ME 72A Final Write-Up

Team Penguinators

James Muren, Jade Millan, Jabri Garcia, Jaylen Shawcross, Sara Razavi

Part I: Introduction

The 2024-2025 competition in the ME72 Engineering Design Laboratory is Bot Hockey, where two teams of three remote-controlled robots compete to score goals using a street hockey puck.

We have adopted a strategic approach centered around specialized roles for our robots: Enforcer, Goalie, and Striker.

- The Enforcer is designed for bot-to-bot interaction, focusing on moving opposite robots for offense and defense by using its heavy magnetic force and high torque to dominate such interactions.
- The Goalie is designed with a 12-inch height to block puck shots, with a trapping intake mechanism and heavy magnetism for defense.
- The Striker is designed with a motorized intake and high-speed shooter for rapid and precise scoring and a smaller design to ensure higher speed and enhanced mobility.

We have divided the labor for creating these robots into three specialized teams: Mechanical Design, Manufacturing, and Electrical/Software. Each of these teams is led by: James Muren, Jaylen Shawcross, and Jade Millan, respectively. The rest of the team contributes to their assigned tasks in each of these teams and provides additional help.

Part II: Design Overview

This term we primarily focused on the design of our Enforcer Bot. This robot prioritizes bot-to-bot interactions with an emphasis on effective weight, durability, and pushing power.

A mix of BotHockey robots and Sumo Robot strategies inspired the overall design.

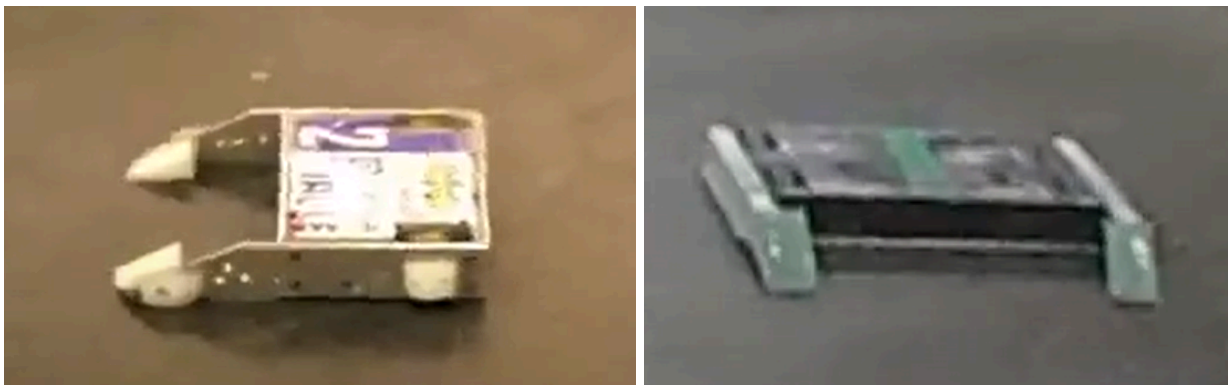


Fig 1. Two robot designs that performed quite well in a Youtube video BotHockey match Robogames Bot Hockey 2013

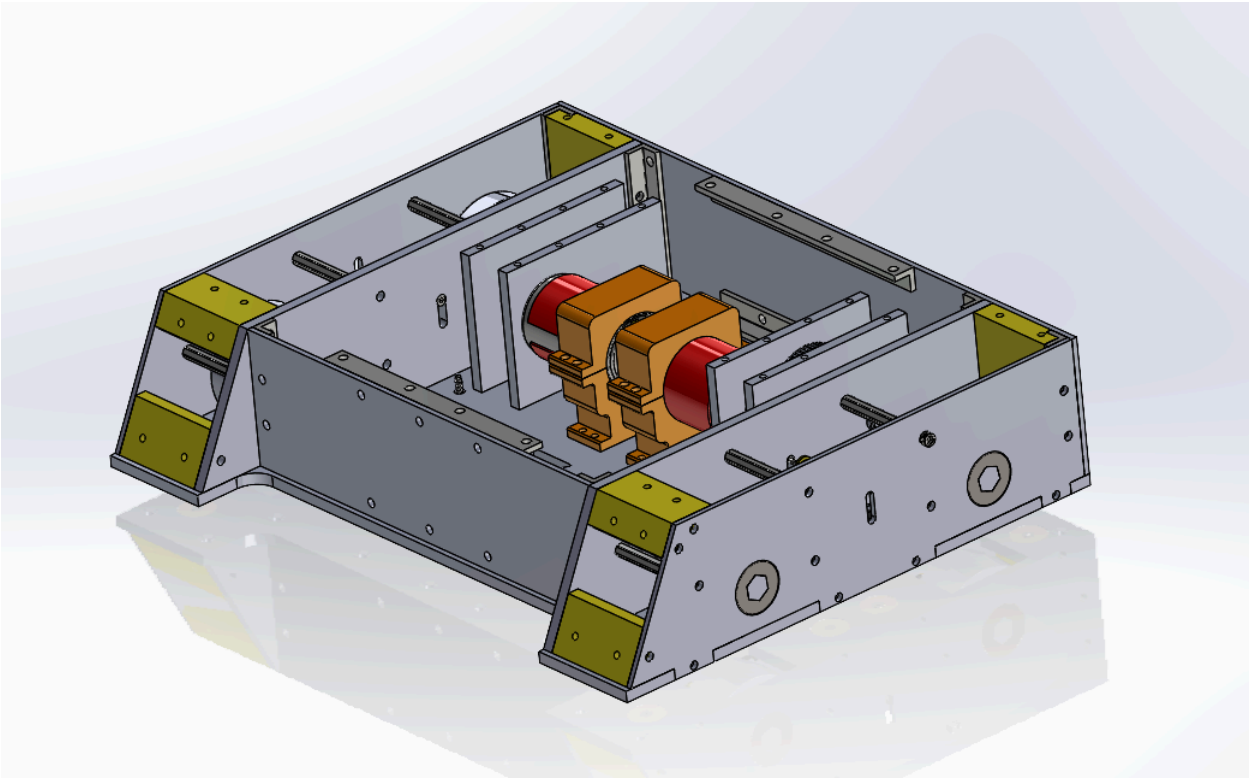


Fig 2. Our final Enforcer Bot design inspired by the BotHockey metagame. The design features a tapered front to get under other robots, and a small corral to control the puck.

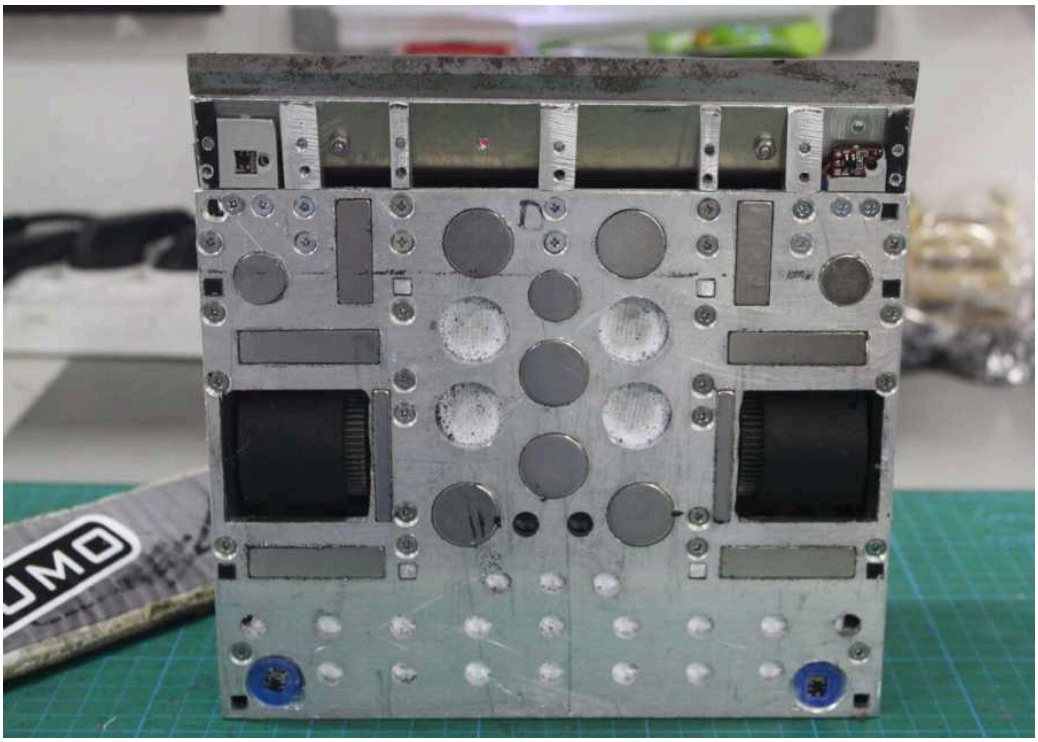


Fig 3. The underside of a sumo robot with neodymium magnets to increase effective weight and traction (Jsumo.com)

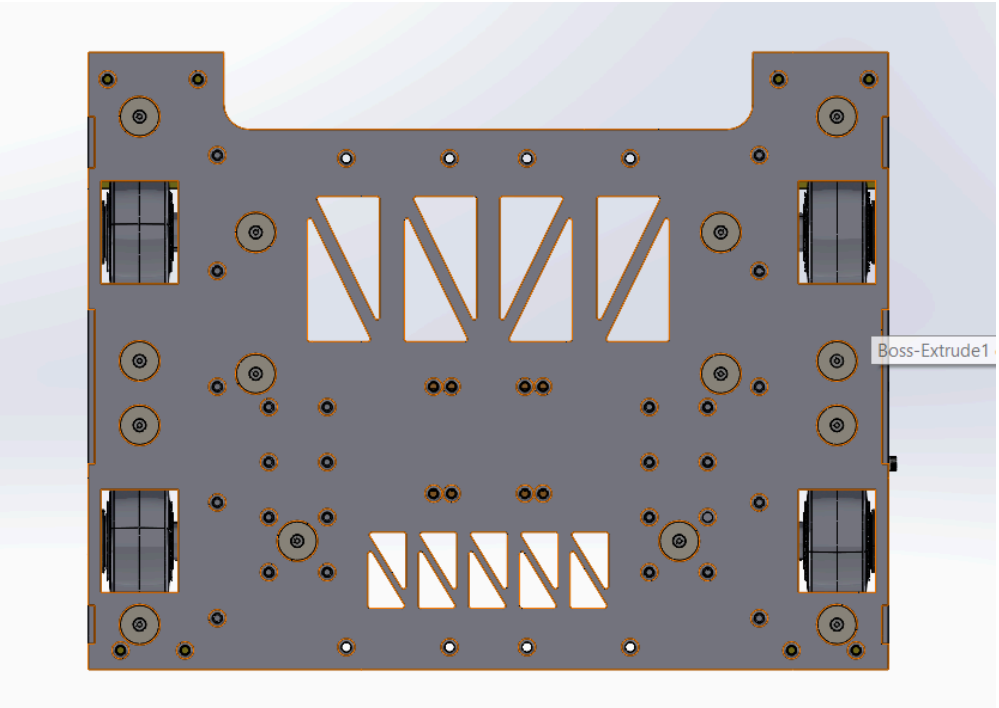


Fig 4. The underside of our robot with 14 neodymium magnets to produce ~80lbs of effective downforce

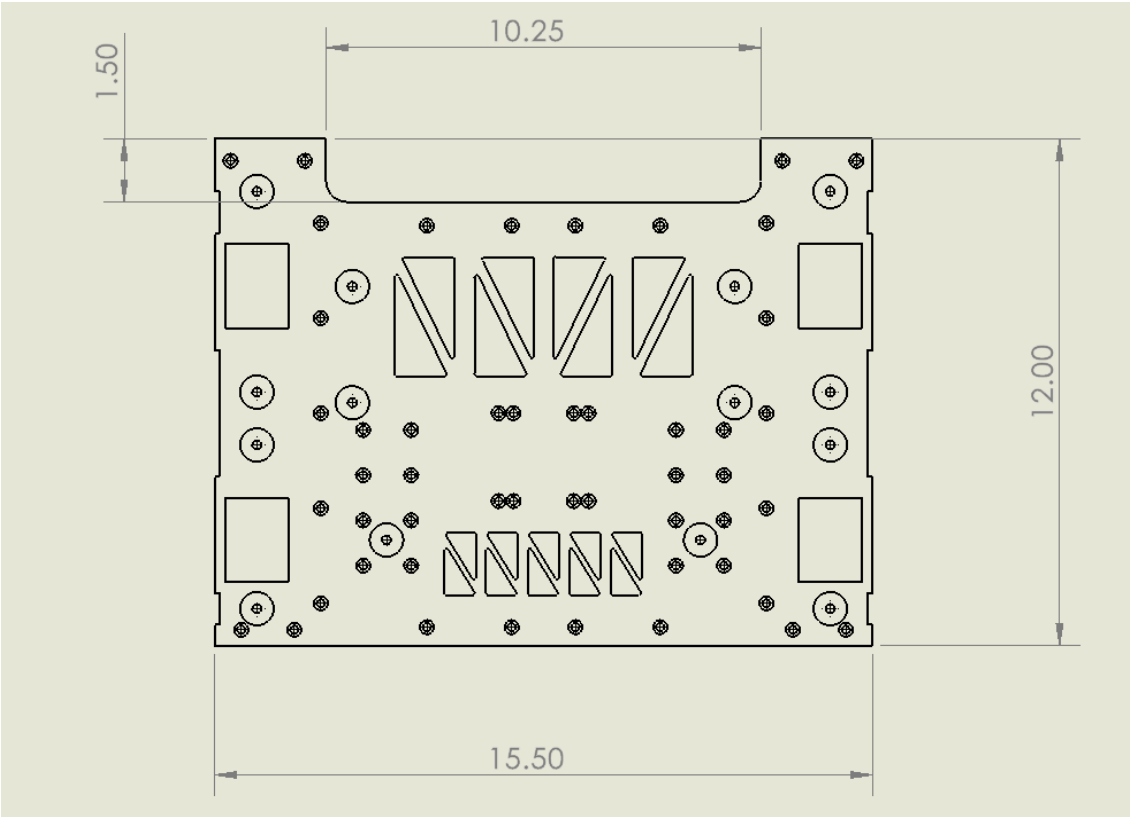


Fig 5. Overall dimensions of our robot: 12” x 15.5” with a 1.5” deep corral for the puck in the front.

This satisfies the sizing requirement of being less than 16” x 16” x 16”, and the possession rule of controlling only half of the 3” diameter puck.

Performance Goals:

- Effective weight = 92 -100 lbs
- 2 Motor 4WD
    - Secondary wheels attached to drive wheels via chain and sprockets
- 7 mph (10 ft/sec) cross field in 2 sec
- Win a push battle against bot of similar weight (5 sec without stalling)
    - Assume static 100 lbs bot = 45.4 kg
- Drive over steel surface with minimal slippage

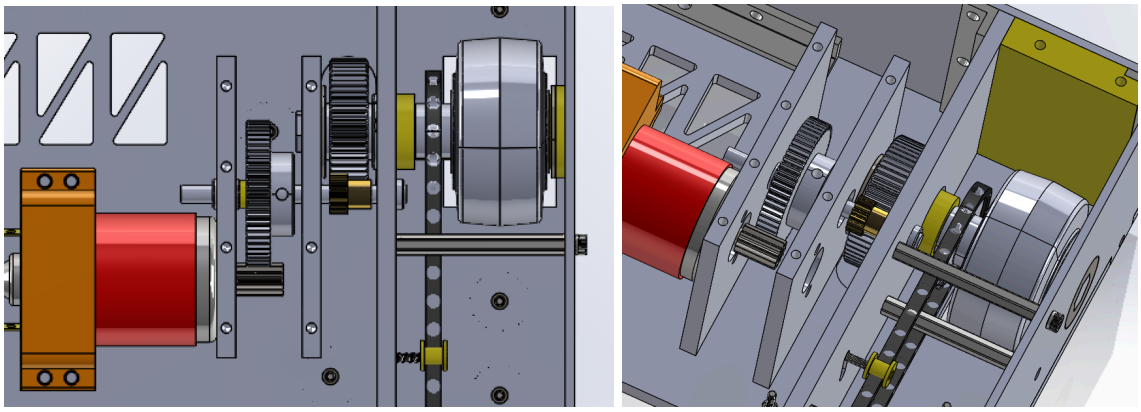


Fig 6. Our gearbox features a 20000 RPM, 327W Andymark Redline Motor attached to a two-stage 20:1 gearbox reduction. This is connected to a 1:1 chain and sprocket transmission between two 2.5” Colson Performa wheels.

This design fulfills all of our performance requirements at less than 10% of the motor’s stall torque. We can further utilize the motor’s power by increasing the effective weight with magnets.

Part III: Experimentation

The summary of the key milestones and the lessons we learned at each milestone are:

1. **Status Update 2**

   Date(s): November 3rd - November 11th, 2024

    - Through experimentation and testing, we were able to determine that some things thought needed in the first schematic were actually unnecessary, such as the breadboard and breaker.

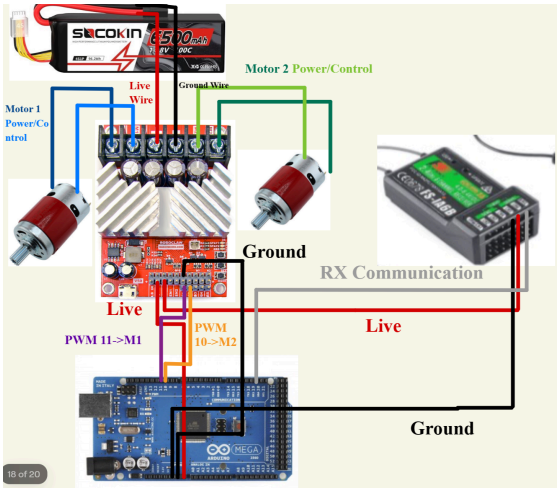


Fig 7. Enforcer Electronic Schematic

- We found that the joystick outputs on the flysky controller can lag or output non-zero values when supposedly set to zero, especially the left stick. Therefore, we avoid this stick and put all driving control on one stick using an “arcade drive”. Furthermore, we can set a TOLERANCE value in the program so that if the joystick is close enough to zero, we drive straight or in a pure turn.

```
/* We have three conditions, pure straight,
* a pure turn, and diagonal drive
*/
// PURE STRAIGHT
if (abs(leftRight) < TOLERANCE && continue_check) { // Give ourselves a small tolerance for driving straight
  rightInput = frontBack;
  leftInput = frontBack;
  continue_check = 0;
}
// PURE TURN
if (abs(frontBack) < TOLERANCE && continue_check) {
  if (leftRight > TOLERANCE) { // turn right
    rightInput = -leftRight;
    leftInput = leftRight;
    continue_check = 0;
  }
  if (leftRight < -TOLERANCE && continue_check) { // turn left
    rightInput = -leftRight; //leftRight is now negative in this if statement
    leftInput = leftRight;
    continue_check = 0;
  }
}
// DIAGONAL
if (leftRight < -TOLERANCE && continue_check) { // turn left
  rightInput = frontBack; // set our non-turning motor to be our max forward/backward value
  joystick_angle = atan2(frontBack, leftRight); // get an angle from the two channels, like a vector
  joystick_angle = bounce(joystick_angle); // put this angle between 0 and pi
  leftInput = scaler(joystick_angle) * rightInput; // our turning motor should be a fraction of the driving motor
  continue_check = 0;
}
if (leftRight > TOLERANCE && continue_check) { // turn right
  leftInput = frontBack;
  joystick_angle = atan2(frontBack, leftRight); // get an angle from the two channels, like a vector. I realize now I could've just put in absolute values i
  joystick_angle = bounce(joystick_angle); // put this angle between 0 and pi
  rightInput = scaler(joystick_angle) * leftInput; // our turning motor should be a fraction of the primary motor
  continue_check = 0;
}

double rightMotor = speedValue(rightInput * duty_cycle * overdrive); // must scale the channel using duty cycle otherwise we can end up changing directions
double leftMotor = speedValue(leftInput * duty_cycle * overdrive); // Convert our values from channel outputs to roboclaw inputs


roboclaw.ForwardBackwardM1(ADDRESS, rightMotor);
roboclaw.ForwardBackwardM2(ADDRESS, leftMotor); // Assume right motor is M1
```

Fig 8. Arcade Drive and Joystick Tolerance Program

- Video Demonstration: https://youtu.be/h-MsLFtndCY

2. **Status Update 4**

Date(s): November 17th - November 24th, 2024

- We managed to get the solenoid fully integrated into the driving electronics for a full demonstration of the striker electronics.

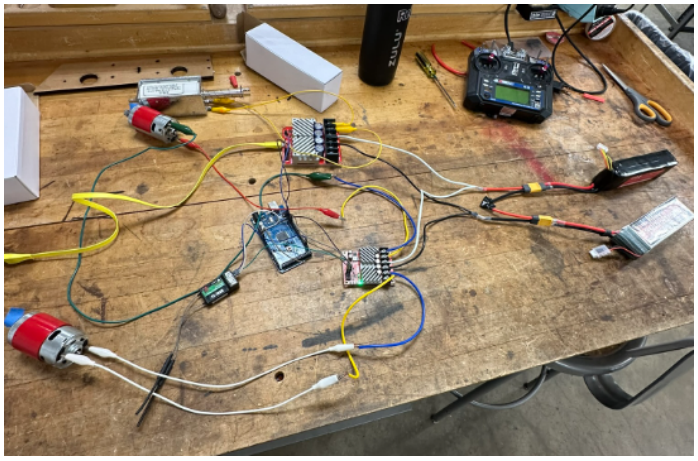


Fig 9. Experimentation with Striker Electronics

- Through experimentation, we found the optimal values to send to the robot claw for the solenoid and how to integrate its control with the kill switch.

```
// these readChannel lines should constrain us between -100 and 100
int leftRight = readChannel(0, -84, 84, 0); // one channel controls turning, one controls forward and back. this one is forward/back
int frontBack = readChannel(1, -84, 84, 0); // these are zero indexed compared to the flysky, so RC Ch1 is computer Ch0 etc
int killSwitch = readChannel(4, -100, 100, 0); // for stopping instantly
int strike = readChannel(5, -100, 100, 0); // for determining when to strike

if (killSwitch > 0){
  roboclaw.ForwardBackwardM1(ADDRESS, killSpeed);
  roboclaw.ForwardBackwardM2(ADDRESS, killSpeed);
  roboclaw2.ForwardBackwardM1(ADDRESS, killSpeed);
  // Serial.println(killSwitch);
  Serial.print("KILLED");
  delay(1000); // stops for a second
  continue_check = 0; // stop driving
  strike = 0; // stop shooting
}
// Serial.println(strike);
// First read the striking status
if (strike > 0) {
  // just directly send in the roboclaw input, should be high
  roboclaw2.ForwardBackwardM1(ADDRESS, int(MAXINPUT * SOLENOID));
}
if (strike <= 0) {
  // reset solenoid
  roboclaw2.ForwardBackwardM1(ADDRESS, killSpeed);
}
```

Fig 10. Striker Programming

https://youtube.com/shorts/V3VCJgxZoXk?si=UTOyVWdPStIHlK9a

- Through solenoid experimentation and working with different solenoid voltages, its abilities are a little underwhelming.
  https://youtube.com/shorts/V3VCJgxZoXk?si=l2Tc7Wmws1NtpjKu

3. **Mobility & Handling Demonstration**

Date: December 5, 2024

- During the driving, we noticed that better electronics mounting and a top plate would be beneficial to our design. Additionally, the arcade drive programming could be edited to use both sticks to make the controlling more intuitive. We also found picking up the robot difficult without handles, so we resolved to add those.

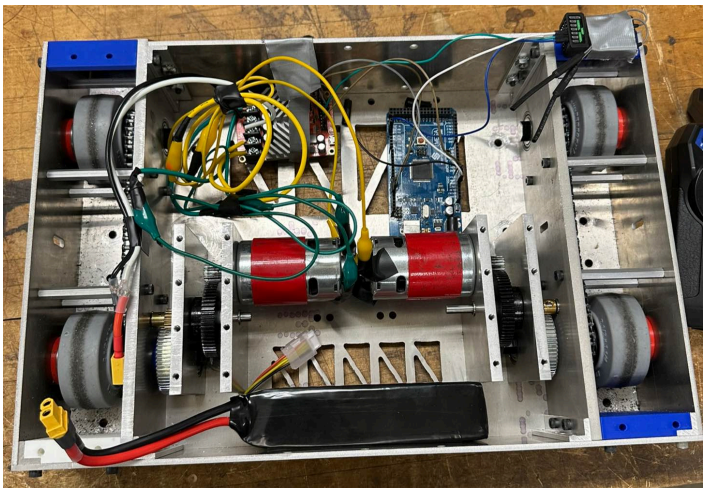


Fig 11. Enforcer Hardware Setup for Prototype

4. **Driving Improvements**
   Date: December 6th 2024
   - We experimented with and implemented a two-stick driving control and a simple gearing system that allows for precise speed and turning control: https://youtube.com/shorts/SxoisqLDBcY?si=n3I9XQYZ5T44Ae3l

Part IV: Future Plans

Budget:

We spent about $650 on the Enforcer Robot this term, which is more than we wanted (target budget per bot: ~$450)

Going forward we have a few ways we plan to budget better and save money for the remaining two robots:

1. Return the Solenoids ($55 each x 2 = $110): After testing we determined that the solenoids may not be the most effective way of launching the puck, and thus returning them could bring back the amount spent from ~$650 down to ~$550 for this term.
2. 3D printing: 3D printing parts such as gears/sprockets/low-stress structures can greatly reduce our cost of buying manufactured metal parts
3. Source better from the shop: We chose to buy a plate of ⅛” aluminum rather than source from the shop which ended up costing us double
4. Reduce Mistakes: We had ordering mishaps and machining mistakes that cost us a decent amount of budget.

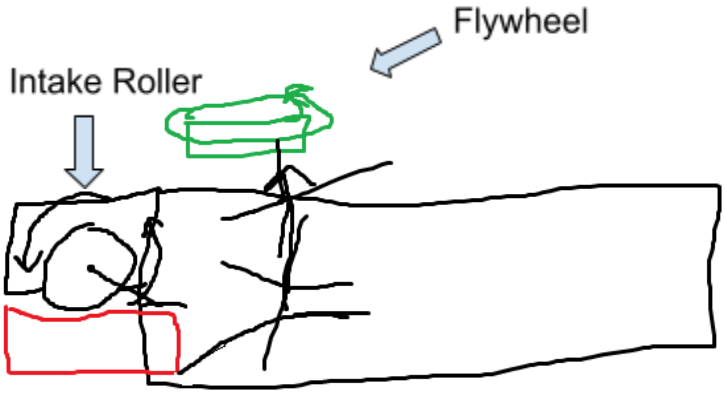
Design Flaws:

1. Magnets
   a. Remove 4 magnets currently from the enforcer, they interfere with our metal chain transmission

2. Chain vs Belt (considering using a belt instead of a chain to avoid magnetic interference, however, budget is a concern)
    a. Other ideas: Smaller sprocket on the enforcer to distance the chain from the bottom
        i. May allow for magnets to come back
        ii. Get chain tensioner to work, less impact from magnets
        iii. Repurpose larger sprockets for different robots needing fewer magnets

Future Designs:

1. Extending space in the interior to allow for a new intake and all electronics to fit
    a. Stick with 4WD instead of 6WD to save money
2. Striker
    a. Intake for collecting and shooting
    b. Roller intake (basic design) doubles as shooter
    c. Roller intake + transfer + flywheel (advanced design)



(see if we have time to figure out + budget over break)
Maybe a simpler brushless motor controller if speed/on-off control is not needed

3. Goalie
    a. Net for trapping
    b. Intake for trapping but not shooting (same as striker)

Part V: Standards

1. **Radio Control Standard**
    a. We use a Flysky FS-I6X radio controller. This unit operates on the 2.4GHz band and permits bidirectional communication between transmitter and receiver. It uses “multi-channel hopping” to switch between 16 available channels between 2.408 and 2.475GHz. The 2.4GHz ISM band is an unlicensed band allowed under Part 15 of Title 47. It’s often used by wireless LANs and meets the standards of IEEE 802.11. We operate the transmitter in an unrestricted area and the model we have comes with its own FCC ID: N4ZFLYSKYI6X. Therefore it meets all necessary regulations and considerations.

2. **Wire Gauge Standard**
    a. For our calculated slipping conditions, we found that each motor reaches 8.6A. Our lowest-rated robot claw can handle 30A in each of its two ports. To ensure that the wires can handle all the current draw that may be needed by the robot claw, they would have to be 8-10 AWG according to the NFPA 70 National Electrical Code 2014 Edition. We are currently using 20 AWG, which is passable but should be improved.
3. **Fastener /Tap Depth Standards**
    a. We are using commercially manufactured fasteners, thus we can safely assume that they meet the standards for ASME B18.3
    b. For the depth of our tapped holes, we are using a depth of 1.0 inches at a #8-32 (0.164" diameter) fastener size. A common rule of thumb is to use a depth of at least two screw diameters but after 4 screw diameters, the marginal gain in strength no longer outweighs the cost to manufacture a deeper thread. In our case, this nominal range falls into 2*0.164 - 4*0.164 = 0.328" - 0.656". Knowing this, we can afford to cut down on our thread depth and save cost (time) manufacturing.
    Reference:https://www.jaxmfg.com/blog/design-guide#:~:text=Tapped%20Hole%20Depth,while%20the%20manufacturing%20cost%20skyrockets.
4. **Sheet Metal Standard**
    a. We used ¼" multipurpose 6061 sheet aluminum and ⅛" multipurpose 6061 sheet aluminum. According to the distributor, McMaster, the ⅛" sheet follows ASTM B209 standards, and the ¼" follows ASTM B221 standards.

Part VI: Safety

The key elements that require safety considerations and the measures we have taken are:

1. Electronics: All wiring is properly insulated and secured via electrical tape and shrink tubing. Further tape is used to prevent any wires from interfering with the moving mechanical parts. Additionally, we have ensured that the entire team attends the battery charging and safety tutorial for the LiPo batteries we use in our bot.
2. Programming: There are multiple safety features built into the program executed by the onboard Arduino. This includes a killswitch on the remote controller and duty cycling within the programming to ensure none of the motors are overvoltage. When the killswitch is flipped, all motor speeds are reset to zero, stopping everything instantly.

```
void loop() {
  // Define duty cycle parameters
  //
  bool continue_check = 1; // basically lets us choose when to continue to the next loop
  double duty_cycle = MOTORVOLT / (4 * LIPO4S); // for scaling our inputs into the motors
  int killSpeed = 64; // for setting our motors to zero velocity
  int rightInput = 0;
  int leftInput = 0; // these inputs values are from [-100, 100] and need to get scaled and cycled
  double joystick_angle = 0.0;
  double overdrive = 1.0; // an overdrive value for temporary boost in speed and power
  // int max_speed = 100;
  // int cycle = (int)(max_speed * duty_cycle); // take the floor by casting to int, always > 0

  // these readChannel lines should constrain us between -100 and 100
  int leftRight = readChannel(3, -84, 84, 0); // one channel/stick controls turning, one controls forward and back. this one is turning
  int frontBack = readChannel(1, -84, 84, 0); // these are zero indexed compared to the flysky, so RC Ch1 is computer Ch0 etc
  int killSwitch = readChannel(4, -100, 100, 0); // for stopping instantly
  int drivetrain = readChannel(6, -100, 100, 0); // three gearings, slow, medium, fast

  // set up killswitch with a small delay that skips over all driving
  if (killSwitch > 0) {
    roboclaw.ForwardBackwardM1(ADDRESS, killSpeed);
    roboclaw.ForwardBackwardM2(ADDRESS, killSpeed);
    Serial.println(killSwitch);
    Serial.print("KILLED");
    delay(1000); // stops for a second
    continue_check = 0;
  }
```

Fig 12. Killswitch and Duty Cycling Programming

3. Moving Parts & Mechanical Safety: The enforcer bot has high-torque motors and moving parts that could potentially cause an injury if mishandled. To address this, we have designed the robot with clearances around moving parts and have incorporated protective 3D-printed covers.

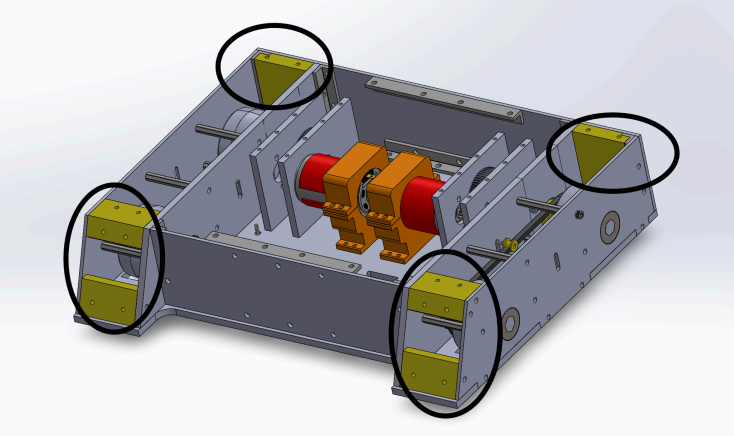


Fig 13. Circled are the top, back, and front mounts of the enforcer bot that act as protective covers

4. Weight: Since the enforcer bot is relatively heavy, ensuring that no one gets injured from this bot is vital. As a result, our team ensures that the robot is not operated near people without appropriate precautions to avoid any injury.