DAY 1 10/11/2024

Jimmy prepared a meeting agenda and then we went around discussing our experiences and technical abilities

Jimmy: Robotics experience, can work in multiple areas but is best at mechanical
Jabri: Almost only mechanical/machining
Luke: Can do electrical/software/mechanical, will fit in wherever. Software is the weakest area
Jaylen: Strong Solidworks experience, prefers mechanical stuff
Sara: Strong in mechanical, wants to try everything
Me (Jade): Strong in software/mechanical, can do electrical. Lots of Caltech robotics experience

Subteam Discussion:

Mechanical Team Lead: Jimmy
Design Team Leads: Jaylen/Luke
Electrical/SW Team Lead: Me (Jade)

Floaters/Want to try everything:
Sara
Jabri

Organizational
A. Team Names?
 i. **Penguinators**
 ii. TBD
B. Shared software stuff
 i. Google Drive?
 ii. **Box** [redacted]
 iii. Git?
 iv. **Team Communication Through Discord**
C. Weekly
i. At least one full team meeting per week (mandatory)
 - Probably 2-3 per week until CDR is done (weekly meetings Sunday afternoon 1:30pm)
 ii. PDR Meeting Monday
 ii. 1-2 other sub-team organizational/building meetings per week

JM

Robot Discussion

Goalie Robot:

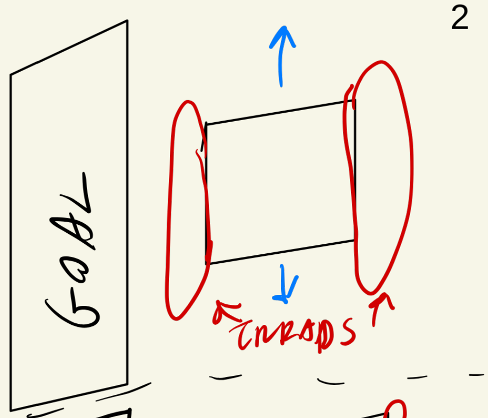
I. Side Mounted Intake?
II. Wheels Move parallel to the goal (side to side)
III. Goalie shooter could be an elevated flywheel similar to the striker?
A. Leaning towards single flywheel
B. Would rub us up against speed shooting limit (25 mph)

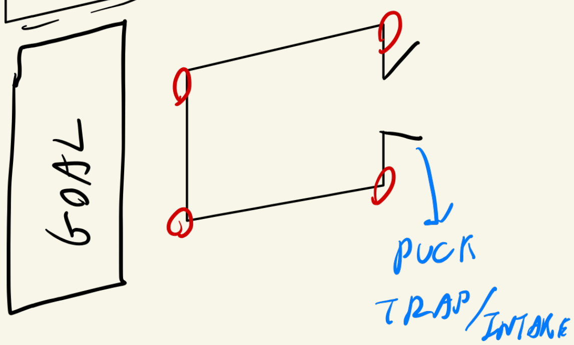
Enforcer Robot:
A. Focus on bot vs. bot combat
B. Moving defenders/goalie on offense, disrupting opponents on defense
C. Higher weight/more magnetic force
D. "Fork" rammer thing for scoring
E. Make the outside

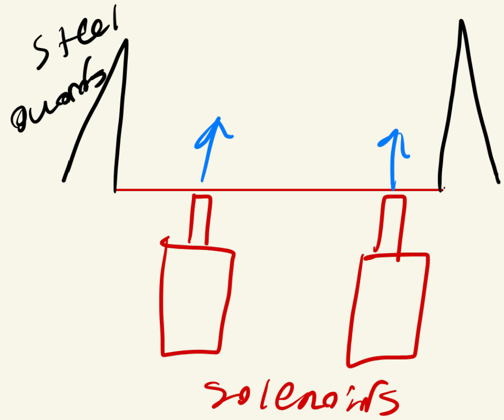
Electromagnets:
A. Power consumption can be quite high (we are limited to 48 volts)
B. Able to turn off for "speed mode", turn on for "power mode"

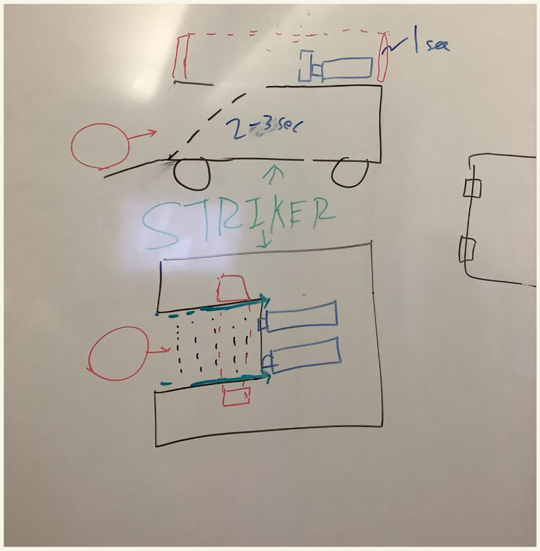
Striker Robot:
A. Solenoid Shooter
I. Quick release mechanism
B. Higher Speed
C. Shorter Turn Radius (wheels closer together)
D. H-Drive?
E. Intake (5 second) to elevate the puck and shoot horizontally

Action Items:
A. Drivetrain is the first thing to build
B. PDR Due soon
C. Next Meeting on Sunday
I. Sara will do all three robot sketches
II. Jaylen will research electromagnets and power
III. Me (Jade) will do solenoid research
iv. Jimmy + Luke: Drivetrain/Motors/Material/Gear ratio Research
v. Jabri: flywheel sketch + research + strategy calculations









Solenoid Research

Something to consider, longer stroke length would allow a larger landing area for the puck, increasing the chance that the solenoid will launch it farther

WE WANT $v \approx 20$ mph ≈ 8.94 m/s

$K_E = \frac{1}{2}mv^2 = \frac{1}{2}mv^2$ $m \approx 0.160$ kg ← FIELD HOCKEY PUCK

$K_E = \frac{1}{2}(0.160)(8.94)^2$ 6.3939 J → SOLENOID NEEDS TO PRODUCE THIS

$W = F \times d = K_E \Rightarrow F_{sol} = \frac{K_E}{d}$ → Stroke Length

JM

FZELBET DC: ~~80N = 0.160 J / 0.015 m~~ ← NOTE THESE MAY BE STANDARD FORCES

80N · 0.015m = 1.2 Joules

uxcell: 60N · 0.010 = 0.6 Joules

HESCHEN: 30N · 0.015 = 0.45 Joules

80N · 0.02 = 1.6 J → WOULD GIVE ~10 mph

Heyiarbeit ↗

↳ COULD GET TWO! ~14.09

https://www.amazon.com/Heyiarbeit-JF-1683B-12V-Electromagnet-Household-Appliance/dp/B08Z7CPK47/ref=sr_1_19?crid=FBNWKIOID9R1&dib=eyJ2IjoiMSJ9.7Njf79gxuaFLQnP6uOx3RtCg4yRRowXkkhw4MBGK4aPGjHj071QN20LucGBJIEps._NXL9DiAmsd9i_9Og1ifqFZEU8ZYpfOxIfcyD5SxSBs&dib_tag=se&keywords=high+force+push+solenoid&qid=1728783493&sprefix=high+force+push+solenoid%2Caps%2C133&sr=8-19

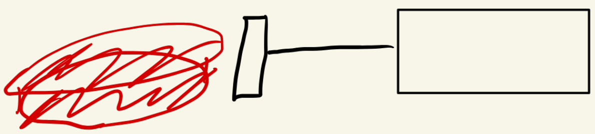
Jm

Voice coil actuators?

TANB 20 N voice coil actuator

$m_{puck} = 0.17$ kg

$a_{puck} = 20N$



V=0.050 m/s & actuator too slow

Two good solenoid options

Better stroke length, known high speed, unknown force, previous experience working with it (134):
https://www.amazon.com/Abletop-Solenoid-Electromagnetic-Electric-Automobiles/dp/B07G15X91N/ref=sr_1_8?dib=eyJ2IjoiMSJ9.2Z1H_beUJKbrov1wVlA5DEFJrht_02yoLkgxD3LO_pxNcO4aBfhb4IhaaCR6VsZLOW9dSmiwn7TgI7lCjce_vwGx9hArJRGGlLcZIUQ9funabFD0lIJJuijepaxZ-Zxv3dKhFuiCjSCNuRbwzHteQM0vbVin337f4inhGHtQ5PCX6JgmhEqW5Y4Uvnq6coBwWmx8eT8SjiGy1ErPiEmHtwBFNyLtz5F-fxCuVGxJPEM._tx9UxwTs3DG8EjppUU1VcmF58dWKkTzTa3CZJ-VJAc&dib_tag=se&keywords=high+force+push+solenoid&qid=1728859409&sr=8-8

Known force, shorter stroke by 15 mm, more expensive (60 dollars vs 18), assumed high speed, no previous work experience:
https://www.amazon.com/Heyiarbeit-JF-1683B-12V-Electromagnet-Household-Appliance/dp/B08Z7CPK47/ref=sr_1_19?crid=FBNWKIOID9R1&dib=eyJ2IjoiMSJ9.7Njf79gxuaFLQnP6uOx3RtCg4yRRowXkkhw4MBGK4aPGjHj071QN20LucGBJIEps._NXL9DiAmsd9i_9Og1ifqFZEU8ZYpfOxIfcyD5SxSBs&dib_tag=se&keywords=high+force+push+solenoid&qid=1728783493&sprefix=high+force+push+solenoid%2Caps%2C133&sr=8-19

Jm

ATTEMPT TO GAUGE PUCK SPEED

SOLENOID FORCE (TABLETOP)

15 cm in 0.01 sec

15 cm/sec = 0.15 m/sec

4.56 → PUCK IS HIT by solenoid

4.60 → PUCK hits other puck ~20 cm away

t = 0.04 s d = 20 cm

$\bar{v}$ = 5 m/s?

.62

.71

t = 0.09 s

d = 24 cm?

TRAVELS 15 cm in 0.02

~7.5 m/s

GUESS FORCE

WEIGHT ≅ 0.03 kg (solenoid interior)

F ≈ 21 N?

FULL STROKE in ~0.01 sec

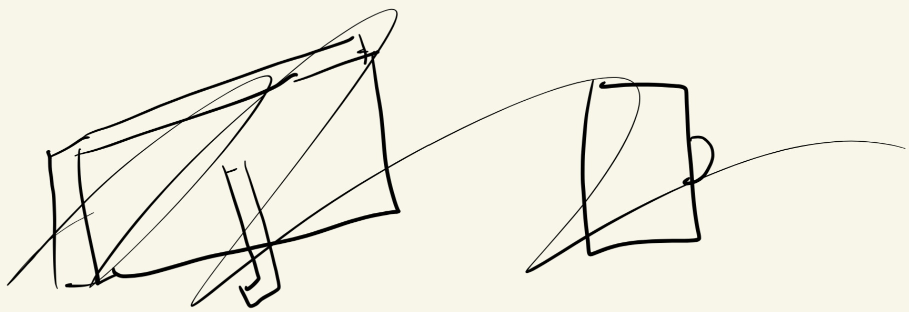
$V_{sol} = \frac{0.035}{0.01} = 3.5$ m/s??

$a = \frac{0.035 m \cdot 2}{0.01^2} = 700$ m/s²

JM

TENSION BASED MECHANISM?

Actuator: https://www.amazon.com/dp/B00NM8H5TG/ref=twister_B08DX9JDMY?_encoding=UTF8&th=1

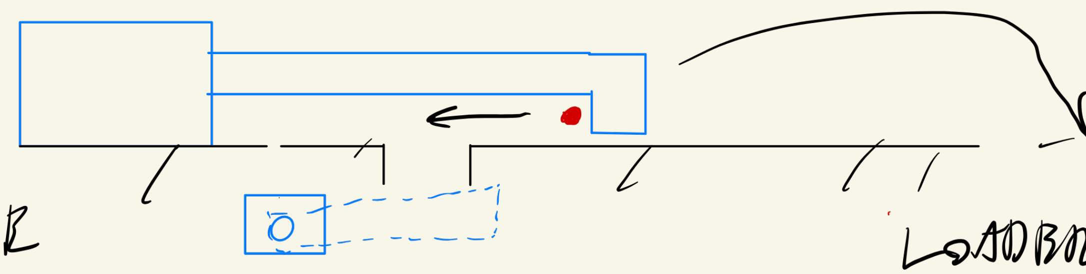


JM

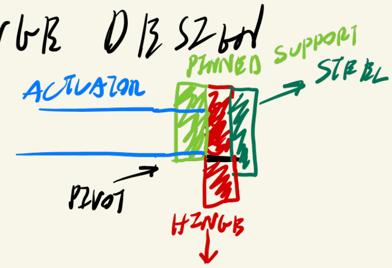
A hinged end on the actuator should allow the arm to regrab the rope while pulling it back. We can add steel reinforcement to make sure the tension won't break the hinge

SIDE VIEW

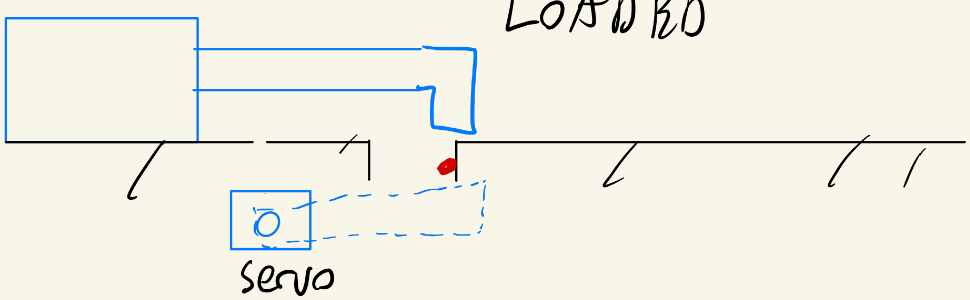
LOADING



LOADER HAS HINGE DESIGN

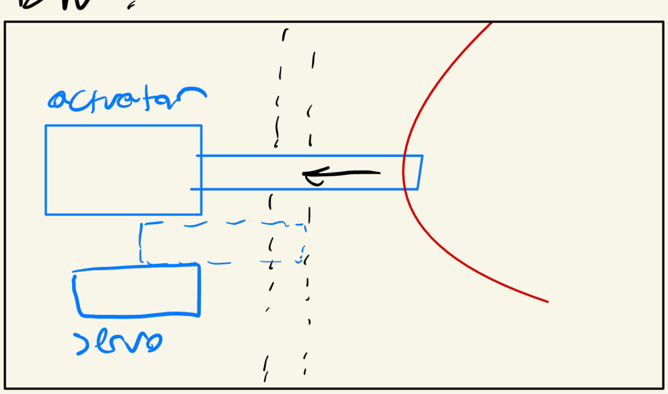


LOADED



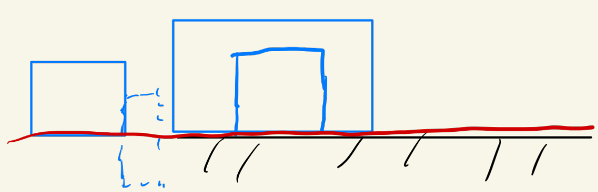
TOP VIEW:

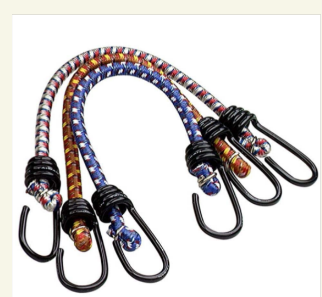
LOADING



Crossbow Inspiration: https://youtu.be/AHtxbVzcWYc

FRONT VIEW:





JM

FOLLOWUP MEETING 10/13/2024

Going with imperial units for everything due to Spaulding supplies and the guidelines given to us being in imperial units.

Work Updates

A. Electromagnets feasible for power consumption

i. during PDR, we should include reasons for the electromagnets and their specific weight

B. Flywheel seems to be non-optimal

i. 25 mph shoots halfway across the field at best with maximum speed at maximum height

ii. Since the goalie won't be able to score, should we worry about using the flywheel anyways?

Pre-PDR Meeting

Mostly discussing presentation details and things we've missed

Gonna add a couple more ideas like a spring moved wall and double use belt.

First Post-PDR Meeting (10/18/2024)

2 wheel drive better for agility and turning.
4WD is better for traction (enforcer and goalie)

Sumo robots press fit magnets into the base of the chassis (aluminum chassis). The question now becomes about bending moment. We could place the magnets closer to the wheels, which would reduce the bending moment.

70A-80A wheels seem ideal for the robot based on bot hockey responses

I'm gonna work on CADing the solenoid shooter plus intake

COULOMB'S LAW FOR MAGNETS:

$F_1 = 5$ kgf (→ 49N) $r_1 = 4$ mm → $\alpha \approx 0.000784$

r_2 4+1 mm

$F_2 = ? =$ $F_2 = \alpha \frac{1}{5^2}$ → $F_2 = 31.36$N or 3.197 kgf

If we want 100 lbf:

$\frac{100}{3.5} \approx 29$ magnets

3.5 lbf per magnet

20mm diameter with 8kgf more efficient?

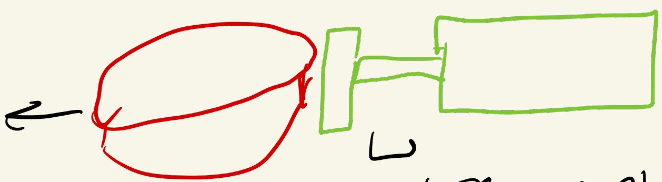
JM

feel confident in 4 inch wheels, how can we get a higher durometer so that they don't break or compress?

chain and sprocket design, higher torque motor with big gear geared down to smaller gears on the wheels

10/19/24

IMPULSE/MOMENTUM CALCS



$\Delta t \approx 0.01$ s (upper bound)

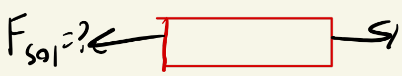
$F = ?$

$m_{puck} = 0.11$ kg → ≈ 3.9 oz

take $\mu_{puck} \approx 0.6$ (upper est for field hockey)

FBD

static friction



$friction = 0.6 \cdot (0.17 \cdot 9.8) = 0.9996\ N$

$F_{sol} \geq 1N$ in order to move puck

IMPULSE (J) = $F \cdot \Delta t$

$F_{sol} = 1N \cdot 0.01 = 0.01\ kg\frac{m}{s}$ ↳ lower estimate

NOW INCLUDE A OBSERVED FINAL SPEED, 20 mph or 8.94 m/s

$a_{sol} = \frac{9\ m/s}{0.01} = 900\ m/s^2$

$F_{acc} = 0.11 \cdot 900 = 99N$

$F_{acc} \approx 80N \Rightarrow V_{final} = 7$ m/s or 15.6 mph consistent with other estimates

CAN PUSH 77N OR 700 m/s² ↖

2 solenoids → 160N enough for 20 mph

THIS DEPENDS ON EVERYTHING BEING SETUP PROPERLY

LESS EFFICIENT SETUP IS LIKELY

Minimum distance for acceleration:

$$d = \frac{1}{2} \cdot 900 \cdot (0.01)^2 = 45 \text{ mm}$$

a Δt

STROKE LENGTH IS 35mm

$0.035 = 0.5 \cdot a \cdot (0.01)^2 \rightarrow a = 700 \text{ m/s}^2$

$F_{acc} \leq 700 \cdot 0.11 \text{ kg}$ 77N → max force possible by two solenoids

So, $V_f = 7$ m/s? ~ 15-16 mph
↳ Similar to previous estimates

1 solenoid optimal?

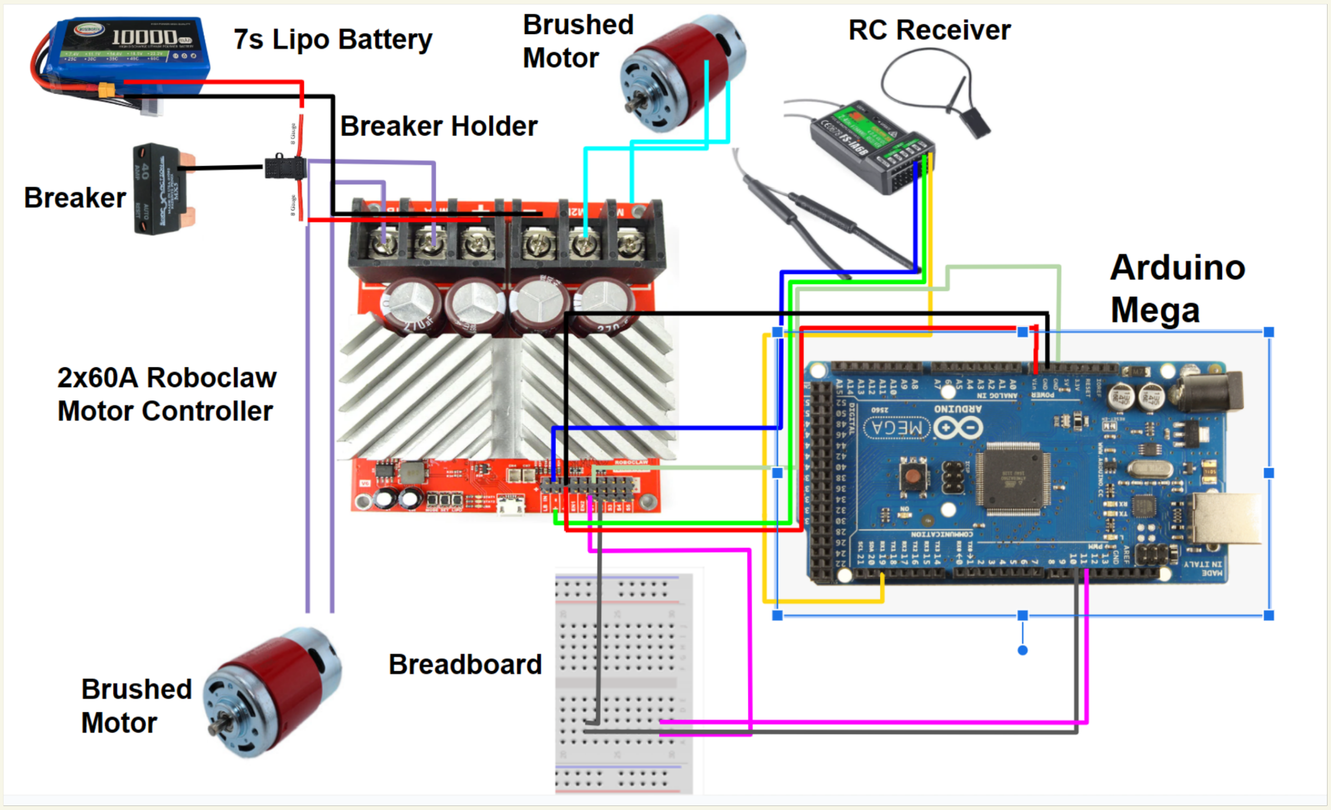
Final Thoughts:
Stroke length, not maximum speed or force of the solenoids, are what is holding the puck back from travelling faster. 35mm is consistently the longest stroke length at a reasonable price, $55-100 option that has 60mm to 100mm: https://www.amazon.com/LeTkingok-Stroke-Electromagnet-Push-Pull-Self-Reset/dp/B0CSVG8Y6T/ref=sr_1_6?crid=1CI268OP2VMEB&dib=eyJ2IjoiMSJ9.lD72rJKJuQXILXRMsvAqXqictIsDld8eUANoaA7qcN77nzeloayFUQdG1MiuLBtNGD1KlJhqBOInaMaC4sWtvsflFkEv-_z215jWE7xgFPpI7ShgZv-itLuUvxp_x3Wl_V6ssSVXYbhoZ0vhMDTO77i4kqpIPynDcmbN_UIhQglA9YgQPdCTz0UdgRw-qQw0O0VM_yTxjFstfKYNLKgK3AbhaAQPkEq3PBUm6a4MyN8.sbE1qHDRS2QmNp7kXQZetg1zArGYz-bb6X9x3KRqNUs&dib_tag=se&keywords=long+stroke+solenoid&qid=1729393590&sprefix=long+stroke+solen%2Caps%2C154&sr=8-6

Lots of estimation still. Puck weight, solenoid shooting time (0.01s might be a big overestimate), assumption that force, acceleration, and speed of the solenoid is pushing at a constant rate. Assumption of the puck being in the best place and not moving around, moving from static position.

In order to move the puck at 9m/s or 20 mph, two 24V solenoids from above would be needed at 60mm stroke length = $110ish

10/24/2024 (Pre CDR Meeting)

I mostly did a lot of stuff for the electronics, including finishing the schematic, doing research on electrical components. I have also been working on the CAD for the shooting mechanism.



Schematic is above

JM

Post CDR Meeting

Discussion of things after the CDR meeting. We talked mostly about workflow with the CDR and what went well and what didn't. One topic was bottleneck tasks, tasks that other things depended on that only one person could work on (like a CAD). These things made working on the CDR take much longer.

During the CDR, Jaylen and I had 2019 Solidworks running, so we couldn't work on Jimmy or Luke's CAD projects.

Three main CAD sub-assemblies: Gearbox (Luke), Robot frame/drivetrain (Jimmy), Shooting/Intake (Me)

I made a Discord server so we could have multiple channels for different discussions.

We figured out team shirt colors.

We then spent the rest of the meeting discussing prototyping materials and each of our responsibilities for prototyping different mechanisms.

With magnets mostly figured, we decided to move Jaylen to the intake mechanism.

JM

First Weekly Meeting with Mello (10/29/2024)

Talked about battery selection, BOM cost.
Talked about solenoid testing.
Two 4s batteries in series = 14.8 + 14.8 = 29.6V, more than enough for a 24V solenoid
Have started talking about buying materials.

A lot of discussion about the mock-up (it would take 2 weeks). Mello is concerned about the time it would take. We would use a lot of cheap/free materials. Definitely is a serious time sink but we think it would be worth it. Page against the machine did it, and we think the physical planning would be really useful.

Talked some about motor selection, why we picked these motors.

Meeting 11/03/22024

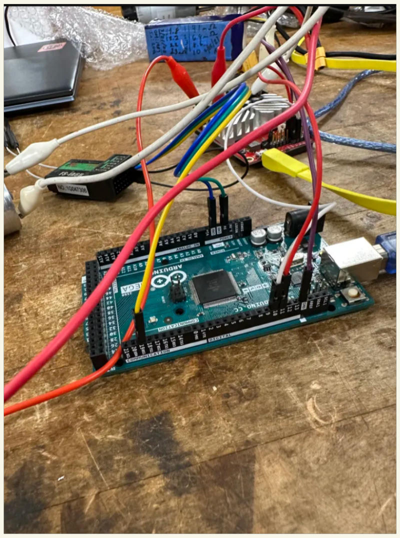
Most of the discussion was on finishing the chassis (Jimmy) and gearbox (Luke) as well as ordering more parts (Luke). My goal is to get a basic electronic system with the motors working. Jabri, Sara, Jaylen will work on the robot prototype. A big issue is getting the designs finalized so we can push forward. Bottleneck tasks are the chassis and gearbox.

We also worked on submitting the first Weekly Status Report

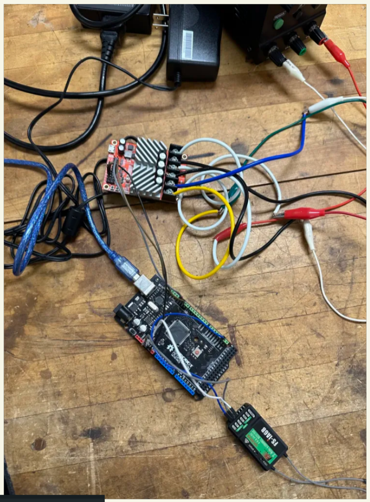
JM

Work Done 11/5/2024

Today I (with some help from Luke) worked on setting up the drivetrain programming and electronics. This included designing the circuit and algorithm for driving the motors using an Arduino Mega, Roboclaw 2x30A, jumper cables, RC, receiver, and two prototype motors very similar to the ones we are using. Below is the demo image from ME 14 that I used to start working with the hardware.



During setup and prototyping, we can plug into the Arduino and Roboclaw to set up their programming and receiving. Getting the motors to spin from RC signal took about a couple of hours.



This is an image of the new setup made today.

The remaining goals for this system are:

Short Term

- Fix motor speeds via duty cycling and programming
- Implement swerve driving (over tank driving)
- Implement a kill switch that will stop the program/motors instantly
- Create a detailed schematic with visuals and math where necessary

Long Term

- Soder or glue connections for robustness against shock
- Expand the circuit to test and prove solenoid can work alongside drivetrain

```
#include <IBusBM.h>

#include <SoftwareSerial.h>

#include <RoboClaw.h>
#define ADDRESS 0x80 // pretty common serial address should work
#define VOLTAGE 14.8 // Battery voltage
const long begin_speed = 38400; // For intializing serial communication

// Defin PWM input pins for reading remote control signals
const int PWM_PIN_LEFT = 3; // Left motor control
const int PWM_PIN_RIGHT = 2; // PWM signal pin for right motor control

// Create a RoboClaw object
SoftwareSerial serial(10, 11);
RoboClaw roboclaw(&serial, 10000); // Use Serial Communication

// Create iBus Object
IBusBM ibus;

// Read the number of a given channel and convert to the range provided.
// If the channel is off, return the default value
int readChannel(byte channelInput, int minLimit, int maxLimit, int defaultValue) {
  uint16_t ch = ibus.readChannel(channelInput);
  if (ch < 100) return defaultValue;
  return map(ch, 1000, 2000, minLimit, maxLimit);
}

// Read the channel and return a boolean value
bool readSwitch(byte channelInput, bool defaultValue) {
  int intDefaultValue = (defaultValue) ? 100 : 0;
  int ch = readChannel(channelInput, 0, 100, intDefaultValue);
  return (ch > 50);
}

void setup() {
  Serial.begin(begin_speed);
  // serial.begin(begin_speed);
  roboclaw.begin(begin_speed); // More initialization
  ibus.begin(Serial1); // for receiver
  // Set PWM input pins
  // pinMode(PWM_PIN_LEFT, INPUT);
  // pinMode(PWM_PIN_RIGHT, INPUT); // Set up the pins for input
}
```

```
  // pinMode(PWM_PIN_RIGHT, INPUT); // Set up the pins for input
}


void loop() {
  // Define duty cycle parameters
  //
  double duty_cycle = 12.0 / VOLTAGE; // bad practice but leave for now
  // int max_speed = 100;
  // int cycle = (int)(max_speed * duty_cycle); // take the floor by casting to int, always > 0

  // Cycle through first 5 channels and determine values
  // Print values to serial monitor
  // Note IBusBM library labels channels starting with "0"

  int speedLeft = readChannel(2, -100, 100, 0);
  int speedRight = readChannel(1, -100, 100, 0);

  speedLeft = duty_cycle * speedLeft;
  speedRight = duty_cycle * speedRight;

  // // Read PWM signals and remote control
  // int pwmLeft = pulseIn(PWM_PIN_LEFT, HIGH); // Read PWM for left motor
  // int pwmRight = pulseIn(PWM_PIN_RIGHT, HIGH); // Read for right

  // Map PWM values to motor speeds (note the usage of duty_cycle values)
  // int speedLeft = map(pwmLeft, 1000, 2000, -cycle, cycle); // Map to RoboClaw range
  // int speedRight = map(pwmRight, 1000, 2000, -cycle, cycle); // Map the right

  // Limit motor speeds to avoid invalid values
  speedLeft = constrain(speedLeft, 0, 127);
  speedRight = constrain(speedRight, 0, 127);

  roboclaw.ForwardBackwardM1(ADDRESS, speedLeft);
  roboclaw.ForwardBackwardM2(ADDRESS, speedRight); // Fundamental motor control lines

  // Print values for debugging
  Serial.print("Left Motor Speed: ");
  Serial.println(speedLeft);
  Serial.print("Right Motor Speed: ");
  Serial.println(speedRight);

  delay(100); // Wait 1 second
}
```

This is the code used for programming the motors (written in C++) using the Roboclaw library for the motor controller and the IBusBM library for the RC. Adjustments still need to be made.

JM

Some more detail on the RC and the steering used:

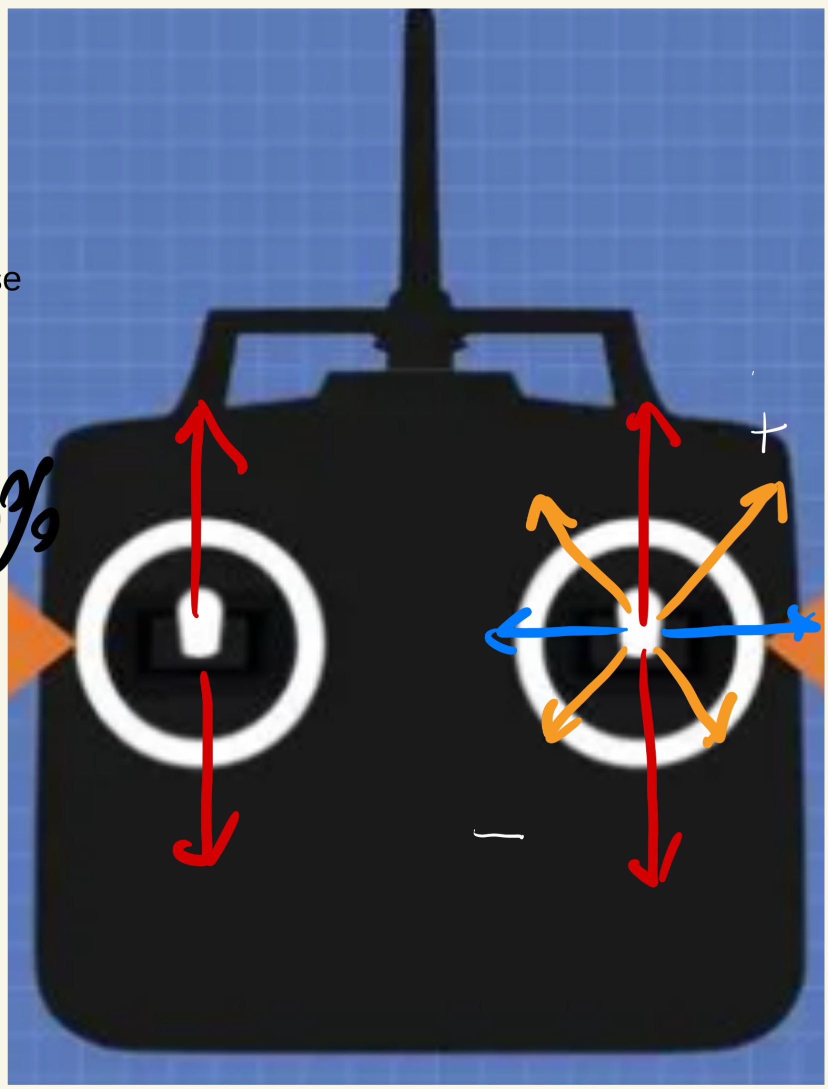
Each arrow points to a channel, 1-6, that are zero indexed in the code. For simplicity, I'll refer to each arrow/channel as the number depicted in the image. Currently, we are trying to use tank drive via controlling each motor's forward/backward with channels 2 and 3. However, it may be more advantageous to use swerve drive on the right joystick only, since the left stick has lag and doesn't reset since it's meant to control throttle and rudder on a plane (things that are precise and held in place for a long time).

I can design a pseudo-code system via diagramming on the right joystick that will allow us to control both motors and thus the whole robot's steering on one joystick. LM = Left Motor. RM = Right Motor. Speed given in percent output

ASSUME
CH1 + CH2 ≤ MAX

later found this was false
both can be at max

|CH 3| > 80%
IS KILL
SWITCH



CH2 ≠ 0, CH1 = 0
FORWARD / BACK
CH2 = LM, RM

CH2 = 0, CH1 ≠ 0
LEFT / RIGHT
CH1 > 0 : LM = CH1
TURN RIGHT RM = 0
CH1 < 0 : RM = CH1
TURN LEFT LM = 0

CH1, 2 ≠ 0
DIAGONAL DRIVE
4 CASES
CAN MAKE THIS MORE COMPLEX
CH1, 2 > 0 : LM = CH1+2
FOR/RIGHT RM = CH2

CH1 > 0, CH2 < 0 : LM = CH1+2
BACK/RIGHT RM = CH2

CH1 < 0 : LM = CH2
RM = CH1 +CH2

LEFT TURNS
TEST!!

JM

writing a new method for channel value to speed conversion so it's easier.
spent a few hours writing code today (11/7/2024) i was getting cooked

Goals accomplished from two days ago include:

- **corrected motor speeds**
- **kill switch**

Goal in progress:

- **swerve driving**

Goals to do:

- **schematic**

```
void loop() {
  // Define duty cycle parameters
  //
  bool continue_check = 1;
  double duty_cycle = 12.0 / VOLTAGE; // bad practice but leave for now
  int killSpeed = 64;
  int rightInput = 0;
  int leftInput = 0;
  // int max_speed = 100;
  // int cycle = (int)(max_speed * duty_cycle); // take the floor by casting to int, always > 0

  int leftRight = readChannel(0, -100, 100, 0); // one channel controls turning, one controls forward and back
  int frontBack = readChannel(1, -100, 100, 0); // these are zero indexed compared to the flysky, so RC Ch1 is computer Ch0 et
  int killSwitch = readChannel(4, -100, 100, 0); // for stopping instantly

  if (killSwitch > 0){
    roboclaw.ForwardBackwardM1(ADDRESS, killSpeed);
    roboclaw.ForwardBackwardM2(ADDRESS, killSpeed);
    Serial.println(killSwitch);
    Serial.print("KILLED");
    while(1);
  }
```

this is the kill switch code. it maps to a literal switch on the remote control that when flipped, stops the motors via an infinite while loop

Implementing swerve drive on one joystick has prove to be quite difficult. I am trying to map the forward/back channel and left/right channel outputs into the output of two motors. I've gotten close, but a bit more work needs to be done.

```
}
// Convert a channel value to a usable speed value via linear transformation
int speedValue(int channel) {
  // Use a linear transformation from [-100, 0, 100]->[0, 64, 127]
  int speed = ((channel + 100) / 200) * MAXINPUT; // 127 is our maximum roboclaw input
  return speed;
}

int sign(int value) { // easy way to check signs of ints
  if (value < 0) {
    return -1;
  }
  else {
    return 1;
  }
}
```

While working, I've had to create a couple functions to help me. One helpful function is the speedValue, which takes a channel output and returns the desired roboClaw motor speed. I use a linear transformation to get this. Another is a basic numeric sign function which isn't given default in C++.

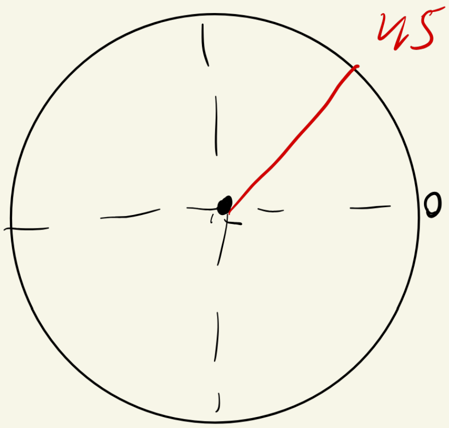
```
  if (sign(frontBack) == sign(leftRight)) {
    int leftInput = max(abs(frontBack), abs(leftRight)) * sign(frontBack);
    int rightInput = frontBack - leftRight;
  }
  else {
    int leftInput = frontBack + leftRight;
    int rightInput = max(abs(frontBack), abs(leftRight)) * sign(frontBack);
  }
  // PURE STRAIGHT
  if (abs(leftRight) < TOLERANCE && continue_check) { // Give ourselves a small tolerance for driving straight
    roboclaw.ForwardBackwardM1(ADDRESS, speedValue(frontBack * duty_cycle));
    roboclaw.ForwardBackwardM2(ADDRESS, speedValue(frontBack * duty_cycle));
    continue_check = 0;
  }
  // PURE TURN
  if (abs(frontBack) < TOLERANCE && continue_check) {
    if (leftRight > TOLERANCE) { // turn right
      roboclaw.ForwardBackwardM1(ADDRESS, speedValue(-leftRight * duty_cycle)); // assume M1 is right motor for now
      roboclaw.ForwardBackwardM2(ADDRESS, speedValue(leftRight * duty_cycle)); // turning in place
      continue_check = 0;
    }
    if (leftRight < -TOLERANCE && continue_check) { // turn left
      roboclaw.ForwardBackwardM1(ADDRESS, speedValue(leftRight * duty_cycle));
      roboclaw.ForwardBackwardM2(ADDRESS, speedValue(-leftRight * duty_cycle));
      continue_check = 0;
    }
  }
  // DIAGONAL
  else if (continue_check) {
    roboclaw.ForwardBackwardM1(ADDRESS, speedValue(rightInput * duty_cycle));
    roboclaw.ForwardBackwardM2(ADDRESS, speedValue(leftInput * duty_cycle));
  }
```

This is the bulk of the driving code. I have the edge cases of driving only straight or only turning down, but creating code that allows for both at once is difficult. Precise driving is key for this competition, so I want to create a scaled diagonal driving output. This involves linear transformations and some basic robotics/linalg knowledge.

JM

11/7/2024

$0 \le atan2 \le 90$



IF $atan2(CH1, CH0) > 45$ → left/right

↓

front/back

THIS IS IT!!

→ $RM = \frac{(atan2 - 45)}{45} \cdot LM$

(right motor)

→ scaled speed for turning

IF $atan2 < 45$

$-\frac{atan2}{135°} \cdot LM$

→ turn opposite for in place

JM

JM

AT 45, WE WANT ZERO

SO, $\frac{45+90}{2} = 67.5 \Rightarrow RM = \frac{1}{2} LM$

DIVIDE BY $2 \cdot 67.5 = 135$

→ USE LINEAR MAPPING

atan2 returns: $[-\pi, \pi]$

HOW TO MAP THIS TO $0 \to 90°$?

i.e. $-\frac{\pi}{4} = 315° \Rightarrow 45°$

IF $RM = \frac{(atan2 - 45)}{45} \cdot LM$

THEN WE CAN REACH FULL SPEED, TURN IN PLACE, AND DIAGONAL DRIVE.

JM

WE CAN "BOUNCE" ANGLES BACK INTO THIS RANGE

```
double bounce(double angle) {
  // return an angle between 0 and 90 or 0 and pi
  angle = fmod(angle, 2.0 * M_PI);  // limit by 180
  if (angle > M_PI) {
    angle = (2 * M_PI) - angle;
  }
  else {
    angle = -(2 * M_PI) - angle;
  }
  return abs(angle); // make sure we return a positive angle
}

double scaler(double angle) {
  // get a value from an angle of the joystick vector to scale the speed of the non primary motor
  double scale_factor = (angle - (M_PI / 4)) / (M_PI / 4); // convert to a value between [-1, 1]
  return scale_factor;
}
```

~~11/5/2024~~ 11/7/24

JM

A better working solution that will be tested tomorrow:

UPDATE:
NOT SWERVE DRIVE
BUT ARCADE DRIVE

```
// DIAGONAL
if (leftRight < 0) { // turn left
  rightInput = frontBack; // set our non-turning motor to be our max forward/backwa
  joystick_angle = atan2(frontBack, leftRight); // get an angle from the two channe
  joystick_angle = bounce(joystick_angle); // put this angle between 0 and pi
  leftInput = scaler(joystick_angle) * rightInput; // our turning motor should be a
}
else { // turn right
  leftInput = frontBack;
  joystick_angle = atan2(frontBack, leftRight); // get an angle from the two channe
  joystick_angle = bounce(joystick_angle); // put this angle between 0 and pi
  rightInput = scaler(joystick_angle) * leftInput; // our turning motor should be a
}

int rightMotor = speedValue(rightInput * duty_cycle);
int leftMotor = speedValue(leftInput * duty_cycle);

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

11/8/24

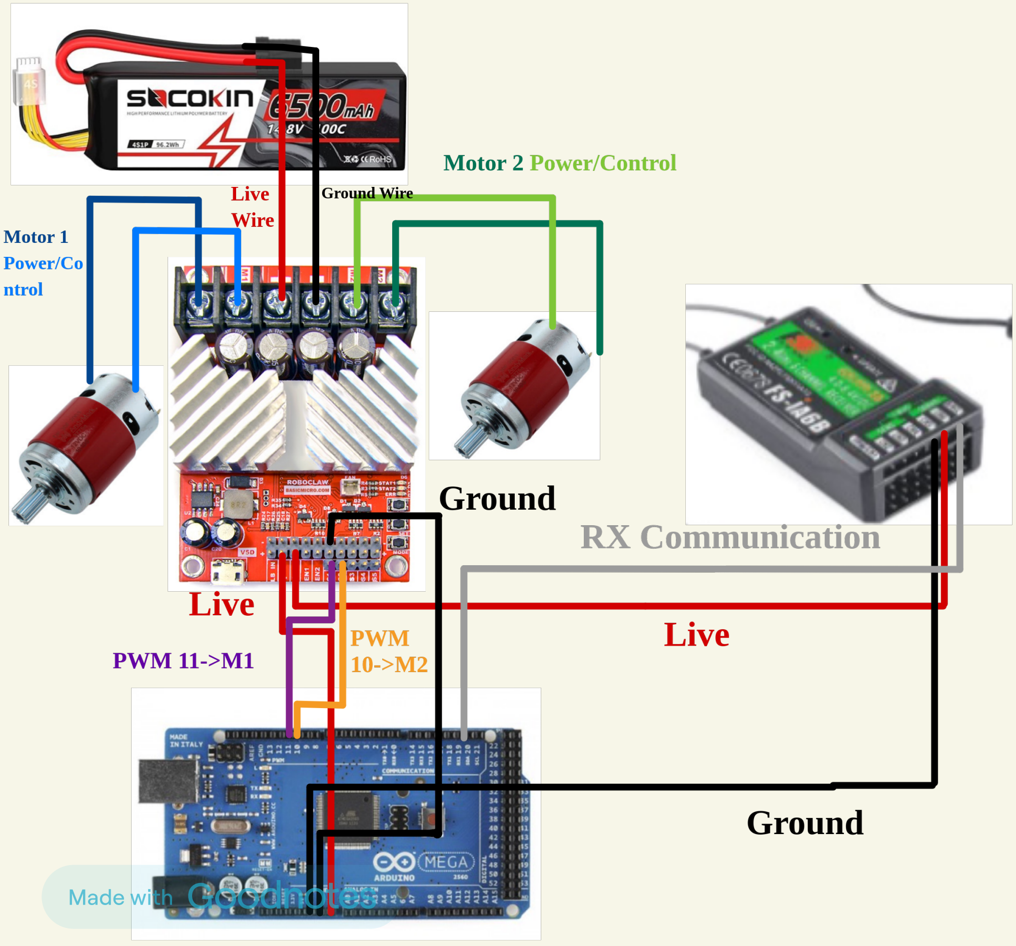
Goals Accomplished:
- **Circuit Design**
- **Motor control with RC**
- **Kill Switch**
- **Arcade Driving**
- **Duty Cycling**

The circuit works entirely off of one battery/power source. Currently, the kill switch turns everything off for one minute before restarting the algorithm ONLY if the switch has been reset. I have changed the circuit design to have the roboclaw power the receiver and arduino. This ensures both get enough power.

Long Term Goals:
- Test with LiPo (not urgent)
- Use 2x60A roboclaw and include solenoid in circuit
- Soder and glue connections so the circuit won't fall apart when driving/colliding

JM

Enforcer Schematic



```
#include <IBusBM.h>
#include <SoftwareSerial.h>
#include <RoboClaw.h>
#include <math.h>

#define ADDRESS 0x80 // pretty common serial address should work
#define VOLTAGE 48 // Battery voltage
#define MAXINPUT 128 // max roboclaw input for motor speed
#define TOLERANCE 5 // wiggle room in case joystick isn't purely zero
const long begin_speed = 38400; // For intializing serial communication

// Defin PWM input pins for reading remote control signals
const int PWM_PIN_LEFT = 3; // Left motor control
const int PWM_PIN_RIGHT = 2; // PWM signal pin for right motor control

// Create a RoboClaw object
SoftwareSerial serial(10, 11);
RoboClaw roboclaw(&serial, 10000); // Use Serial Communication

// Create iBus Object
IBusBM ibus;

// Read the number of a given channel and convert to the range provided.
// If the channel is off, return the default value
int readChannel(byte channelInput, int minLimit, int maxLimit, int defaultValue) {
  uint16_t ch = ibus.readChannel(channelInput);
  if (ch < 100) return defaultValue;
  return map(ch, 1000, 2000, minLimit, maxLimit);
}

// Read the channel and return a boolean value
bool readSwitch(byte channelInput, bool defaultValue) {
  int intDefaultValue = (defaultValue) ? 100 : 0;
  int ch = readChannel(channelInput, 0, 100, intDefaultValue);
  return (ch > 50);
}
// Convert a channel value to a usable speed value via linear transformation
double speedValue(int channel) {
  // Use a linear transformation from [-100, 0, 100]->[0, 64, 127]
  double speed = ((channel + 100) / 200.0) * MAXINPUT; // 127 is our maximum roboclaw input, need to cast to floating point to not get zero
  return speed;
}
```

```
int sign(int value) { // easy way to check signs of ints
  if (value < 0) {
    return -1;
  }
  else {
    return 1;
  }
}

double bounce(double angle) {
  // return an angle between 0 and 90 or 0 and pi
  angle = fmod(angle, M_PI);  // limit by 180
  if (abs(angle) > (M_PI / 2)) { // we have overshot and need to bounce back into our range
    angle = M_PI - abs(angle);
  }
  else {
    angle = abs(angle);
  }
  return abs(angle); // make sure we return a positive angle
}

double scaler(double angle) {
  // get a value from an angle of the joystick vector to scale the speed of the non primary motor
  double scale_factor = (angle - (M_PI / 4)) / (M_PI / 4); // convert to a value between [-1, 1] via linear transform
  return scale_factor;
}

void setup() {
  Serial.begin(begin_speed);
  // serial.begin(begin_speed);
  roboclaw.begin(begin_speed); // More initialization
  ibus.begin(Serial1); // for receiver
  // Set PWM input pins
  // pinMode(PWM_PIN_LEFT, INPUT);
  // pinMode(PWM_PIN_RIGHT, INPUT); // Set up the pins for input
}


void loop() {
  // Define duty cycle parameters
  //
  bool continue_check = 1; // basically lets us choose when to continue to the next loop
  double duty_cycle = 12.0 / VOLTAGE; // bad practice but leave for now
  int killSpeed = 64;
  int rightInput = 0;
  int leftInput = 0; // these inputs values are from [-100, 100] and need to get scaled and cycled
```

```
//
bool continue_check = 1; // basically lets us choose when to continue to the next loop
double duty_cycle = 12.0 / VOLTAGE; // bad practice but leave for now
int killSpeed = 64;
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;
// 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(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

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;
}
/* 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
```



```
    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 here and not bounce the angle but eh
  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); // must scale the channel using duty cycle otherwise we can end up changing directions
double leftMotor = speedValue(leftInput * duty_cycle); // Convert our values from channel outputs to roboclaw inputs


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

continue_check = 1; // reset our continue variable



// Print values for debugging
// Serial.print("leftRight: ");
// Serial.println(leftRight);
// Serial.print("frontBack: ");
// Serial.println(frontBack);

// Serial.print("rightMotor: ");
// Serial.println(rightMotor);
// Serial.print("leftMotor: ");
// Serial.println(leftMotor);

delay(100); // Wait 0.1 second
}
```

full code used

11/08/2024

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

https://xiaoxiae.github.io/Robotics-Simplified-Website/drivetrain-control/arcade-drive/
Luke later found this link on Arcade drive, and while my algorithm has similar fundamental logic, it is better in a few ways. Since it's a trigonometry based system, I can modify the pivot from pi/4 to 2*pi/3 to give more emphasis turning, or to pi/3 to give more emphasis on going straight. I also have extra cases that allow for easier turning in place and going straight.

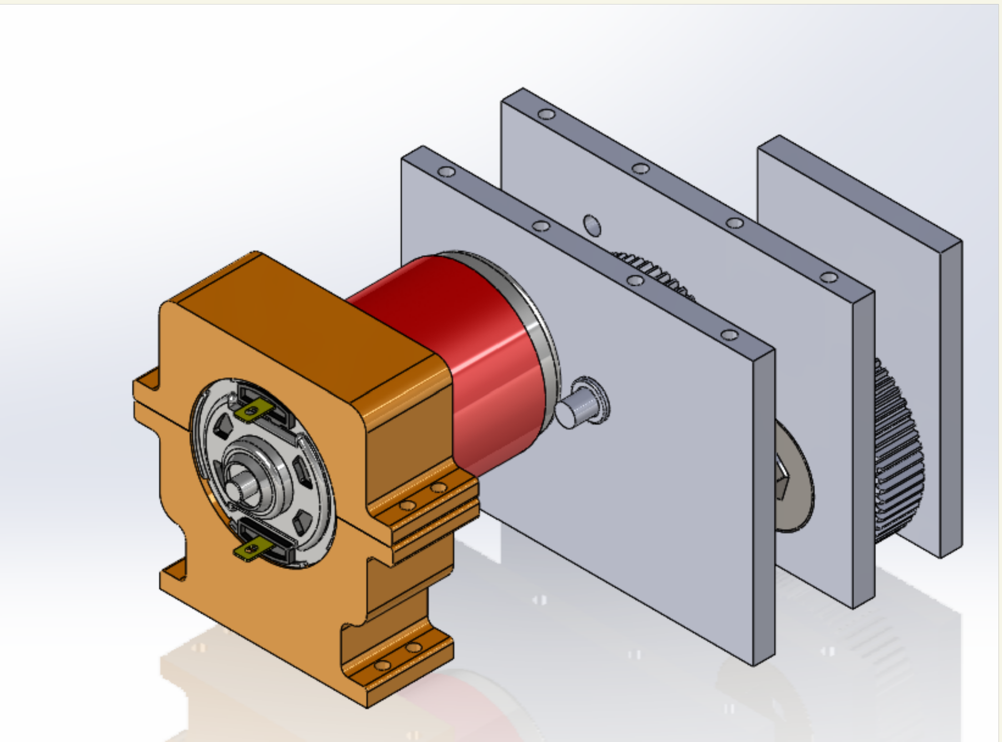
As head of electronics and software, it's my job to ensure my work is easily understood by my teammates and that I can communicate what I've accomplished effectively.

11/12/2024
Weekly meeting with Mello. My tasks for this week are to work with the solenoid and battery connectors and get a fully integrated circuit working with two batteries and driving + solenoid. This would be an electrical proof of concept for the goalie robot and the striker robot. Also to finish the motor mounts.

11/14/2024
Wrote over the settings to another roboclaw. I might write over to the 60A as well.

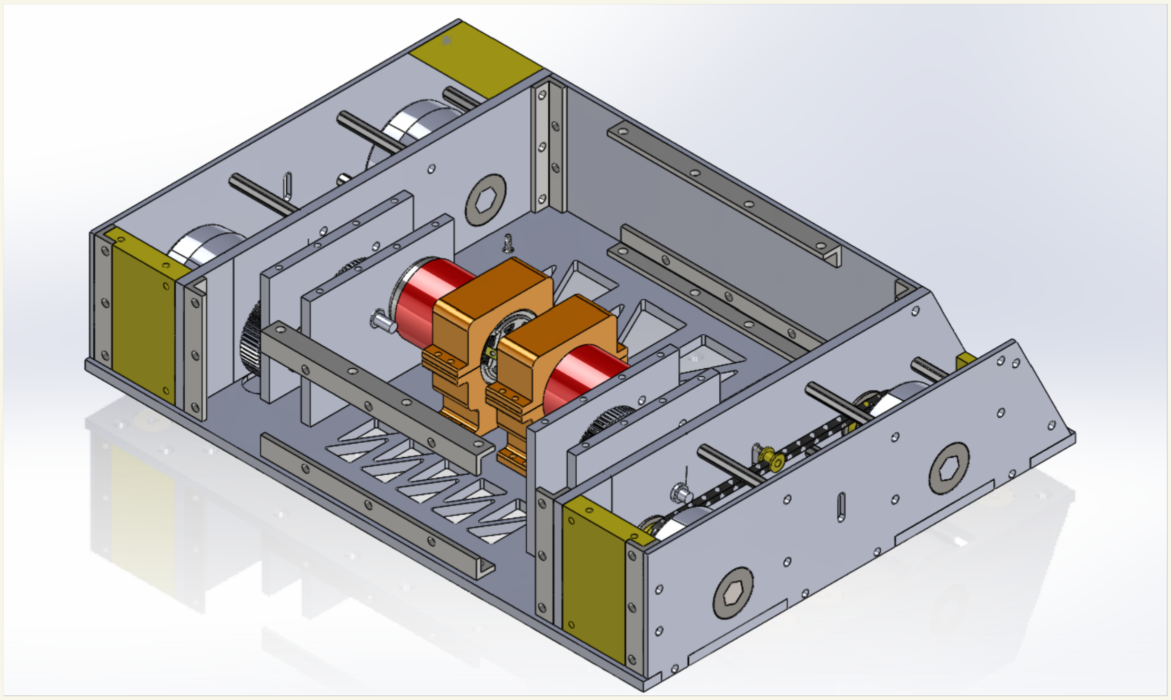
Got the right connectors on both batteries. Talked to Trent a lot about connecting them in series and splitting up the ground and live lines to power two roboclaws. It can be done but plenty of soldering needs to be done. Next up is some electronics work tomorrow, working with the solenoid and such.

I finished working on the motor mount CAD



It's in two sections so that we can 3D print easier. It should screw into the belly pan via through holes and screws with nuts.

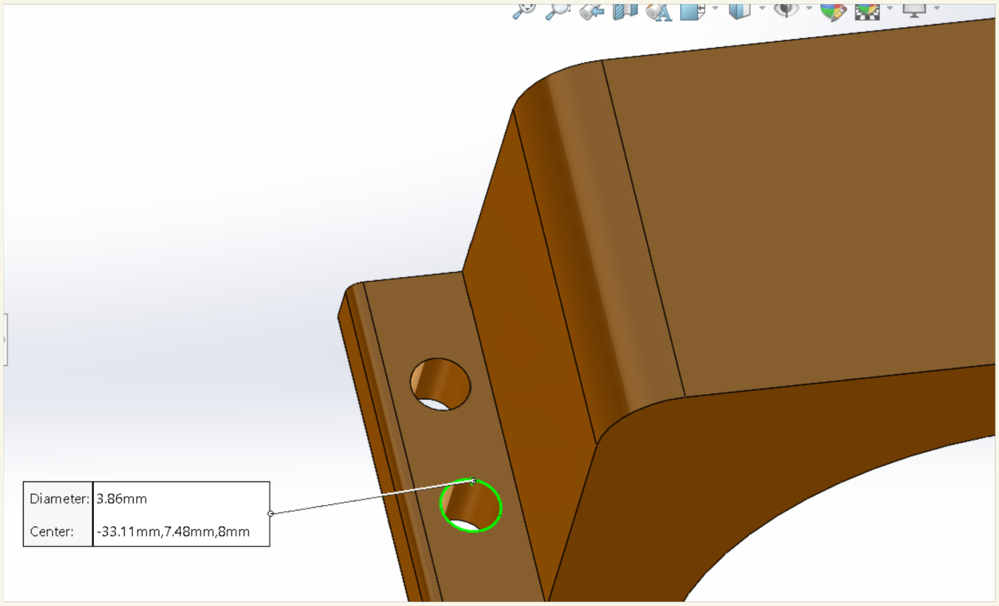
11/16/2024
I fixed the gearbox CAD and finished integrating everything into the big robot assembly.



11/19/2024
Meeting with Mello and team meeting. New batteries have arrived. We later had an independent team meeting working on organizing the building.

Jaylen will work on the L brackets for putting the frame together.

Me: Make motor mount holes 3.86 in diameter



JM

11/20/2024•

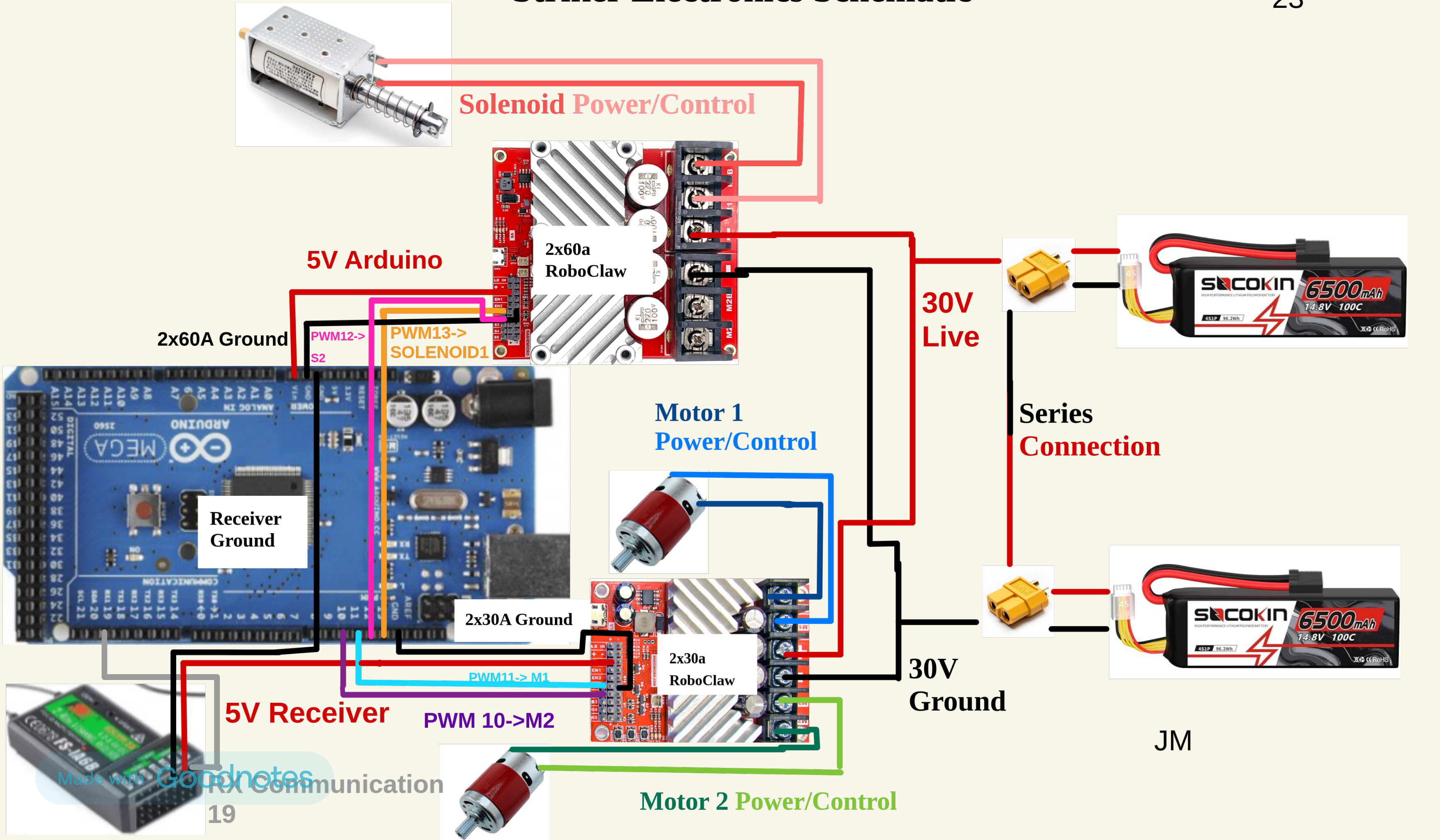
Today I worked extensively on testing the solenoid and putting the electronics in series. Paul helped me get a new adapter on one of the new batteries, which work great. I ran into a lot of issues setting up the circuit in series, as the robot was showing an error. After lots of debugging and TA help, I got the issue resolved (the light isn't showing anymore, not sure why) and was able to setup the circuit. I tested the solenoid movement with direct PWM control via the basicmirco software. The next step is programming it. Luke also sent me some stuff about a boost converter and a channel relay module that would allow us to bypass a second roboclaw and just use an on/off switch for the solenoid controlled by the arduino. This would involve instead connecting the batteris in series and im not sure about that. We could also use a buck converter? To step down from the 30V+ to 24V. More thought should be done about this, but this could be a next term problem as long as we have an electronics proof of concept.

Goals Accomplished:
- Solenoid Moving
- New batteries
- New battery connection
- Series battery connection
- Schematic

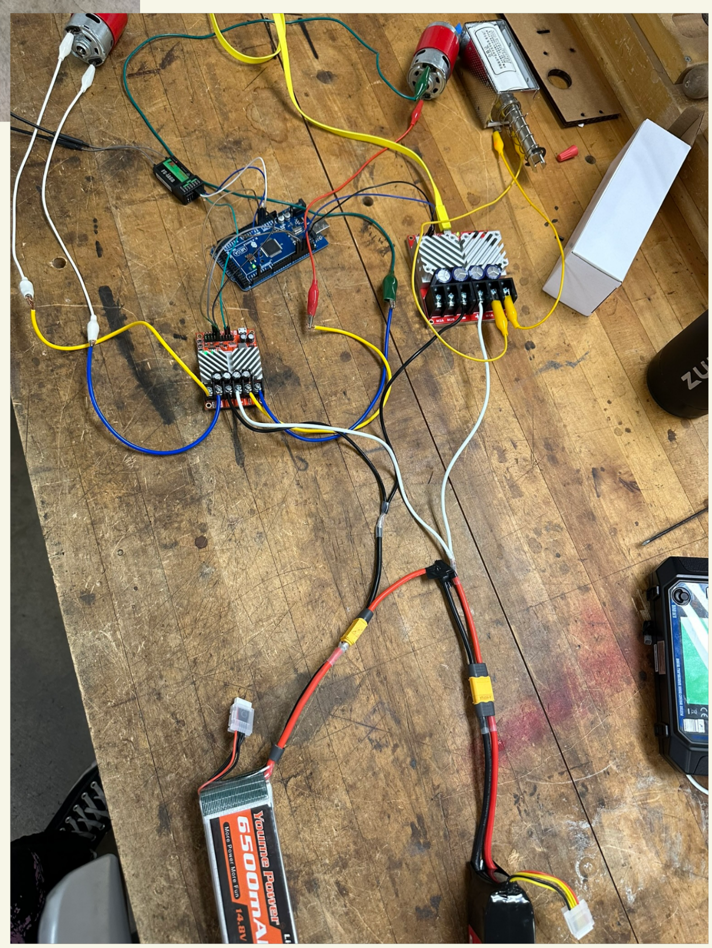
Goals To Do:
- Program Arduino to be able to control solenoid via RC
- Change connection of other battery
- File down new battery connection to make it fit better

JM

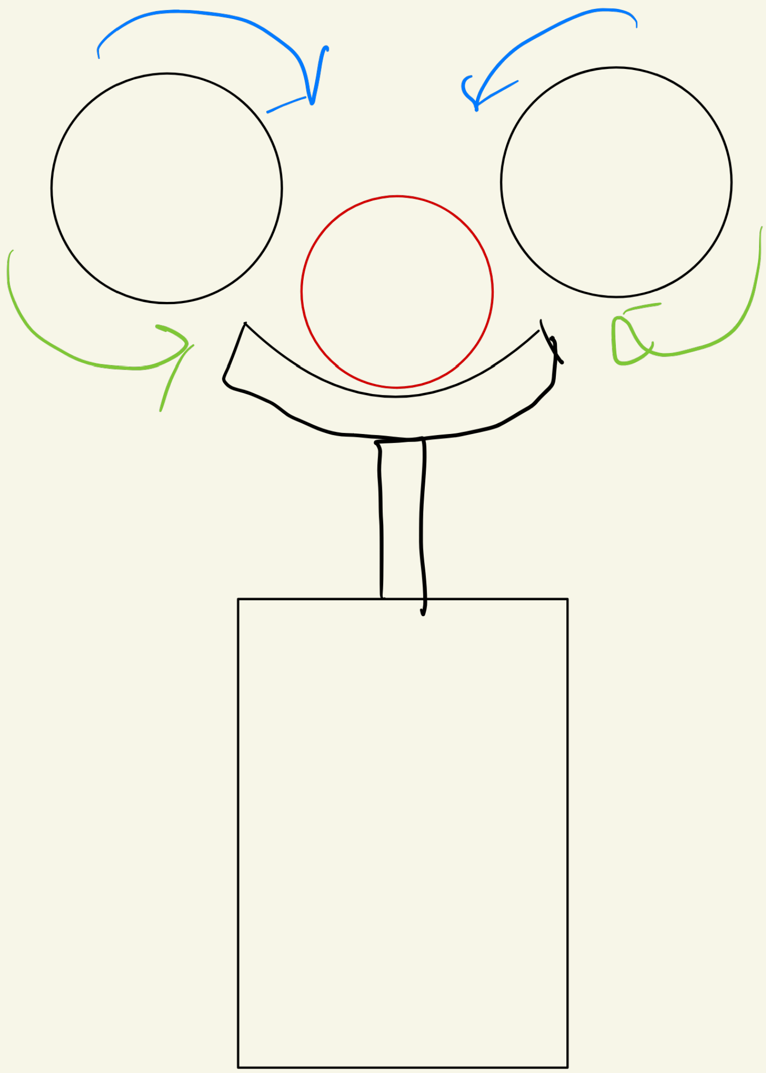
Striker Electronics Schematic



Pictures of irl solenoid testing



Concept design for a reversible intake/shooter mechanism using the solenoid.



11/21/2024
Today we tested the solenoid capabilites after finally getting it working in the circuit. It works in tandem with the kill switch and other circuit capabilities. There's a video of it being demonstrated here: https://youtube.com/shorts/V3VCJgxZoXk?si=l2Tc7Wmws1NtpjKu

Overall, it's abilities are a little underwhelming. It can move the puck decently, but nowhere near the top speed, and the puck slows down on the arena surface after traveling about 20% of the field length. I suspect stroke length is the limiting factor here, but it may be worth testing the two solenoids in tandem just to see. I should also make a proper end effector for the solenoid so the force is better applied to the puck. I would estimate the solenoid is indeed shooting in the 10-15 mph range we have previously estimated, but hard to say for sure.

JM

11/21/2024

Goals Completed:

- Integration and testing of solenoid in the striker circuit.

Goals to Do:

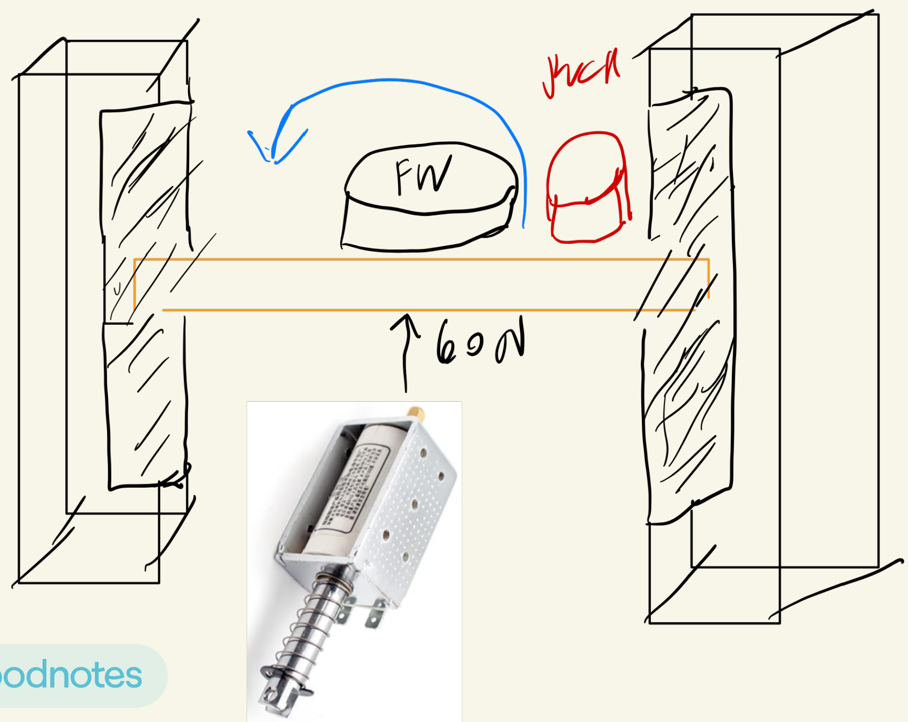
- Change connection of other battery
- File down new battery connection to make it fit better
- CAD and 3D print solenoid end effectors for one and two solenoids
- Start working on other ways to shoot like a flywheel and elevated shooting mechanism.

Code for solenoid shooting

```
// 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);
}
```

FRONT VIEW



ELEVATED FLYWHEEL VIA SOLENOID

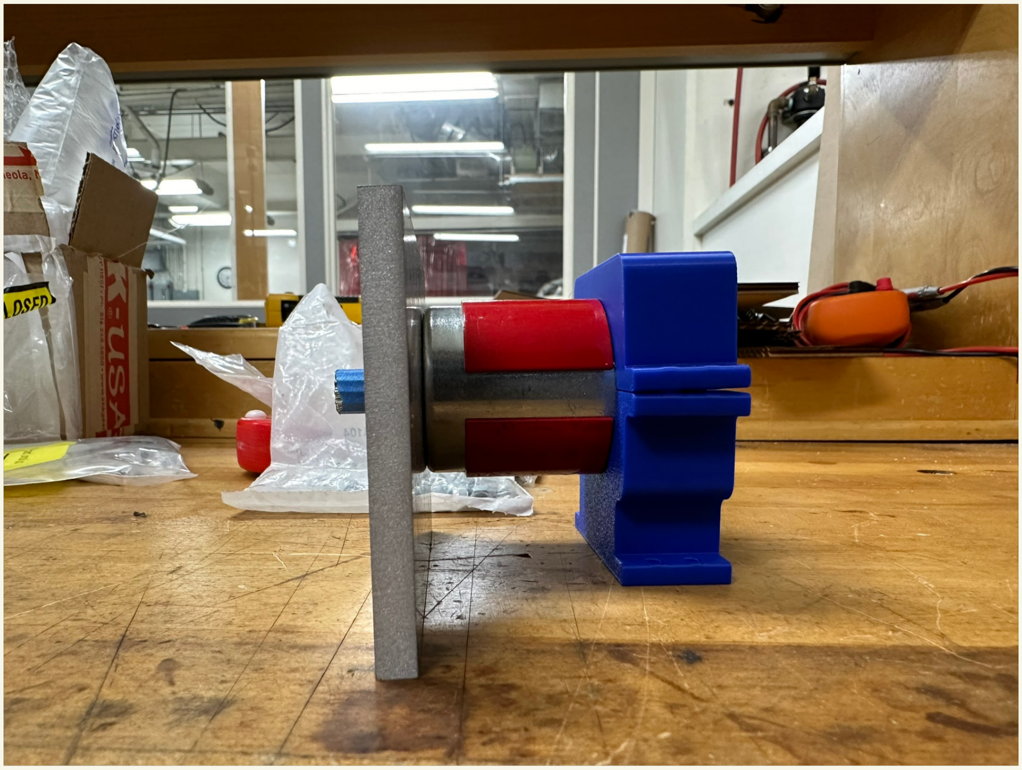
The idea here is that the solenoid could instantly elevate the flywheel shooting mechanism with the puck by 4 inches. The puck and mechanism would be constrained by "rails" on a moving carriage. CAD coming soon.

JM

11/24/2024

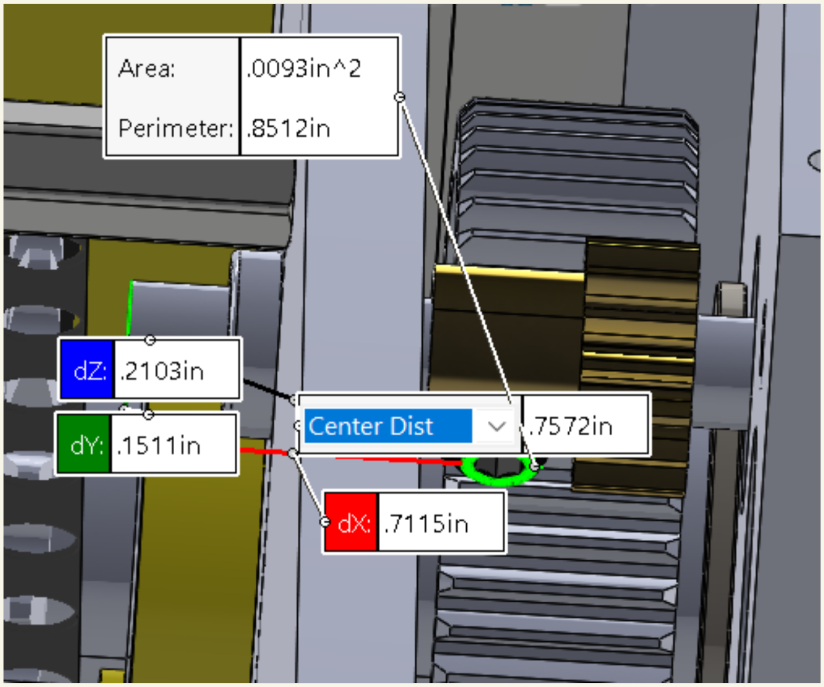
Met virtually with the team today. My tasks are to continue to work on the shooter prototype and solenoid, and then ask about gear pinning and shaft drilling for pinning our big gear. They are the two super small brass gears and the big gears that haven't come yet.

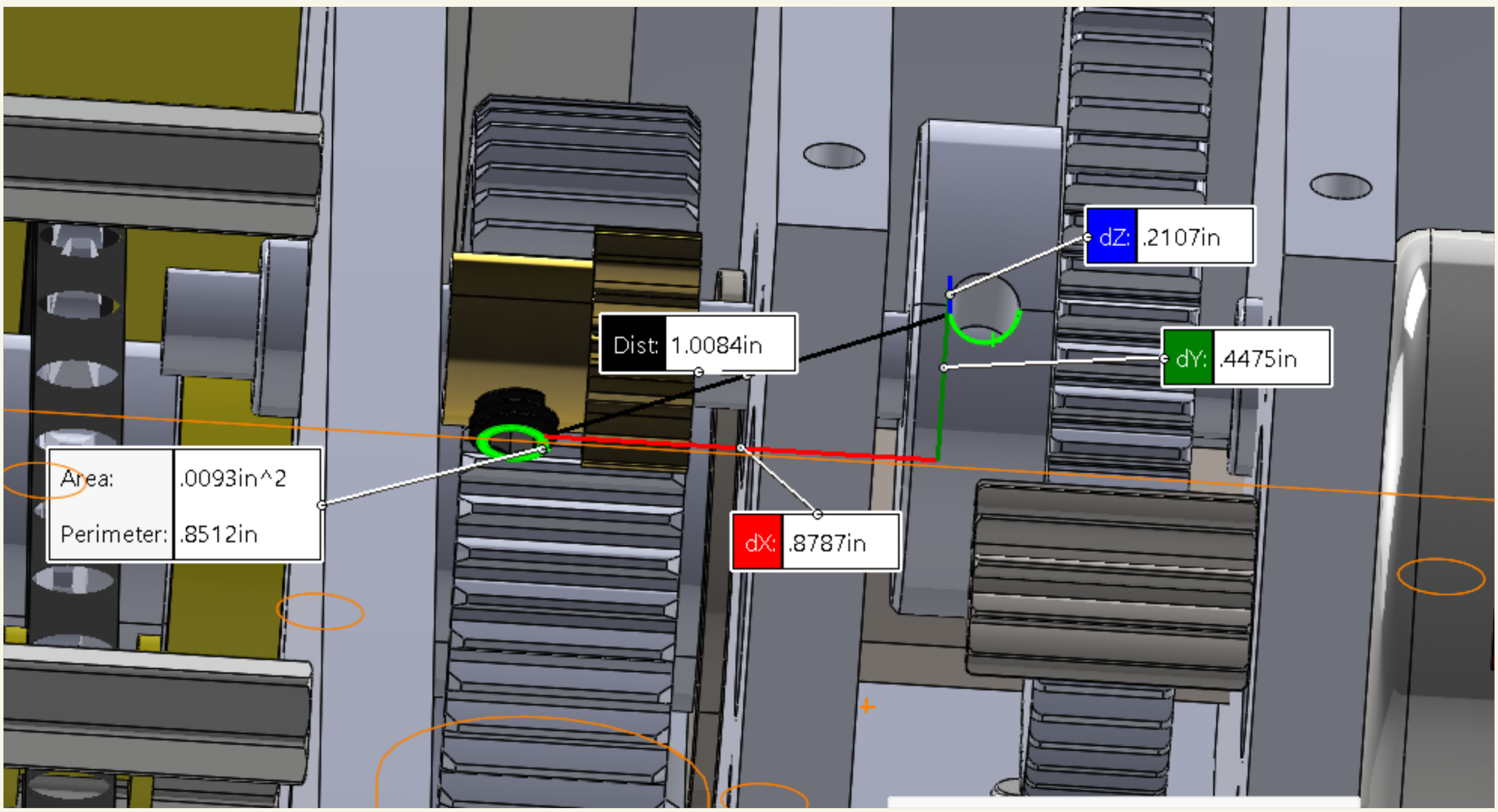
11/26/2024
Good machining progress today. Tested out the freshly printed motor mounts and gearbox walls to check fit, they look good.



We got shipped the wrong gears, so we are still missing two gears of 0.8 modulus for our gearbox. I've decided to go ahead and move forward with pinning the small brass gears on the shafts. I will work with Trent on this tomorrow morning. In order to pin the gears, we need to find out where they are relative to the end of the shaft.

JM





brass lil guy will be 0.7115 inches along the shaft, bigger gear will be 0.8787 inches further along than the lil guy

0.4725

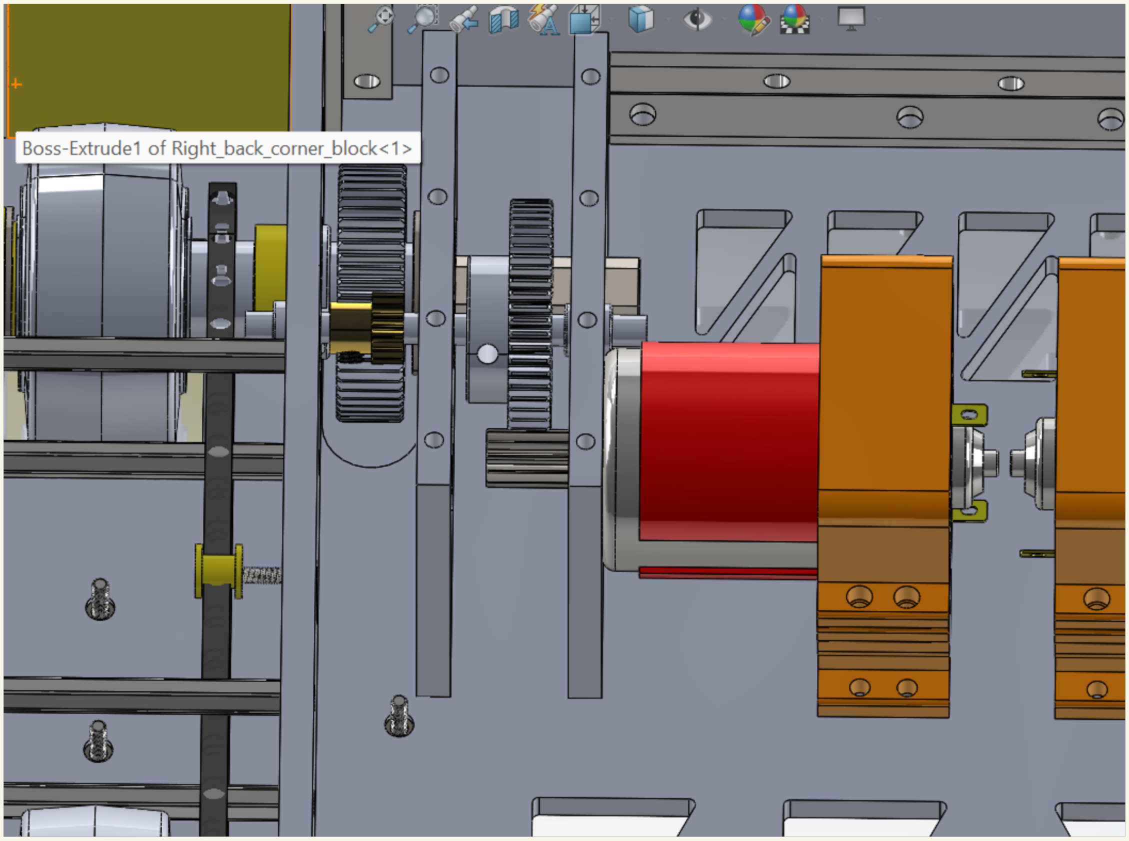
1.184

end to small

small to big

JM

Broader CAD reference picture:



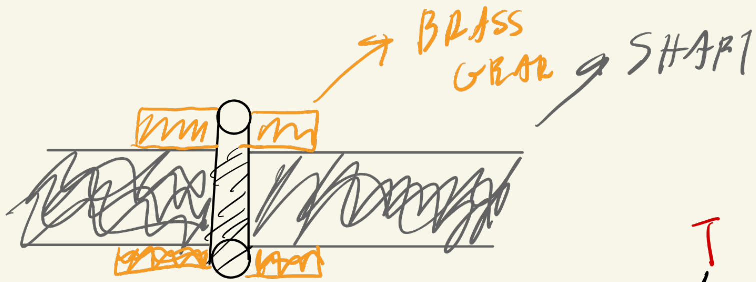
11/27/2024

When getting ready to work on pinning the gears today, I ran into technical difficulties. When using one of the set screws to hold the brass gear in place, the set screw ate through the brass material. Fortunately, I was able to use quick thinking and redrill and retap a hole for the set screw, allowing me to fix the gear into place for pinning. However, this took too much time and I wasn't able to work on pinning the gear.

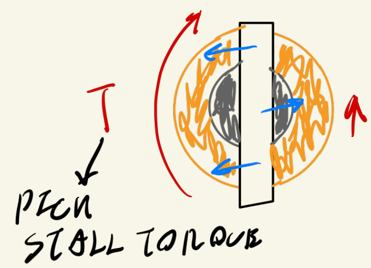
JM

12/01/2024

WE NEED TO FIND
DOUBLE SHEAR STRENGTH FOR PIN DIAMETER
AND MATERIAL



UNDER DOUBLE SHEAR



$F_{pin} = T \cdot \text{gear radius}$

$$\tau_{pin} = \frac{F_{pin}}{2 \cdot A_{pin}}$$

↳ MUST BE WITHIN PIN SHEAR STRENGTH

12/02 SUBMITTED WEEKLY REPORT

REDLINE MOTORS HAVE A STALL TORQUE OF 0.7N-m

BASED ON CPR CALCULATIONS, STALL TORQUE IS
7.552 lbs-ft = 10.25 N-m FOR TRANSMISSION

OD OF 15-tooth BRASS GEAR = 9.75mm

$r_{inner} - r_{outer} = \frac{9.75}{2} - \frac{6}{2} = \frac{3.75}{2}$ mm = 1.875 → HUB wall thickness

SPIROL DESIGN GUIDELINES: MAX PIN DIAMETER = $\frac{1.875}{1.5}$ = **0.935mm** → JM ~0.039"

↓

SHAFT CONSIDERATION: MAX PIN DIA. = 6 · 0.25 = 1.5mm
↓
SHAFT DIAMETER

12/02/2024

THROUGH PIN DOUBLE SHEAR FOR 15-tooth BRASS GEAR ON 6mm SHAFT

$T_{stall} = 10.25\ N\text{-}m$ $\quad F_{stall\ on\ pin} = 10.25\ N\text{-}m / 4.875 \cdot 10^{-3}$

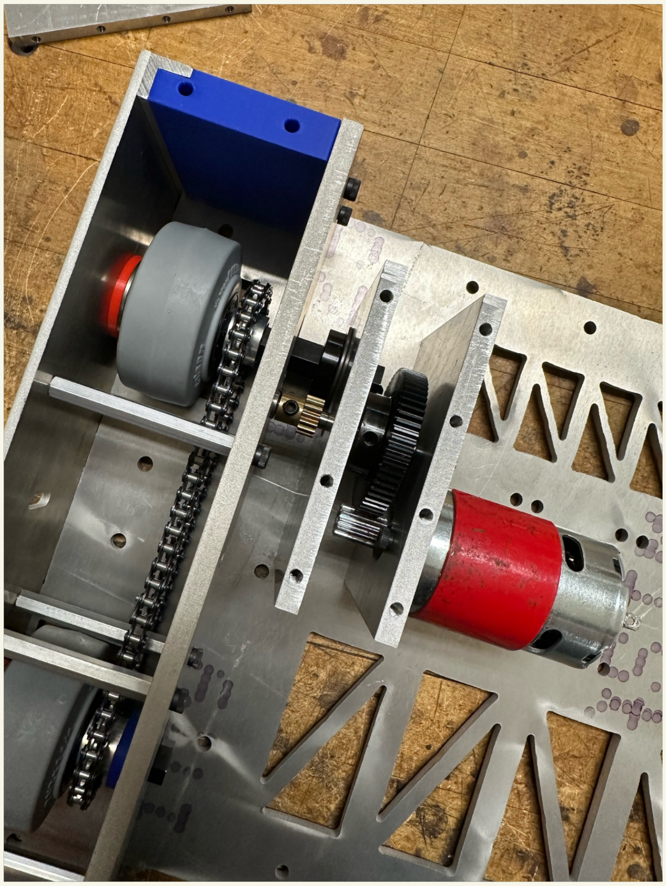
↓ wall of shaft thickness

$F_{stall\ on\ pin} = 2103\ N$

$$\tau_{pin} = \frac{F_{stall}}{A_{pin}} = \frac{2103}{\pi \cdot \left(\frac{0.935}{2} \cdot 10^{-3}\right)^2} = 3.063 \cdot 10^{9}\ Pa$$

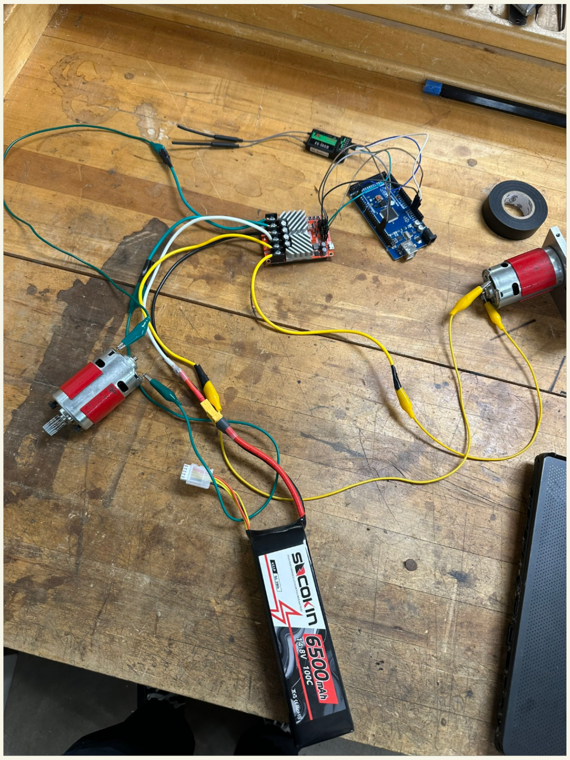
JM

12/03/2024



Today we met with Professor Mello. I went over my concerns on pinning the gears. In particular, I was concerned with the material strength of the brass gears as it would fail quicker than any steel/carbon steel pins. I had found high carbon steel pins 3/8 inch and 3/4 inch long with 5/64 diameter for the brass and blacksteel gear respectively. I was also concerned about the double shear strength of the pins on both the 15 tooth brass gear and the 60 tooth steel gear. While I had done the calculations on the pins and they seemed to hold up with a safety factor of about 1.5, I wasn't super confident in my understanding of the fundamental mechanics involved. Professor Mello gave me strong industry-adjacent advice, "Don't hold up the entire project because you aren't sure. You've done the work, and now its time to experiment and prototype. You wouldn't holdup a project deadline in industry by a week to just run more calculations, with projects you need to just try." So, we are going to try. I will pin the gears tomorrow. Today we confirmed the gear placement.

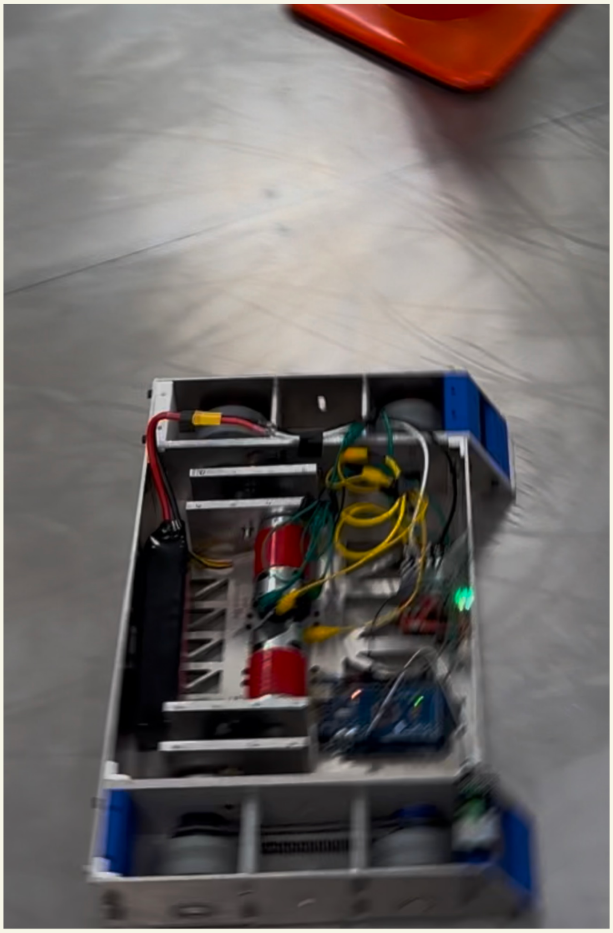
12/04/2024



Early this morning, Paul and Sara helped me pin the gears. Things went well and I'm confident in the placement of the pins. We then worked on assembly, my primary focus was finalizing the electronics and securing them for robotic movement. Pictured is the finalized assembly (same as the enforcer schematic). I properly seated in the electronics after all the mechanical assembly (both shafts of pinned gears fit exceptionally). It was then time to put the robot on the track and test. At first, it barely moved. The wheels couldn't drive the robot. I found the error was in duty-cycling the motors for free spin, and I needed to duty-cycle for the high torque now. I found the duty-cycle ratio during the CDR, and used that value. With more power going to the motors, the robot now drives excellently.

On this page is the results of the first test the day before the mobility demo.

Video of our first driving test:
https://youtube.com/shorts/au-thv-2QbY?si=XKAAxYKBKvWoJ-P6



Robot in movement

Quick Notes of Improvement:

While arcade drive works, it can be hard to control the full robotic locomotion with just one stick. I will try and switch to a form of two-joystick drive that emulates swerve drive. Left stick= turning left and right
Right stick = forward and back. This is more an ease of use type of thing, not a big priority, but I would like to try and get to it after the mobility demo sometime. Meanwhile, the bellypan pockets aren't deep enough to prevent the magnets from stick out of the bottom of the frame. So, we need to make these deeper. For this test and the mobility demo, we will work without the magnets. Finally, having proper electronic mounts beyond tape would be nice. I could try and work on this next week. But again, this isn't a big need.

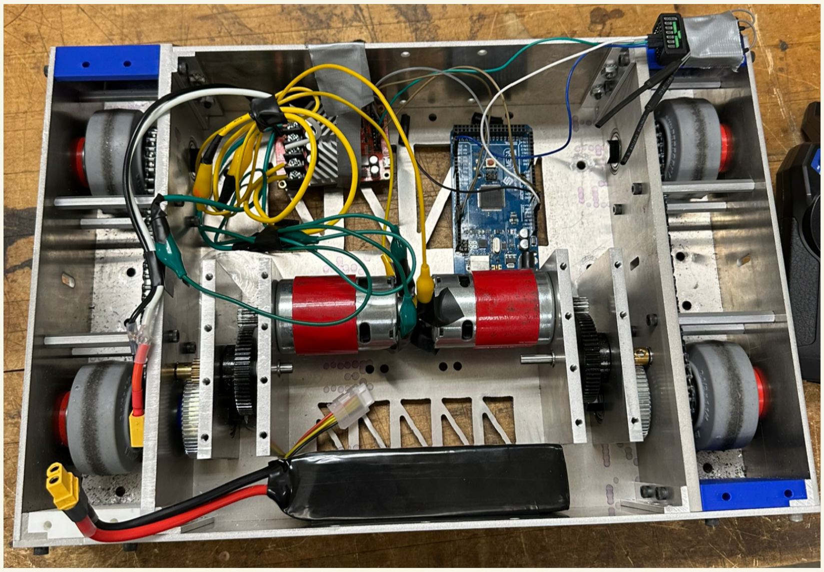
Also, we will still be adding a top plate and better wiring attachments. That being said, we are well under weight at 13.2 pounds. We should succeed in the mobility demo!

JM

12/05/2024

Today was the mobility demo

We did quite well, I was very happy with our performance. Jimmy's driving was solid and will only get better. I think we are definitely one of the best robots insofar accounting for robust design, driving, programming, and security of electronics as well as puck manipulation.



I took several videos of our testing. Below are the best peformances we had for each challenge.

Test 1: Simple Loop
https://youtube.com/shorts/DCp2Qy-slwQ?si=QT0tplAJ4qUHLgNW

Test 2: Slalom
https://youtube.com/shorts/QYDFDDO1H1k?si=aAvZC1PiAcTS4N7u

Submitted first term BOM

Test 3: Wall tracing
https://youtube.com/shorts/C4xSs5jYKZo?si=21HPL06QOhvkHCPx

Test 4: Wall tracing + puck handling/shooting
https://youtube.com/shorts/WBguBtaI7pU?si=Rg0RvHHhmRtzAkiK

I have also organized the team into final tasks before the end of the year:

sara: cut handles into outer side plates for grabbing robot
jimmy: top plate out of polycarbonate or plexiglass
jarbi and I: 3d print mounts for roboclaw, arduino, battery, and receiver
jaylen: get magnets in place and make sure they don’t affect the chains
me: implement double stick drive and overdrive motor control

JM

Today I implemented double stick drive and also a simple gearing system. The double stick drive allows us to control turning with the left joystick and forward/back with purely the right joystick. The reason behind this design decision is that it makes precise control of the robot easier and more intuitive than having everything on one stick. The three stage gearing system has a low gear, high gear, and medium gear.

High Gear: Maximum speed and torque. Best for crossing the field quickly, ramming other bots out of the way, and escaping quickly with the puck. It has full duty cycling, meaning it runs at the maximum operating voltage of the motors.

Medium Gear: Best for typical gameplay as it mixes good speed and power with efficient battery usage.

Low Gear: Best for precise movements, especially ones with turning. Allows for easier puck manipulation and slowing down quickly.

I had to switch and reprogram our flysky controllers as the one we had didn't have full usage of all 10 channels. Now we have this full usage, and this allows for many more user operations on the robot. I have programmed the switches for killing, shifting, and shooting so far.

Video Demonstration: https://youtube.com/shorts/SxoisqLDBcY?si=MA5R5nHccTA8adk_

My first task being done, I will work on the mounts with Jabri next week. The next steps are to submit my first term notebook and have a tea meeting Sunday to discuss the final report.

JM