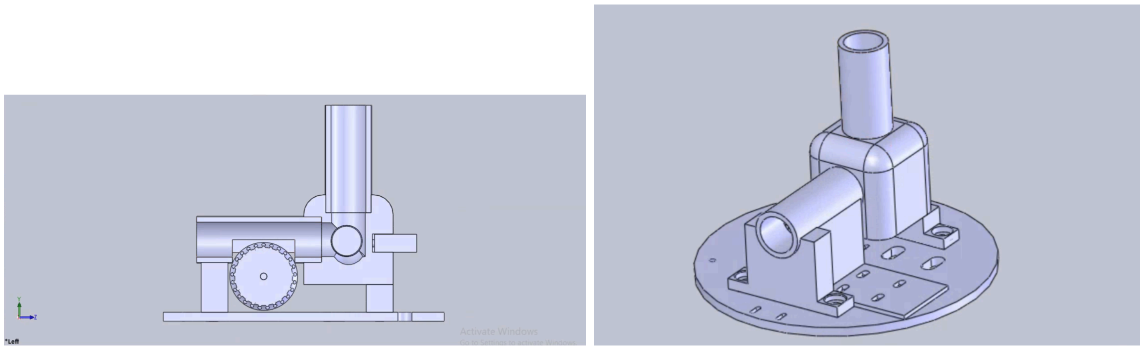


Figure 5: Ball Launching Mechanism

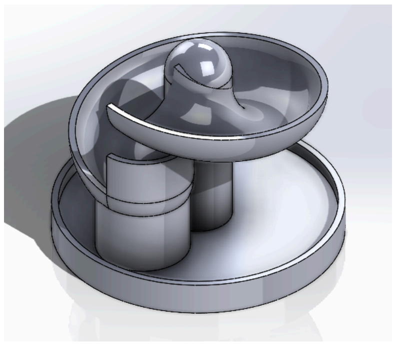


Figure 6: Spiral Loading Mechanism

Electrical Design

The shooter design utilized a TIP31C power transistor to connect the solenoid to ground when receiving a HIGH signal from the Arduino. In the code's first iteration, we used a simple delay function to cycle the solenoid's extension and retraction. However, since the solenoid shared a controller with the turret servo, we quickly discovered that the *delay* function was blocking the servo code and its operation. To solve this problem, we used the *millis* function to measure the elapsed time between solenoid extension, retraction, and vice versa, and send HIGH and LOW based on the time difference that defined the desired shooting period. Last, since the Pixy's targeting of the opposing robot would trigger a stream of shoot commands, we coded a "shooting flag" that would lock out the shoot function when a shooting cycle was in progress.

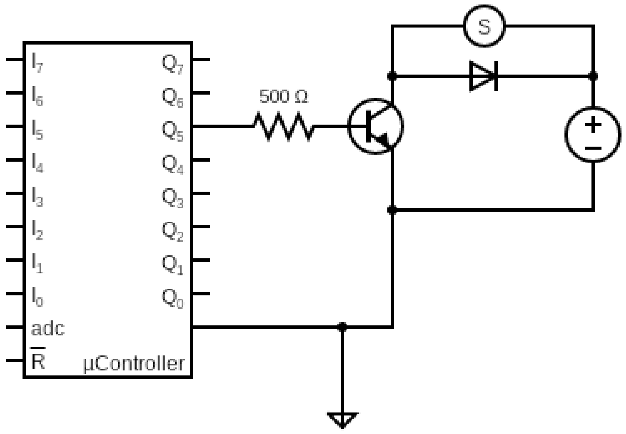


Figure 7: Solenoid shooter schematic

Section 5: Ball Silo

Design Process

The ball silo design is a simple mechanical design where the can of the robot would push against a lever arm which rotated to open the silo lid to unload the silo and load the top can. The lever arm would automatically return to its closed position as soon as it left contact with the lever. The return action was possible with the help of just the right amount of counterweight to keep the silo from opening on its own.

Mechanical Design

The mechanical design includes a wooden frame to keep the silo at the required height. The can was about 15 inches above the ground. The lever arm was designed to touch the upper 2-3 inches of the can such that we would achieve enough contact to push the lever arm. The silo itself is a 3D printed part serving as a container, integrated with the lid plate which gets actuated by the lever arm motion. This container was held in place by a cantilever arm that extended above into the arena.

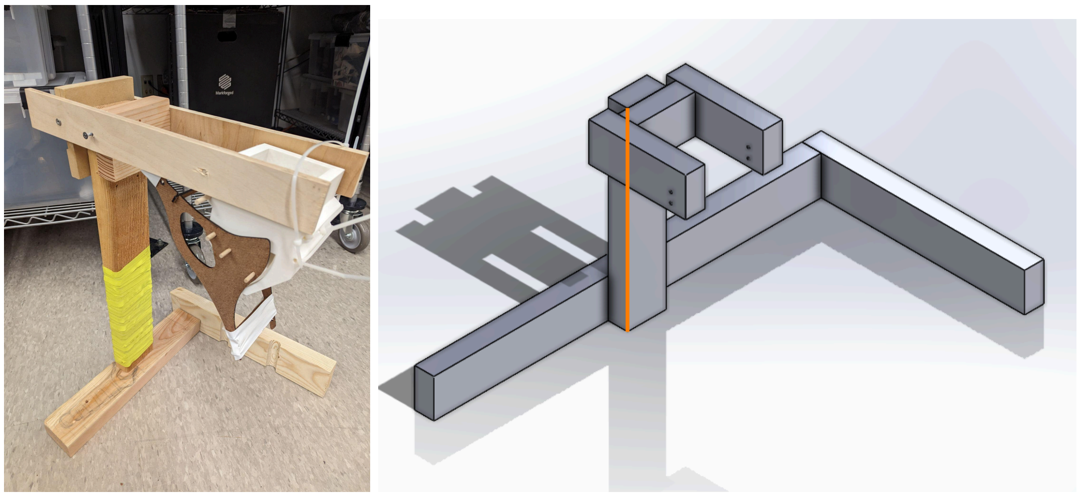


Figure 8: Completed Silo (left) and CAD mockup of silo structure (right)

Section 6: Results

Round 1 - Performance Analysis

In the inaugural round, our robot encountered challenges that impacted its overall performance. A single wire disconnection disrupted the intended path, leading to deviations from the desired trajectory. This setback was exacerbated during the shooting phase when the robot unexpectedly detected a random pink color, initiating premature firing and depleting all allotted balls.

Further wire disconnections caused the robot to deviate from the allocated path, ultimately leading to our disqualification from the race. These challenges underscore the importance of addressing and mitigating technical issues for enhanced reliability in subsequent rounds.

Round 2 - Technical Resilience and Advancement

During Round 2, our team diligently addressed previous wire connection issues, ensuring the robot's robustness for competition. Our opponent's robot encountered difficulties and was unable to move. Seizing this opportunity, our robot demonstrated reliable performance, effectively navigating the course.

Consequently, we were declared the winner of Round 2, allowing us to progress further in the competition. This success emphasizes the significance of technical troubleshooting and preparation in achieving positive outcomes in competitive robotics.

Round 3 - Successful Advancement to Playoffs

In Round 3, our robot showcased improved mobility and accurate shooting capabilities, marking a significant enhancement in performance. As our robot progressed through the competition, it successfully executed precise maneuvers and demonstrated practical shooting skills.

Our robot scored three hits in a competitive exchange, while the opposing robot managed two hits. This compelling performance secured our victory in Round 3, earning us a well-deserved place in the playoffs. This success underscores the positive impact of continuous refinement and technical optimization on competitive robotic achievements.

Round 4 - Quarter Finals: Triumph Amidst Technical Challenges

Our robot displayed commendable performance during the intense quarterfinals in Round 4, securing 6 hits while the opposing robot achieved 5 hits. However, the match's fast-paced nature led to the servo wire's disconnection and the loosening of a few fasteners on our robot.

Undeterred by these technical challenges, our team promptly leveraged the Cylo mechanism to restore functionality and replenish the ammunition. Despite the unforeseen obstacles, we successfully advanced to the semifinals, demonstrating our robot's resilience and our strategy’s adaptability in the face of competition.

Results: Round 5 - Semifinals: Unfortunate Technical Setback and Loss

In the crucial fifth round, our team grappled with persisting issues of loosened fasteners and a disconnected servo wire. Despite time constraints, we entered the arena hoping to overcome these challenges. Unfortunately, our robot faced difficulties, failing to hit any balls due to unresolved technical issues.

Conversely, the opposing robot successfully made a hit, securing its advancement to the next stage of the competition. Regrettably, this outcome marked the end of our journey to the semifinals. This experience underscores the critical importance of timely issue resolution and highlights areas for improvement in future competitions.

Section 7: Lessons Learned and Advice

The most important lesson we learned was the necessity of reading documentation and data sheets before moving forward. Many of our signaling and power issues could have never occurred if we had looked at our hardware and signaling requirements before getting too far into the integration process. Additionally, optimizing how efficient our robot was by reducing the amount of power supplies and creating intelligent power networks would have made the robot more effective and more robust. I also learned that we should have made more circuit diagrams to

It is also vital to communicate regularly between portions of the project. Since we separated into small groups and pairs working on different features of the robot, it was often hard to keep track of what others were doing. If we had regular all-team check-ins, we may have caught integration issues sooner and changed course more smoothly. Especially at the end, when any and all problems become “all hands on deck,” it can be difficult for members to chip in when they see the design for the first time. Additionally, if we also had a timeline beyond the one set for us in class that would have made us move faster and efficiently.

3D printing takes a lot of time and can greatly reduce your ability to iterate. Use other manufacturing methods whenever possible. 3D-printed parts are also far weaker than you might expect. They deform quickly and snap easily along layer lines. We started from scratch on a second shooting system relatively late into the project, but something as simple as using PVC pipe and a hacksaw was far superior to CADing and printing those parts.

We took pride in our robot assembly because every component on the robot was designed by us and only us. You could buy a robot chassis, preprogrammed brushless motors that beep when you turn them on, and several other premade components; however, the more you do yourself, the more pride you will have in the result. We put a lot of time and effort into every robot component. A factory-machined chassis may have looked better, but our acrylic-cut, three-tiered, turreted robot had character and performed as we designed it. There are a lot of advantages to getting prefabricated components and going straight for the obvious solution, including having more time for diagnosing problems and calibration. However, you will have far more ownership and pride in your work by doing everything as a team, taking risks, failing, and succeeding.

For this type of project, scrappiness and tenacity are essential. We made enormous adjustments when things were not working up to our expectations. When we determined that our spring and piston launcher would not be as powerful as our opponents, we threw it away and made an entirely new system in a week. When we had signaling issues between the Pixy and the servo, we worked until the ITLL closed for several days to get it to work. To succeed in this course, you need to push through some problems and adapt to others.

Appendix: Code

```
POWERTRAIN CODE- MOTORS
#include <Pixy.h>
#include <Wire.h>
//PIN VARIABLES
//PIXY VALUES
Pixy pixy;
int signature = -10;
int x = -10;
int y = -10;
int width = -10;
int height = 10;
int angle = -10;
int index = -10;
int age = -10;
int mot_speed = 0;
int x2deg = 90;
bool cosmic_x = 0;
//MOTOR PINOUT
//the motor will be controlled by the motor A pins on the motor driver
const int AIN1_FL = 38;          //control pin 1 on the motor driver for the Front Left motor
const int AIN2_FL = 36;           //control pin 2 on the motor driver for the Front Left motor
const int PWMA_FL = 11;           //speed control pin on the motor driver for the Front Left motor
const int BIN1_FR = 40;          //control pin 1 on the motor driver for the Front Right motor
const int BIN2_FR = 42;           //control pin 2 on the motor driver for the Front Right motor
const int PWMB_FR = 10;           //speed control pin on the motor driver for the Front Right motor
const int AIN1_RL = 28;          //control pin 1 on the motor driver for the Rear Left motor
const int AIN2_RL = 26;           //control pin 2 on the motor driver for the Rear Left motor
const int PWMA_RL = 9;           //speed control pin on the motor driver for the Rear Left motor
const int BIN1_RR = 30;          //control pin 1 on the motor driver for the Rear Right motor
const int BIN2_RR = 32;           //control pin 2 on the motor driver for the Rear Right motor
const int PWMB_RR = 8;           //speed control pin on the motor driver for the Rear Right motor
int switchPin = A7;           //switch to turn the robot on and off
//PARAMETERS
int dura = 5000;
int motspeed = 255;
//
//IR SENSOR PINOUTS
const int IN_A0_FL = A3; // analog input
const int IN_D0_FL = 5; // digital input
const int IN_A0_FR = A4; // analog input
const int IN_D0_FR = 4; // digital input
const int IN_A0_RL = A2; // analog input
const int IN_D0_RL = 7; // digital input
const int IN_A0_RR = A0; // analog input
const int IN_D0_RR = 12; // digital input
bool clear1 = true;
//VARIABLES
```

```
int motorSpeed_FL = 0;      //starting speed for the Front Left motor
int motorSpeed_FR = 0;      //starting speed for the Front Right Motor
int motorSpeed_RL = 0;      //starting speed for the Rear Left motor
int motorSpeed_RR = 0;      //starting speed for the Rear Right Motor
int aaa=1;
void setup() {
 pinMode(switchPin, INPUT_PULLUP);   //set this as a pullup to sense whether the switch is flipped
 //set the motor control pins as outputs
 pinMode(AIN1_FL, OUTPUT);
 pinMode(AIN2_FL, OUTPUT);
 pinMode(PWMA_FL, OUTPUT);
 pinMode(BIN1_FR, OUTPUT);
 pinMode(BIN2_FR, OUTPUT);
 pinMode(PWMB_FR, OUTPUT);
 pinMode(AIN1_RL, OUTPUT);
 pinMode(AIN2_RL, OUTPUT);
 pinMode(PWMA_RL, OUTPUT);
 pinMode(BIN1_RR, OUTPUT);
 pinMode(BIN2_RR, OUTPUT);
 pinMode(PWMB_RR, OUTPUT);

 pinMode(IN_A0_FR, INPUT);
 pinMode (IN_D0_FR, INPUT);
 pinMode(IN_A0_FL, INPUT);
 pinMode (IN_D0_FL, INPUT);
 pinMode (IN_A0_RR, INPUT);
 pinMode (IN_D0_RR, INPUT);
 pinMode (IN_A0_RL, INPUT);
 pinMode (IN_D0_RL, INPUT);
  Serial.begin(9600);                  //begin serial communication with the computer
}
int value_A0_FR;
int value_A0_FL;
int value_A0_RR;
int value_A0_RL;
bool check = true;
int ajsetrt = 0;

void loop()
{
 value_A0_FR = analogRead(IN_A0_FR);
 value_A0_FL = analogRead(IN_A0_FL);
 value_A0_RR = analogRead(IN_A0_RR);
 value_A0_RL = analogRead(IN_A0_RL);
 //forwards();
 if(ajsetrt == 0) //ajestrt is a booling that turns the motors forward or backward depending if the IR sensors are
activated or not.
 {
  forwards(200);
 }
 else if(ajsetrt == 1)
 {
  backwards(200);
 }
 bool frontall = stuckFRONT_ALL();
 bool backall = stuckREAR_ALL();
 bool leftall = stuckLEFT_ALL();
 bool rightall = stuckRIGHT_ALL();
 bool FL_sensor = stuckFL();
 bool FR_sensor = stuckFR();
 bool RL_sensor = stuckRL();
 bool RR_sensor = stuckRR();
 if(frontall == true)
 {
  backall = false;
  ajsetrt = 1;
 }
 if(backall == true)
 {
  frontall = false;
  ajsetrt = 0;
 }
 if(cosmic_x = 1)
 {
    RELOADED();
 }
}
//************************************************************************************************
***************//BOUNDARY
TESTING*************************************************************************************************
//*************************************************//Two sensors detected at a
time*******************************************
bool readIRSensor()
{
 }
bool correctedF = false;
bool correctedR = false;
bool correctedL = false;
bool correctedS = false;
bool correctedFL = false;
bool correctedFR = false;
bool correctedRL = false;
bool correctedRR = false;
bool stuckFRONT_ALL()
{
 correctedF = false;
 while (value_A0_FR < 100 && value_A0_FL < 100 && value_A0_RL > 100 && value_A0_RR > 100)
 {
   correctedF = true;
   off();
   backwards(255); //duration?
   value_A0_FR = analogRead(IN_A0_FR);
```

```
    value_A0_FL = analogRead(IN_A0_FL);
    value_A0_RR = analogRead(IN_A0_RR);
    value_A0_RL = analogRead(IN_A0_RL);
  }
  return correctedF;
bool stuckREAR_ALL()
 correctedR = false;
  while (value_A0_RL < 100 && value_A0_RR < 100 && value_A0_FL > 100 && value_A0_FR > 100)
  {
    correctedR = true;
    off();
    forwards(255); //duration?
    value_A0_FR = analogRead(IN_A0_FR);
    value_A0_FL = analogRead(IN_A0_FL);
    value_A0_RR = analogRead(IN_A0_RR);
    value_A0_RL = analogRead(IN_A0_RL);
  }
  return correctedR;
}
bool stuckLEFT_ALL()
{
  correctedL = false;
  while (value_A0_FL < 100 && value_A0_RL < 100 && value_A0_FR > 100 && value_A0_RR > 100)
  {
    off();
    RT_Trans(255); //duration?
    value_A0_FR = analogRead(IN_A0_FR);
    value_A0_FL = analogRead(IN_A0_FL);
    value_A0_RR = analogRead(IN_A0_RR);
    value_A0_RL = analogRead(IN_A0_RL);
    correctedL = true;
  }
  return correctedL;
}
bool stuckRIGHT_ALL()
{
 correctedS = false;
  while (value_A0_FR < 100 && value_A0_RR < 100 && value_A0_FL > 100 && value_A0_RL > 100)
  {
    off();
    LT_Trans(255); //duration?
    value_A0_FR = analogRead(IN_A0_FR);
    value_A0_FL = analogRead(IN_A0_FL);
    value_A0_RR = analogRead(IN_A0_RR);
    value_A0_RL = analogRead(IN_A0_RL);
    correctedS = true;
  }
  return correctedS;
}
//*********************************************************************//Single sensor
detected*******************************************
```

```
bool stuckFL()
{
 correctedFL = false;
 while (value_A0_FL < 100 && value_A0_RL > 100 && value_A0_FR > 100 && value_A0_RR > 100)
 {
  correctedFL = true;
  off();
  Rotate_CW(200); //duration?
  delay(50);
  RT_Trans(200);
  delay(100);
  value_A0_FR = analogRead(IN_A0_FR);
  value_A0_FL = analogRead(IN_A0_FL);
  value_A0_RR = analogRead(IN_A0_RR);
  value_A0_RL = analogRead(IN_A0_RL);
 }
 return correctedFL;
bool stuckFR()
{
 correctedFR = false;
 while (value_A0_FR < 100 && value_A0_RR > 100 && value_A0_FL > 100 && value_A0_RL > 100)
 {
  correctedFL = true;
  off();

  Rotate_CCW(200); //duration?
  delay(50);
  LT_Trans(200);
  delay(100);
   value_A0_FR = analogRead(IN_A0_FR);
  value_A0_FL = analogRead(IN_A0_FL);
  value_A0_RR = analogRead(IN_A0_RR);
  value_A0_RL = analogRead(IN_A0_RL);
 }
 return correctedFR;
bool stuckRL()
 correctedRL = false;
 while (value_A0_RL < 100 && value_A0_FL > 100 && value_A0_RR > 100 && value_A0_FR > 100)
 {
  correctedRL = true;
  off();

  Rotate_CCW(200); //duration?
  delay(50);
  RT_Trans(200);
  delay(100);

  value_A0_FR = analogRead(IN_A0_FR);
  value_A0_FL = analogRead(IN_A0_FL);
  value_A0_RR = analogRead(IN_A0_RR);
  value_A0_RL = analogRead(IN_A0_RL);
 }
 return correctedRL;
}

bool stuckRR()
{
 correctedFL = false;
 while (value_A0_RR < 100 && value_A0_FR > 100 && value_A0_FL > 100 && value_A0_RL > 100)
 {
  correctedFL = true;
  off();

  Rotate_CW(200); //duration?
  delay(50);
  LT_Trans(200);
  delay(100);

  value_A0_FR = analogRead(IN_A0_FR);
  value_A0_FL = analogRead(IN_A0_FL);
  value_A0_RR = analogRead(IN_A0_RR);
  value_A0_RL = analogRead(IN_A0_RL);
 }
 return correctedRR;
}
int off()
{
 spinMotor_FR(0);
 spinMotor_FL(0);
 spinMotor_RR(0);
 spinMotor_RL(0);
}
int forwards(int v)
{
  motorSpeed_FR = v;
  motorSpeed_FL = motorSpeed_FR;
  motorSpeed_RL = motorSpeed_FR;
  motorSpeed_RR = motorSpeed_FR;
  spinMotor_FR(motorSpeed_FR);
  spinMotor_FL(motorSpeed_FL);
  spinMotor_RR(motorSpeed_RR);
  spinMotor_RL(motorSpeed_RL);
int backwards(int v)
  motorSpeed_FR = -v;
  motorSpeed_FL = motorSpeed_FR;
  motorSpeed_RL = motorSpeed_FR;
  motorSpeed_RR = motorSpeed_FR;
   spinMotor_FR(motorSpeed_FR);
   spinMotor_FL(motorSpeed_FL);
   spinMotor_RR(motorSpeed_RR);
   spinMotor_RL(motorSpeed_RL);
```

```
int LT_Trans(int v)
 //left turn translational
  motorSpeed_FR = v;
  motorSpeed_FL = -v;
  motorSpeed_RL = v;
  motorSpeed_RR = -v;
    spinMotor_FR(motorSpeed_FR);
    spinMotor_FL(motorSpeed_FL);
    spinMotor_RR(motorSpeed_RR);
    spinMotor_RL(motorSpeed_RL);
}
int RT_Trans(int v)
{
//right turn translational
  motorSpeed_FR = -v;
  motorSpeed_FL = v;
  motorSpeed_RL = -v;
  motorSpeed_RR = v;
    spinMotor_FR(motorSpeed_FR);
    spinMotor_FL(motorSpeed_FL);
    spinMotor_RR(motorSpeed_RR);
    spinMotor_RL(motorSpeed_RL);
}
int Rotate_CCW(int v)
{
  //rotating ccw(+)
  motorSpeed_FR = v;
  motorSpeed_FL = -v;
  motorSpeed_RL = -v;
  motorSpeed_RR = v;
    spinMotor_FR(motorSpeed_FR);
    spinMotor_FL(motorSpeed_FL);
    spinMotor_RR(motorSpeed_RR);
    spinMotor_RL(motorSpeed_RL);
int Rotate_CW(int v)
    spinMotor_FR(motorSpeed_FR);
    spinMotor_FL(motorSpeed_FL);
    spinMotor_RR(motorSpeed_RR);
    spinMotor_RL(motorSpeed_RL);
}
int Front_Left_Diag(int v)
{
//front left diag.
  motorSpeed_FR = v;
  motorSpeed_FL = 0;
  motorSpeed_RL = v;
  motorSpeed_RR = 0;
    spinMotor_FR(motorSpeed_FR);
    spinMotor_FL(motorSpeed_FL);
    spinMotor_RR(motorSpeed_RR);
    spinMotor_RL(motorSpeed_RL);
```

```
}
int Back_Right_Diag(int v)
{
//back right diag.
  motorSpeed_FR = -v;
  motorSpeed_FL = 0;
  motorSpeed_RL = -v;
  motorSpeed_RR = 0;
   spinMotor_FR(motorSpeed_FR);
   spinMotor_FL(motorSpeed_FL);
   spinMotor_RR(motorSpeed_RR);
   spinMotor_RL(motorSpeed_RL);
}
int Front_Right_Diag(int v)
{
 //front right diag.
  motorSpeed_FR = 0;
  motorSpeed_FL = v;
  motorSpeed_RL = 0;
  motorSpeed_RR = v;
   spinMotor_FR(motorSpeed_FR);
   spinMotor_FL(motorSpeed_FL);
   spinMotor_RR(motorSpeed_RR);
   spinMotor_RL(motorSpeed_RL);
}
int Back_Left_Diag(int v)
{
//back left diag.
  motorSpeed_FR = 0;
  motorSpeed_FL = -v;
  motorSpeed_RL = 0;
  motorSpeed_RR = -v;
   spinMotor_FR(motorSpeed_FR);
   spinMotor_FL(motorSpeed_FL);
   spinMotor_RR(motorSpeed_RR);
   spinMotor_RL(motorSpeed_RL);
}
void spinMotor_FL(int motorSpeed_FL)                //function for driving the right motor
{
 if (motorSpeed_FL > 0)                            //if the motor should drive forward (positive speed)
 {
  digitalWrite(AIN1_FL, HIGH);                     //set pin 1 to high
  digitalWrite(AIN2_FL, LOW);                      //set pin 2 to low
 }
 else if (motorSpeed_FL < 0)                       //if the motor should drive backward (negative speed)
 {
  digitalWrite(AIN1_FL, LOW);                      //set pin 1 to low
  digitalWrite(AIN2_FL, HIGH);                     //set pin 2 to high
 }
 else                                              //if the motor should stop
 {
      digitalWrite(AIN1_FL, LOW);                    //set pin 1 to low
      digitalWrite(AIN2_FL, LOW);                    //set pin 2 to low
    }
    analogWrite(PWMA_FL, abs(motorSpeed_FL));             //now that the motor direction is set, drive it at the entered
speed
  }
  void spinMotor_FR(int motorSpeed_FR)                //function for driving the right motor
  {
    if (motorSpeed_FR > 0)                          //if the motor should drive forward (positive speed)
    {
      digitalWrite(BIN1_FR, HIGH);                   //set pin 1 to high
      digitalWrite(BIN2_FR, LOW);                    //set pin 2 to low
    }
    else if (motorSpeed_FR < 0)                       //if the motor should drive backward (negative speed)
    {
      digitalWrite(BIN1_FR, LOW);                    //set pin 1 to low
      digitalWrite(BIN2_FR, HIGH);                   //set pin 2 to high
    }
    else                                    //if the motor should stop
    {
      digitalWrite(BIN1_FR, LOW);                    //set pin 1 to low
      digitalWrite(BIN2_FR, LOW);                    //set pin 2 to low
    }
    analogWrite(PWMB_FR, abs(motorSpeed_FR));             //now that the motor direction is set, drive it at the entered
speed
  }
  void spinMotor_RL(int motorSpeed_RL)                //function for driving the right motor
  {
    if (motorSpeed_RL > 0)                          //if the motor should drive forward (positive speed)
    {
      digitalWrite(AIN1_RL, HIGH);                   //set pin 1 to high
      digitalWrite(AIN2_RL, LOW);                    //set pin 2 to low
    }
    else if (motorSpeed_RL < 0)                       //if the motor should drive backward (negative speed)
    {
      digitalWrite(AIN1_RL, LOW);                    //set pin 1 to low
      digitalWrite(AIN2_RL, HIGH);                   //set pin 2 to high
    }
    else                                    //if the motor should stop
    {
      digitalWrite(AIN1_RL, LOW);                    //set pin 1 to low
      digitalWrite(AIN2_RL, LOW);                    //set pin 2 to low
    }
    analogWrite(PWMA_RL, abs(motorSpeed_RL));             //now that the motor direction is set, drive it at the entered
speed
  }
  void spinMotor_RR(int motorSpeed_RR)                //function for driving the right motor
  {
    if (motorSpeed_RR > 0)                          //if the motor should drive forward (positive speed)
    {
      digitalWrite(BIN1_RR, HIGH);                   //set pin 1 to high
      digitalWrite(BIN2_RR, LOW);                    //set pin 2 to low
    }
    else if (motorSpeed_RR < 0)                    //if the motor should drive backward (negative speed)
    {
      digitalWrite(BIN1_RR, LOW);                    //set pin 1 to low
      digitalWrite(BIN2_RR, HIGH);                   //set pin 2 to high
    }
    else                                   //if the motor should stop
    {
      digitalWrite(BIN1_RR, LOW);                    //set pin 1 to low
      digitalWrite(BIN2_RR, LOW);                    //set pin 2 to low
    }
    analogWrite(PWMB_RR, abs(motorSpeed_RR));            //now that the motor direction is set, drive it at the entered
speed
  }
  int RELOADED()
    blocks = pixy.getBlocks();
    if (blocks)
      Serial.println("Pixy on!");
      off();
      x = pixy.blocks[0].x;              //The x location of the center of the detected object (0 to 316)
      width =  pixy.blocks[0].width;        //The width of the detected object (1 to 316)
      bool FL_sensor = stuckFL();
      bool FR_sensor = stuckFR();
      bool RL_sensor = stuckRL();
      bool RR_sensor = stuckRR();
      if (width < 130 && width > 5)
      {
        Serial.print("Width: ");
        Serial.print(width);

        Serial.print(" X pos: ");
        Serial.print(x);

        if (x > 153 && x < 163)
        {
          backwards(255);
          Serial.println("  yo! ");
        }
        else if (x < 153)
        {
          Back_Right_Diag(255);
        }
        else if (x > 163)
        {
          Back_Left_Diag(255);
        }
      }
      else if (width > 130)
      {
        off();
    delay(4000);
```

```
PIXY ONE TURRET CODE -SOLENOID, MOTOR, AND PIXY ONE
#include <Servo.h>
#include <Pixy.h>
//#include <Pixy2CCC.h>
#include <SPI.h>

Pixy pixy;
int signature = -10;
int SLND = 5;
int x = 0;
int y = -10;
int width = -10;
int height = 10;
int angle = -10;
int index = -10;
int age = -10;
int mot_speed = 0;
int x2deg = 90;
int servo_pin = 6;

unsigned long startMillis;
unsigned long currentMillis;
unsigned long endShoot;
const unsigned long period = 250;
int shooting = 0;


Servo myservo;  // create servo object to control a servo

const int AIN1 = 8;           //control pin 1 on the motor driver for the motor
const int AIN2 = 9;           //control pin 2 on the motor driver for the motor
const int PWMA = 3;           //speed control pin on the motor driver for the motor
int motorSpeed = 255;       //starting speed for the motor
int pos = 0;     // variable to store the servo position
void setup() {
 SPI.begin();
 Serial.begin(115200);
 pinMode(SLND, OUTPUT);
 myservo.write(90);
 myservo.attach(servo_pin);  // attaches the servo on pin 9 to the servo object
 pixy.setBrightness(100);
 pixy.init();
}
int blocks;
int i = 0;
void loop()
{
 blocks = pixy.getBlocks();
 x = pixy.blocks[0].x;
 Serial.print("OBJ POS: ");
 Serial.println(x);
 if (blocks)
 {
  spinMotor(70);
  x = pixy.blocks[0].x;       //The x location of the center of the detected object (0 to 316)
  if (x < 148 && x > 0)
  {
   if (x2deg >= 180)
   {
    x2deg = 90;
     }
   else
   {
    x2deg = x2deg + 1;
    myservo.write(x2deg);
       if ( ( x >= 140 && x <= 180)) {
        shoot();
    }
   }
  }
  else if (x > 168 && x < 316)
  {
   if (x2deg <= 0)
   {
    x2deg = 90;
      }
   else
   {
    x2deg = x2deg - 1;
    myservo.write(x2deg);
    if ( ( x >= 140 && x <= 180)) {
        shoot();
    }
   }
  }
 }
 Serial.print(shooting);
  if (shooting) {
  currentMillis = millis();
  if (currentMillis - startMillis >= period) {
   digitalWrite(SLND, LOW);
   shooting = 0;
   endShoot = millis();
  }
 }
}
void spinMotor(int motorSpeed)                    //function for driving the right motor
{
 analogWrite(PWMA, abs(motorSpeed));              //now that the motor direction is set, drive it at the entered speed
}
void shoot() {
 currentMillis = millis();
 if (!shooting && (currentMillis - endShoot >= period)) {
  Serial.print("shooting");
  shooting = 1;
  digitalWrite(SLND, HIGH);
  startMillis = millis();
 }
}
```

Special Thanks

Thank you to the wonderful TAs for all of your help throughout the semester. It was vital to have your support through struggles and successes.

Also thank you to the ITLL and the Idea Forge for your resources and advice.