StuyPulse

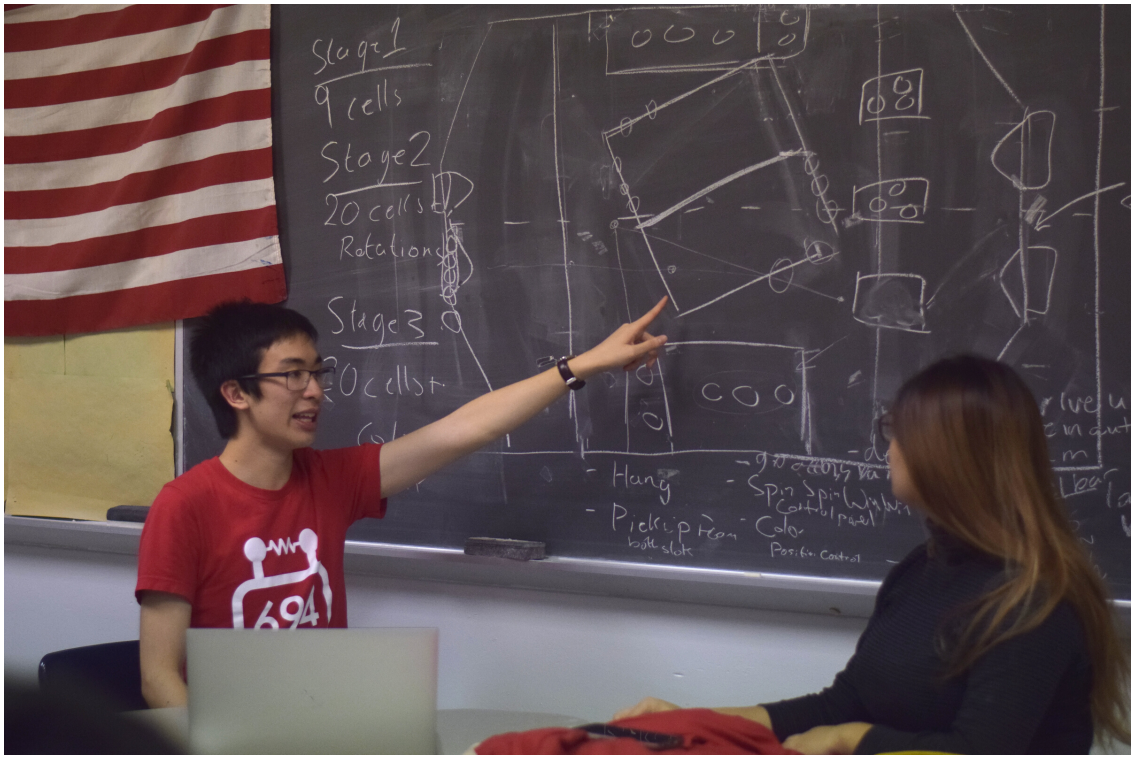
EDWIN

2020 Technical Binder

Table of Contents

STRATEGIC ANALYSIS

Initial Game Analysis



Our game analysis process begins with a team manual reading immediately following the kickoff broadcast. This was followed by small group discussions, which merged into a team discussion brainstorming types of robots and alliances we would see at competitions, differentiating robots by all of their possible tasks, outlining potential defensive strategies and cycles, and isolating winning strategies.

Strategic Decisions

AUTONOMOUS

We knew from the outset that each of our routines would be shooting into the high goal exclusively, and that we would always utilize the 3 preloads. We prioritized developing our first 3 routines by our first competition, while we aimed to finish the next 2 by championships, and included the final 2 with the goal of compatibility with other high scoring robots during championships eliminations.

NEEDS:

- Mobility
- Score 3: We initially expected scoring our first three preloads to be more consistent from the initiation line, but we eventually began scoring them from the trench for the sake of better efficiency between

cycles and a better chance of scoring in the inner port.

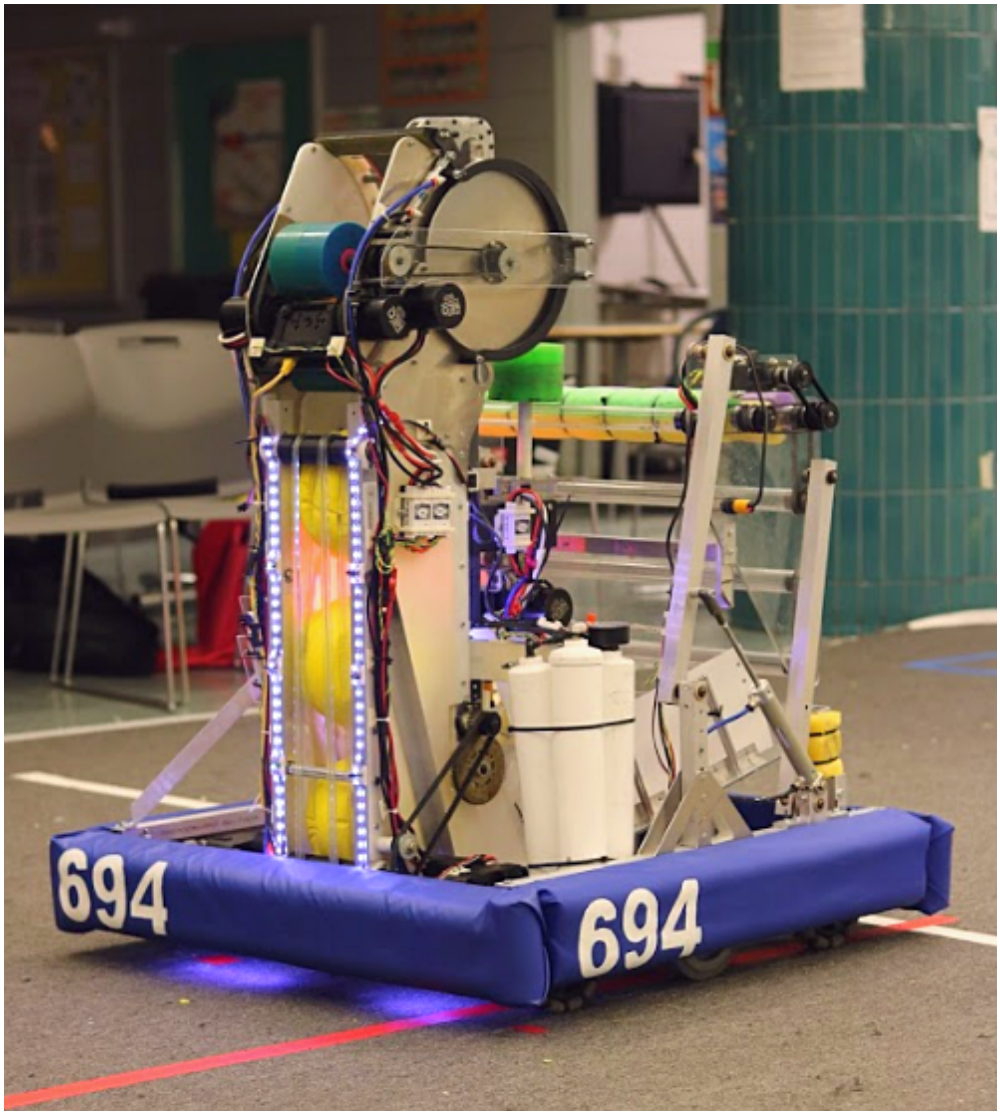
- Score 6 (3 from the trench): This autonomous routine was what allowed us to win our first competition. As the balls lost firmness throughout the competition, we had to begin shooting in the first few seconds of tele-op to better align.

BY CHAMPIONSHIPS:

- Score 8 (3 from trench and 2 from rendezvous): This routine was in development at the close of the season, and its shortest time running was 21 seconds.
- Score 5 (2 from opposite trench): We anticipated this autonomous routine to be very valuable at championships, when robots could “steal” 2 balls from the opposing trench faster than their opposing alliance could pick up more than 3 from their own.

NICE FOR CHAMPIONSHIPS ELIMINATIONS ROUNDS:

- Score 6 (3 from rendezvous): We prioritized this routine to be compatible with partners who scored 6 power cells from the trench during autonomous time.
- Score 8 (5 from rendezvous)



TELEOPERATED

Our major strategic takeaway regarding tele-op was that our primary priority for the 2020 season would be becoming the best possible high goal scorer. We also concluded that we wouldn't build a robot that could pass under the trench, or one that scored from many places around the field (i.e. a turret) because we would jeopardize our primary goal, considering our limited resources.

NEEDS:

- Scoring in the outer goal: We chose maximizing accuracy and speed rather than scoring locations.
- Rotational and positional control: We expected use of the control panel to be critical for the "energized" ranking point.

WANTS:

- Scoring in the inner goal: We expected to

work towards this goal all season. We always aimed for the inner goal and generally attempted to improve accuracy, but we mostly scored outer.

DON'T NEEDS:

- Low goal scoring: The low points earned by scoring in the low goal, and the cycle time wasted in driving to and from power cells, made this a quick decision for us. Ironically, we did end up being able to score in the low goal by pressing balls between our acquirer and the power port, and then running it backwards.
- Scoring in the outer goal from many locations

END GAME

After high-goal related tasks, climbing efficiently was our second highest priority.

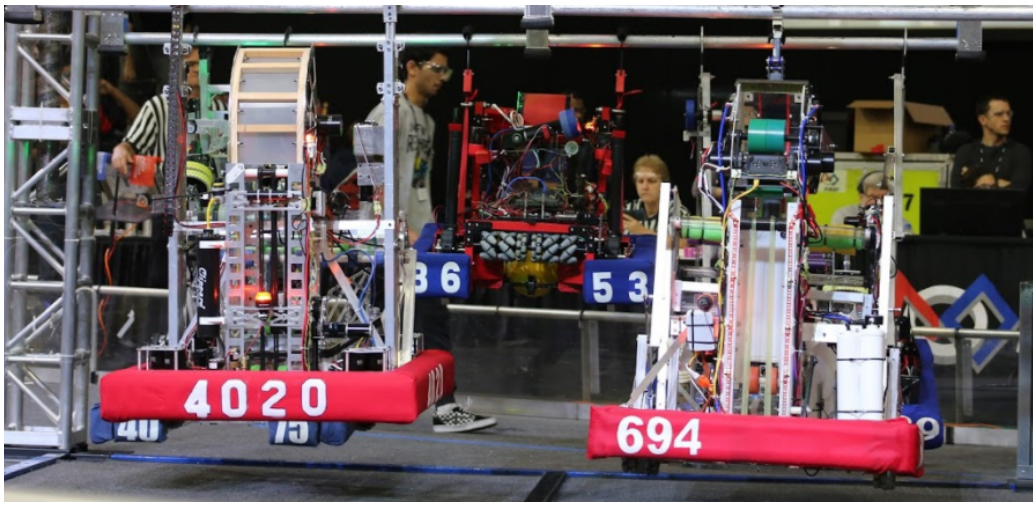
NEEDS:

- Climb solo
- Climbing from any position on the rung
- Not sliding along the rung
- Climb quickly
- Remaining stable during and after climbing

EXPLORE:

- Climbing at any height of the rung: We pursued a two-stage telescoping arm climber in order to get the necessary height for climbing flexibility. Though original designs permitted us to climb at any height, later construction and integration errors limited our maximum height, and the climber ultimately could reach approximately 5/6ths of the tilted generator switch.

- Climb assist: After the prototyping period, we decided to not pursue a buddy climb because it would significantly limit robot design. We also predicted that such a mechanism would have diminishing returns as the season progressed.



- Generator Switch Balancing: We built a “yo-yo” (a mechanism that would allow the robot to move along the generator switch after it climbed), but we initially decided not to put it on the robot because it didn’t integrate well with our climber, which was working consistently at our first competition. We also predicted that it would have diminishing returns as teams got better at climbing with a partner.

Priority List

After compiling our individual priority lists and considering our wants, needs, and nice-to-haves, we concluded with this priority list.

1

Drive

2

Acquire power cells from the ground

3

Score 1 power cell into the inner goal

4

Score 5 power cells into the inner goal

5

Solo climb and balance

6

Climb on one of the two sides of the generator switch rung

7

Score 3 power cells into the inner goal in autonomous

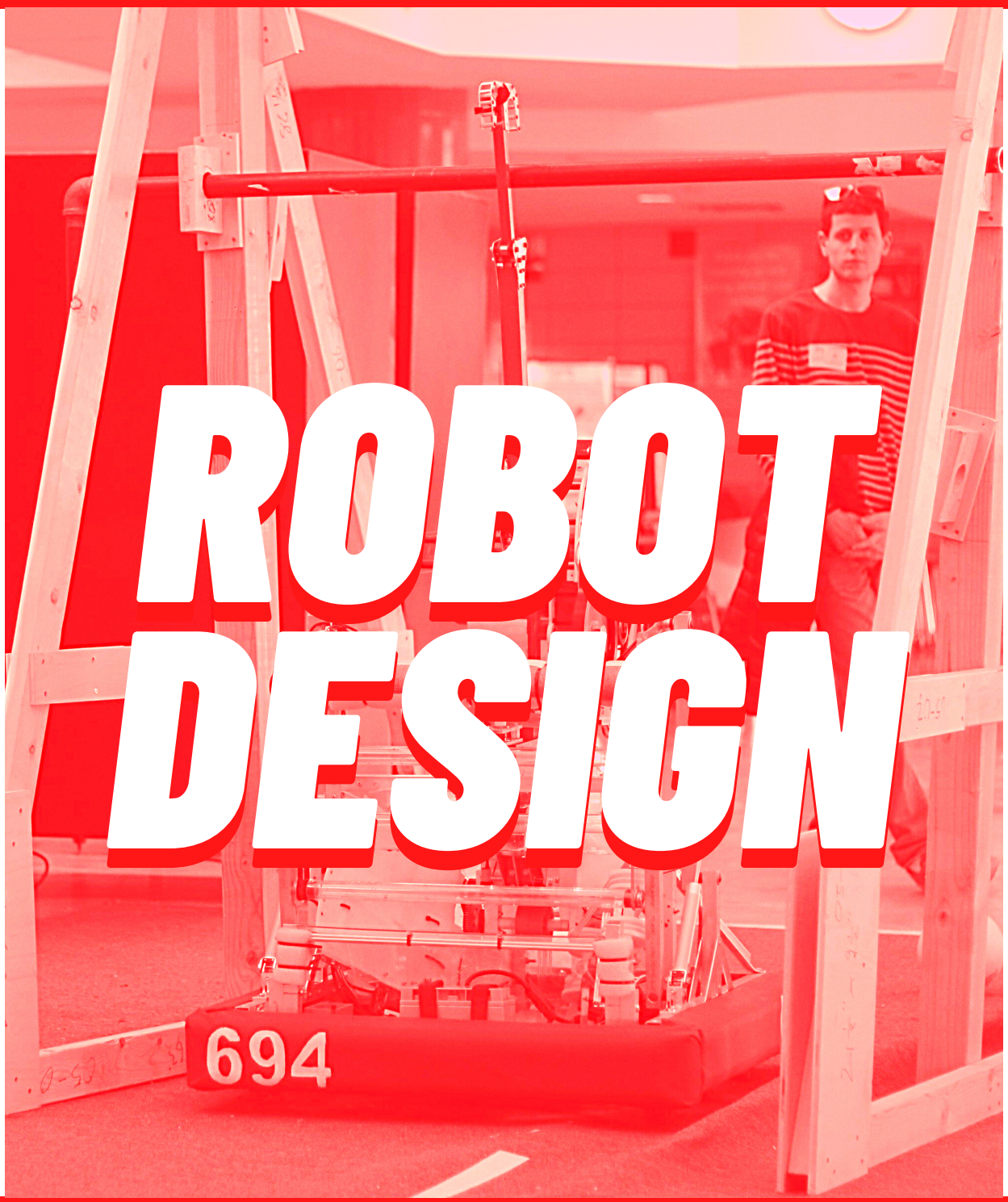
8

Score 6 power cells into the inner goal in autonomous

9

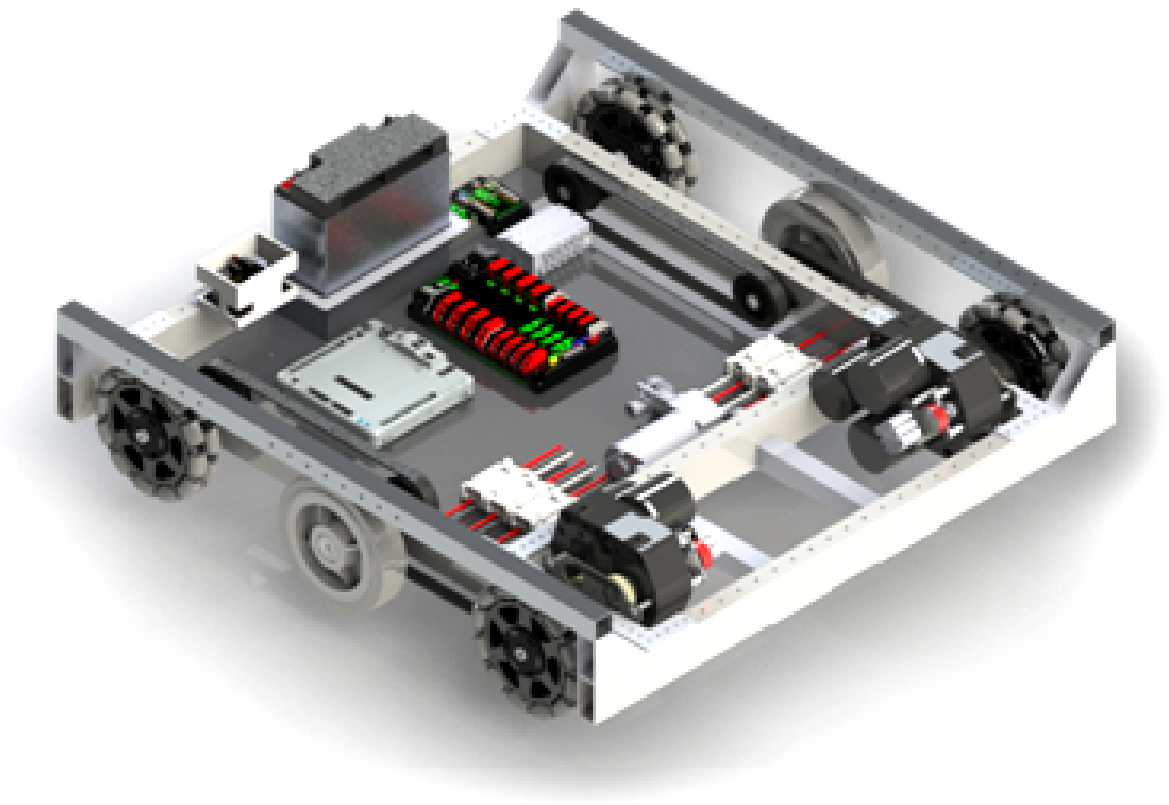
Rotation control and position control

In reflection, our large picture goals of prioritizing power cells in the high goal followed by climbing remained consistent throughout the build season and competition season. In retrospect, our prioritization of the inner goal over-estimated the specificity we would be able to achieve, but we knew that working towards it would also lead towards more accurate outer goal shooting. Though we never got to see it play out later in the season, we found rotation and position control surprisingly under-used, but we saw efficient point returns in the few times it was utilized.



ROBOT DESIGN

Drivetrain

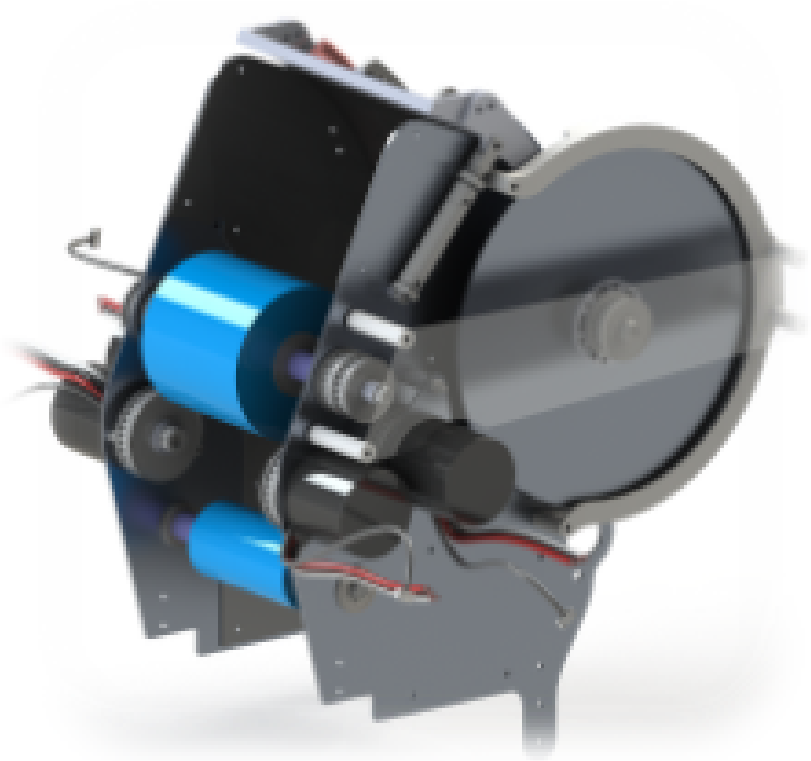


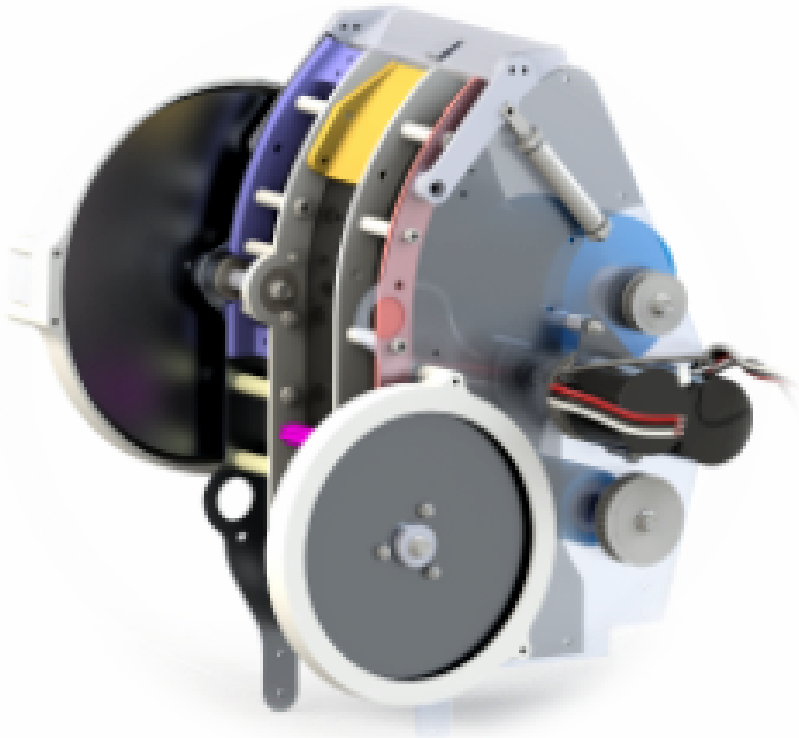
The drivetrain is a mechanism by which our robot moves across the field. The bars encircling the rendezvous zone posed a challenge this year, as going over those boundaries at high speeds would put the drivetrain under great stress. Furthermore, we sought to minimize beaching and crushing balls in order to maximize cycle times.

SPECS

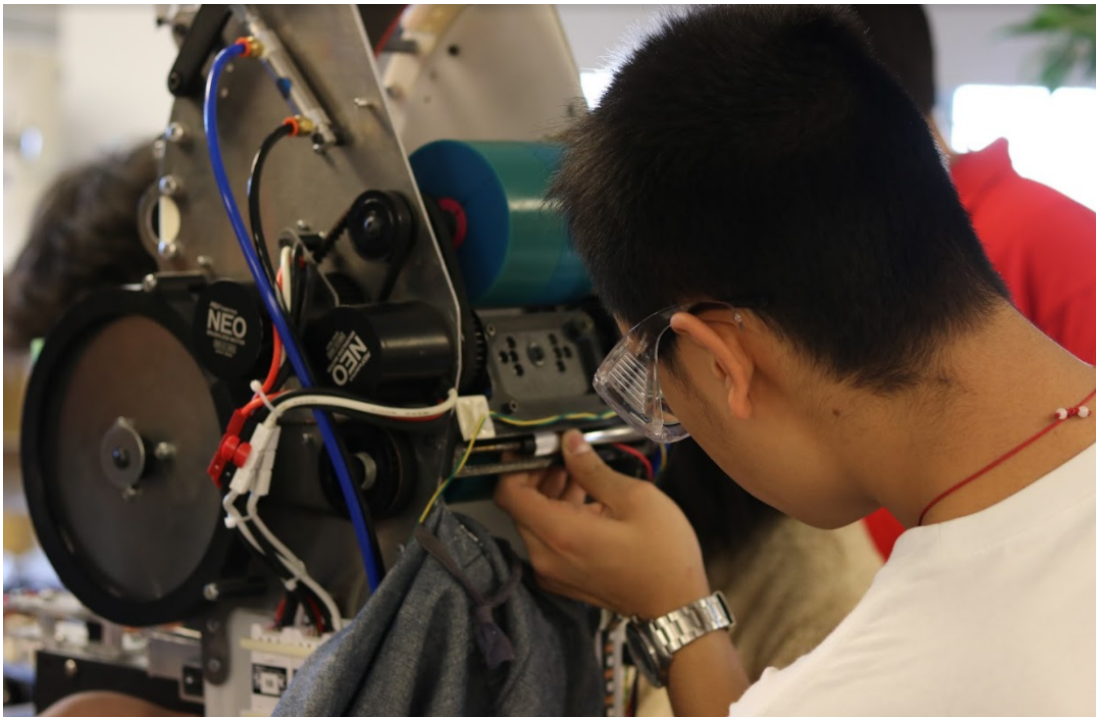
- 30” x 29” (long) frame, 6-wheel tank drive
  - 6-inch omni-wheels used on front and rear wheels, 6-inch colson wheels used on central wheels
    - Allowed for easy turning about the center
    - Mechanically familiar and reliable
- Driven by two VEX 3 CIM Ball Shifters
  - Each gearbox equipped with 3 NEOs
  - High gear: 16.14 ft/s
  - Low gear: 7.46 ft/s
- Aluminum rails
  - Much of the superstructure is mounted and secured to these rails, such as the intake
  - Its mounting to the chassis is sturdy, making it ideal for attaching supports for vital mechanisms

Shooter





The shooter is a mechanism with the goal of delivering the power cell to the outer and inner goal on the field. We wanted to be able to shoot accurately from both the trench run and from the starting line. To achieve this, we wanted the arc of the ball to be as flat as possible to maximize the range of confidence for getting the ball into the inner port. Based on kinematics calculations (done using desmos), we needed to have a piston-actuated hood to switch between two shooting angles. Two sets of rollers, one set to bring the ball to half its shooting velocity and one to bring it to full speed, worked to reduce variations



between shots, as the feeding roller brought the ball to a consistent speed before being shot out by the second set of rollers. Additionally, flywheels are used to keep the rollers running at their intended speed throughout the match and as power cells go through. These flywheels are designed to be as big as possible to maximize stored inertia while reducing added mass. This machine is powered by 4 NEO motors which drive the rollers and flywheels using belts and pulleys. 4 motors ensured we could bring the shooter to speed in less than a second while current limited to 40A and reduced RPM variation while shooting. Polyethylene was added on the sides of the shooter for the power cells to slide on so they come out more consistently.

SPECS

- Wheels
  - Shooter wheels: 2, 4-inch Fairlane 60A Urethane Rollers with 3D printed hubs

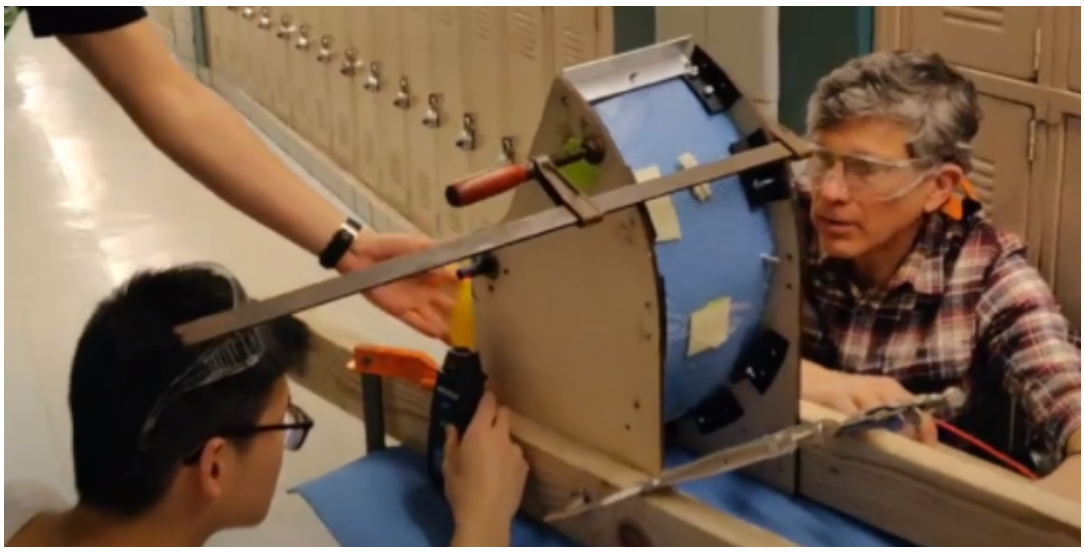
    - Feeder wheels: 2, 2-inch Fairlane 60A Urethane Rollers with 3D printed hex hubs
- Flywheels
    - Flywheel for Feeder wheels: 2.54 lb-in^2
    - Flywheel for Shooter wheels: 6.94 lb-in^2
- Power
    - 4 NEO motors
        - 3 drive the shooter wheels
        - 1 drives the feeder wheel
    - Pulleys connect the motors to the shooter wheels with a 1:1.5 speed up
- Main Hood
    - 1/16 in Lexan supported by 3D-printed rails
- Adjustable Hood
    - Two 1-in stroke, 7/16 bore pistons
    - 0.04in thick lexan attached to a 3D printed part
- Construction
    - 0.1 in Aluminum walls
    - 0.06in polyethylene

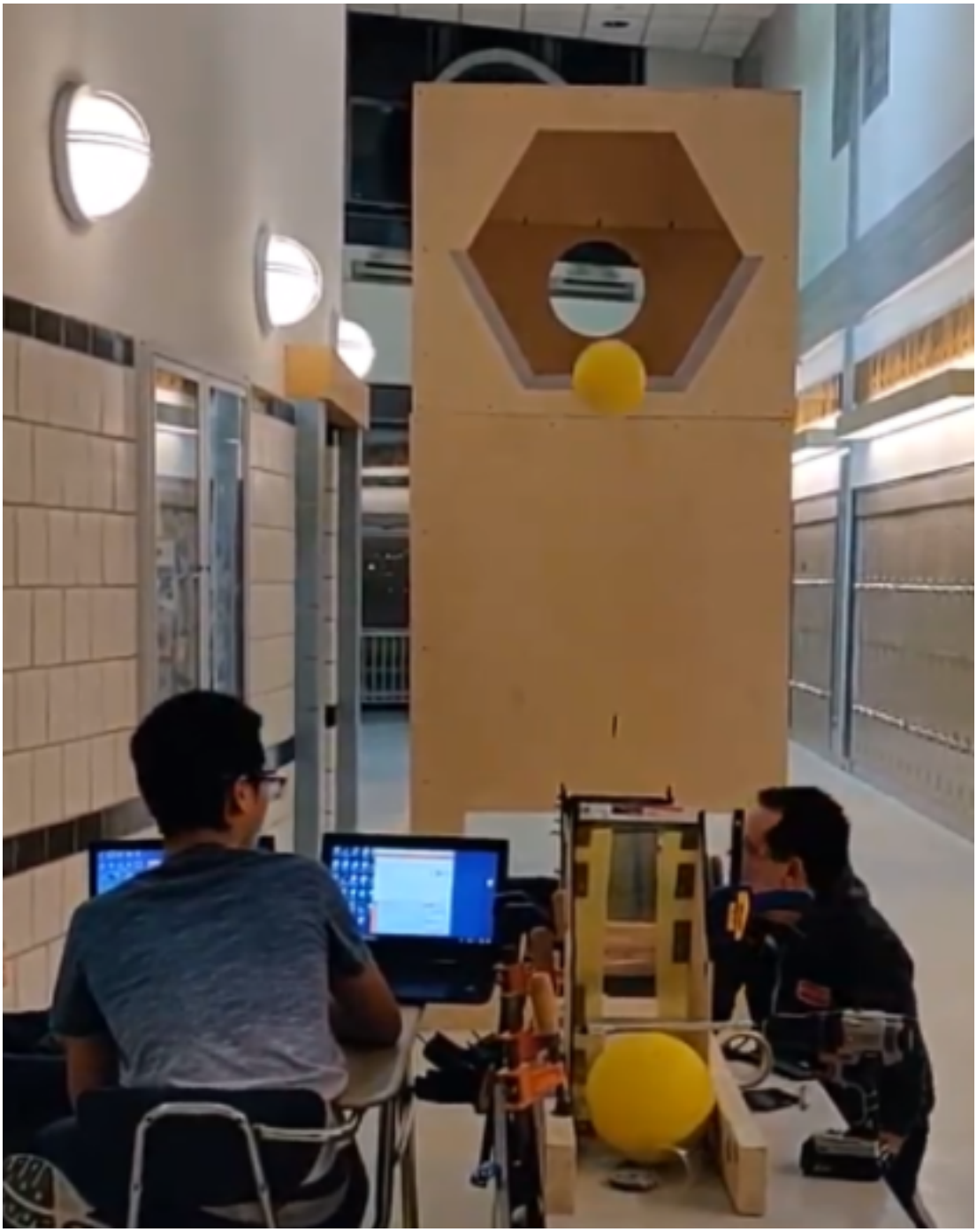
MODES:

- Starting Line Shot
  - 3000RPM, 8 (m/s)
  - Adjustable Hood Up
- Trench Line Shot
  - 4500RPM 12 (m/s)
  - Adjustable Hood Down

DEVELOPMENT

We began development by building a NEO powered wooden prototype shooter. It was constructed from laser cut wood. Different amounts of compression on the ball could be tested by adding layers of yoga mat pieces onto the hood.



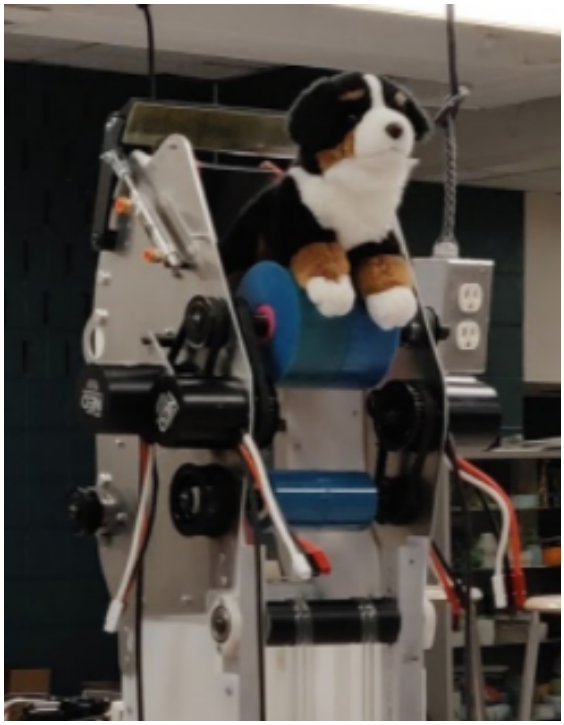


A tachometer was useful to validating our speeds, but since the prototype was powered using a NEO motor, we ran the motor using the SPARK MAX software on the computer, allowing us to monitor the changing speeds.

Later iterations to the prototype allowed us to change between several preset angles. For our tests, we focused on getting the most accurate possible shots at the initiation line and the trench run.

Remarkably, the shooter has not failed mechanically, but there remains work to be done to improve its accuracy. An idea we had before the pandemic was to add additional slippery polyethylene on the sides to compress the ball to reduce left-right variation. We also wanted to make changes to the PID loop.

NOTE: NYC regional's lead robot inspector Noah Tom Wong told us our flywheel was a "spinning wheel of death", so a cover was added during week 6.



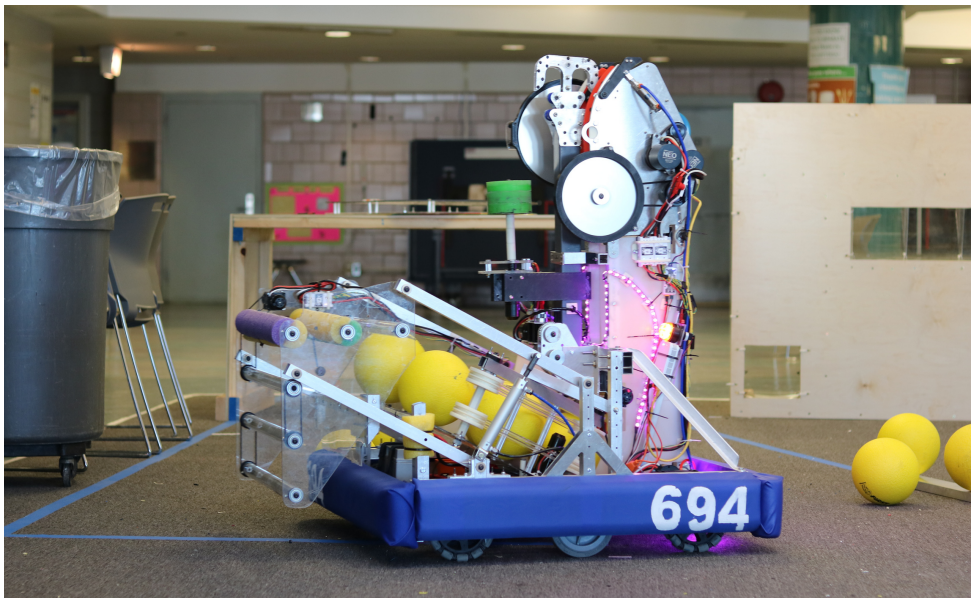
Intake

This mechanism picks up power cells from the ground and human player station and then feeds them into our indexer mechanism. It consists of a series of rollers that move the power cell against polycarbonate plates in order to bring balls from the ground into the indexer. It is deployed using a four-bar linkage (actuated by a pneumatic cylinder).

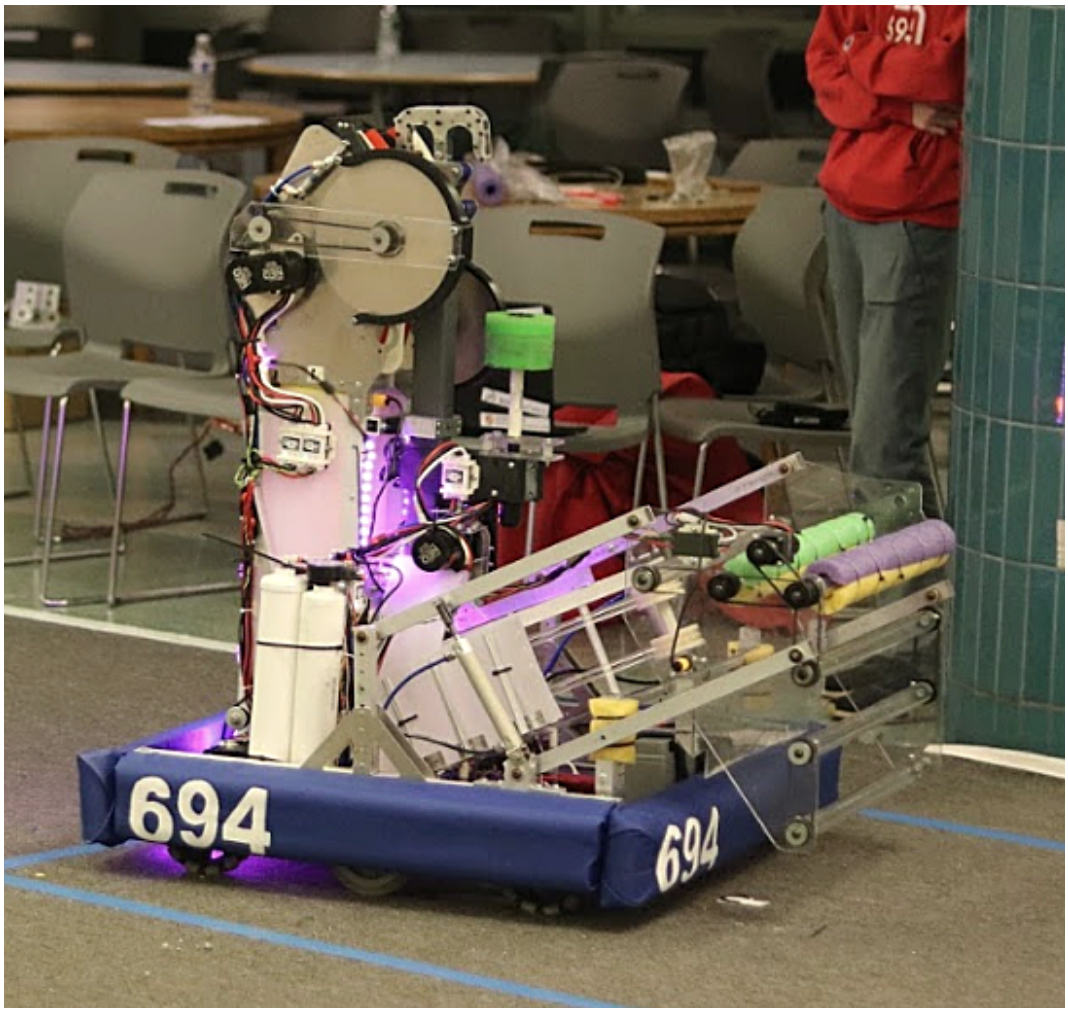
Major design constraints include the height of the indexer mechanism, which resulted in a large, tall intake that moved balls along a 90-degree turn in order to bring them to the required height. The height of the indexer and weight concerns also led to the decision to use a four-bar linkage, rather than a pivot or arm, because the size of the intake meant that a small, pneumatic pivot simply wasn't possible, and a large, motor-driven arm would likely have been too heavy. A second (currently unfinished) version of this mechanism incorporates a ramp to easily load balls from the human player station.

SPECS

- Driven by NEO on a 2:1 reduction for speed of approximately 16ft/sec
  - Uses 24t and 48t GT2 3mm pulleys for main reduction
  - Rollers driven together using 36t GT2 3mm pulleys
  - Initially driven using NEO 550 on a 9:1 reduction with a VersaPlanetary gearbox
    - Slower than current gearing, had issues with running over balls while intaking, so swapped to the current setup



- Uses combination of 1.25in and 2in OD round polycarbonate tubing as rollers
    - Larger rollers on “corner” to eliminate dead zones
        - Initially used pool noodles ziptied onto rollers, (unfinished) Mk2 uses larger diameter rollers instead
    - Polycarbonate chosen because lightweight, durable, and sufficiently grippy on balls
- Approximately 1.2in of compression throughout
    - In combination with intake speed, allows fast, touch-it-own-it pickup of balls anywhere on the field, including the rendezvous zone barriers
- Incorporates ramp for human player station loading
    - Ramp height just under height of human player station
    - Initially dropped power cells onto pool noodle rollers while reversing intake to load from human player station



- Deployed using pneumatically-actuated four-bar linkage
  - Utilizes 2 impact-resistant cylinders with a ¾” bore size and 4” stroke
  - Uses steel clevises for pivoting
- Approximately 23in wide
  - Spans most of the robot’s width
  - Allows pickup of up to three power cells at once

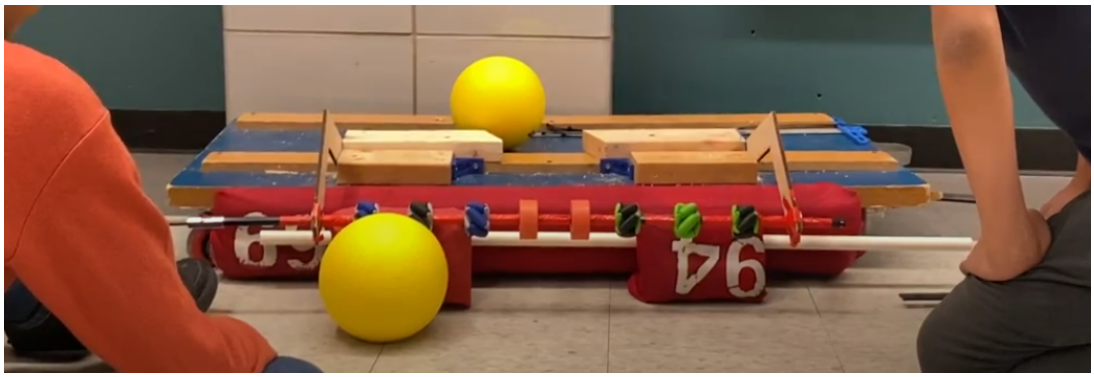
DEVELOPMENT

INITIAL THOUGHTS

The moment that balls were revealed to be the game piece, we knew we wanted an over-the-bumper top roller of some sort. This would maximize the available area for picking up balls, allowing us to run faster, more efficient cycles and making it easier on the driver than horizontal rollers. We also knew we wanted this mechanism to be as “touch-it-own-it” as possible in order to further decrease cycle time, especially after the complications of previous seasons.

PROTOTYPING

Because of our initial decision on having a top roller, we ended up prototyping three main mechanisms: a mecanum top roller, a polycord top roller, and a 973 2012-style polycord intake, using designs we were familiar with from previous games involving balls. Additional designs considered include a



125 2017-style roller mechanism and an early compliant wheel top roller, similar to those commonly seen in 2019.

The 125 2017 intake was quickly discarded due to numerous issues during prototyping, including being out of frame perimeter when retracted, a lot of compression, and limited size due to requiring a bumper cutout, not fitting our initial criteria of maximizing intake area. The compliant wheel roller was discarded as well because our indexer prototyping determined that it needed to move the ball from the ground to a height that was much greater than what it was capable of.

After discarding those mechanisms, we moved to decide on which one would ultimately end up on the robot. The debate

essentially boiled down to whether we wanted to index power cells inside or outside the robot. During testing, we found that both the mecanum and 973 2012 rollers were very conducive to jamming, with both mechanisms jamming when picking up multiple balls, while our polycord top roller and “powered V” indexer prototypes had no such issues. As a result, we chose to index power cells inside of the robot and with that, the polycord top roller and “powered V” indexer combo.

BUILD SEASON

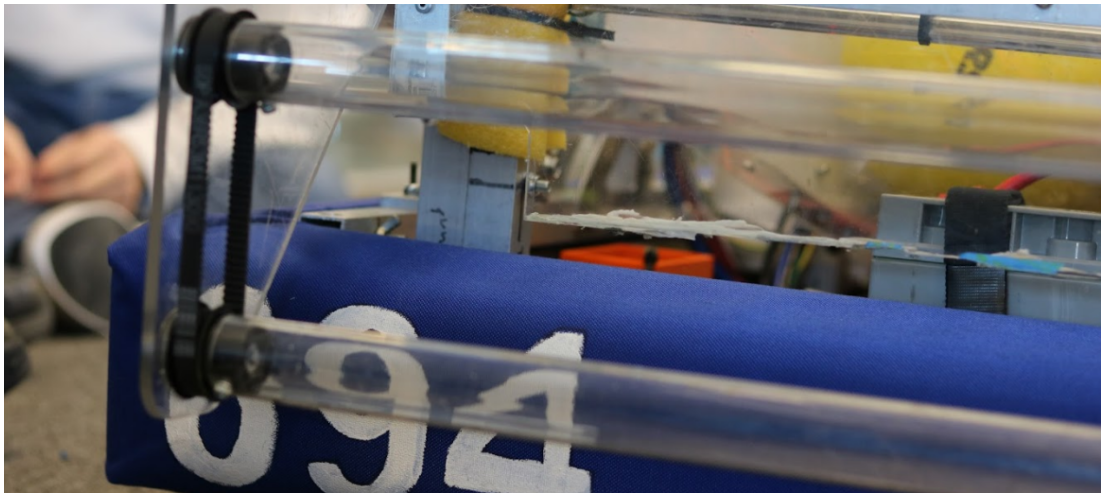
Once we settled on an intake design, implementation began. Some immediate concerns were durability and the deployment of the intake. In order to keep the intake intact when deployed, we could either design it to be nearly indestructible or have it flex when hit. In the end, we chose flexible polycarbonate because it would be much lighter. An additional point of contention was how we would deploy the intake. A smaller, pneumatically-actuated pivot would likely have

Once we had the design finalized and started building, we realized that the polycord had a silly amount of tension and was causing the entire intake to flex. We tried putting less polycord on the intake, decreasing the amount it was tensioned, and adding aluminum crossbars for support, but none of that stopped the flexing. In addition, the decrease in tension meant that the polycord was too loose and as a result couldn't properly drive the rollers. So, we redesigned the intake to be driven using timing belts and pulleys, and added more rollers to account for the lack of polycord. The intake also incorporated 1/16in wall polycarbonate tubing instead of the previous 3D printed rollers in order to save weight. This eliminated the flexing issue, with the intake being sufficiently rigid while still having enough give in order to properly take hits. The new rollers also had no significant differences in performance from the polycord that we tested during prototyping, and may have actually performed better because there were no potential slips.

There were also a few other minor changes made to the original design. The most notable one being the addition of pool noodles to the top of the intake. When testing, we found that there was a dead zone as the power cells turned the corner of the intake, and as a result added pool noodles to increase the outer diameter of the problematic rollers to eliminate the dead zone. Additionally, we found that one of the back plates in the original design was unnecessary, so we removed it.

INTEGRATION ISSUES

Ultimately, integration wasn't a very significant issue. Due to some pretty major oversights in CAD, several parts of the intake ended up colliding with other parts of our robot. For example, the intake plates collided with the indexer supports and the indexer pulleys, the back plate of the intake hit the main indexer plate, and when retracted, the intake rollers hit our Wheel of Fortune mechanism. Fortunately, all of these issues were easy fixes.

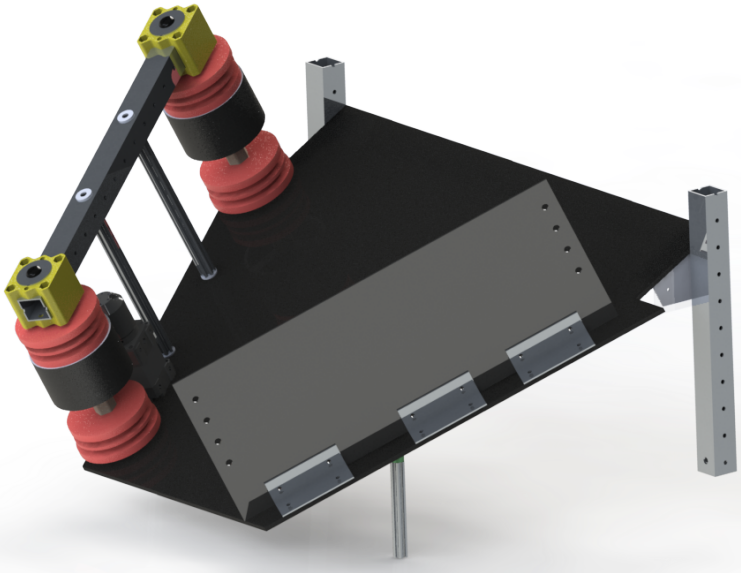


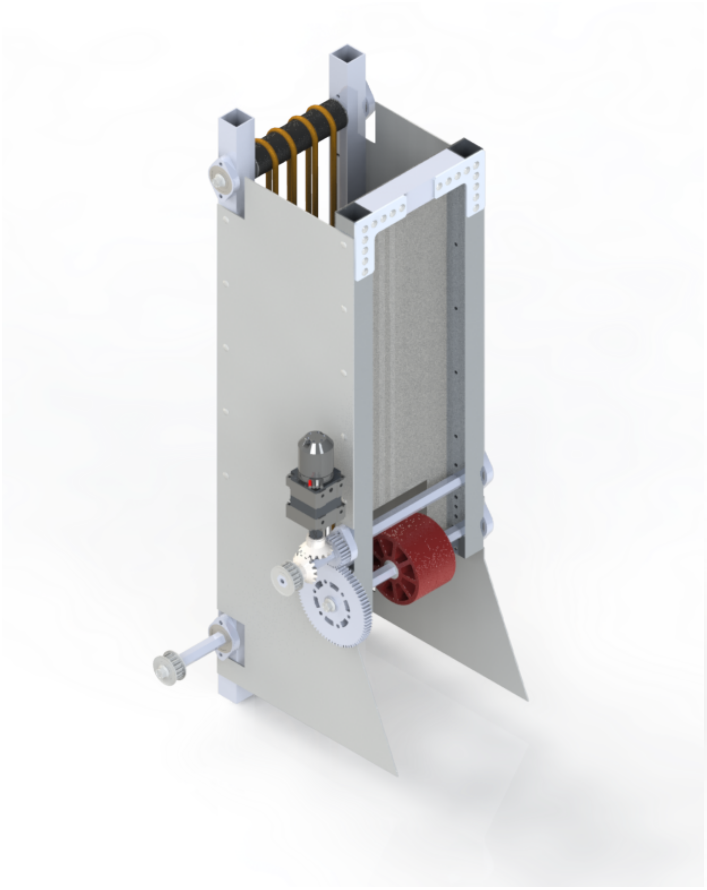
PERFORMANCE

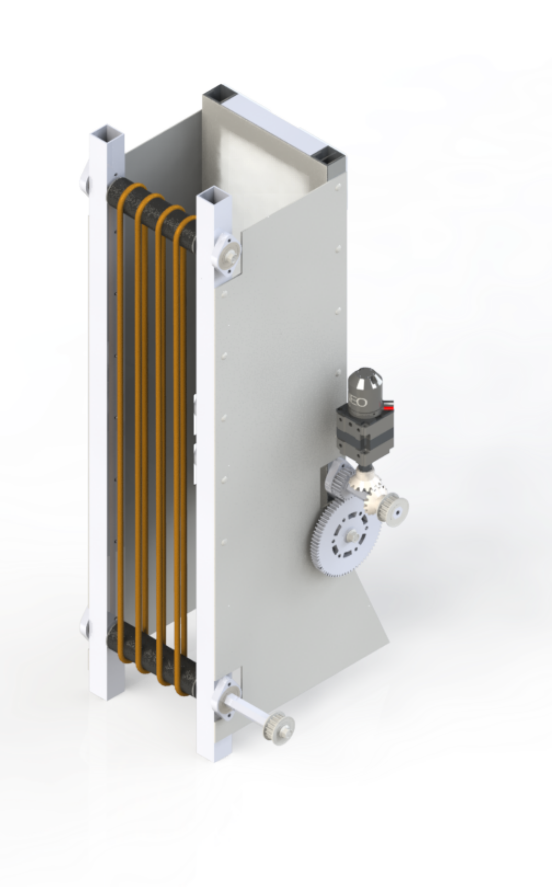
Overall, the intake performed very well. However, we did have problems with the intake being too slow, which, in combination with other issues, meant we ended up driving over power cells a lot. In addition, we had a few more minor problems. For example, the belts occasionally slipped due to inconsistent tension from the flexible polycarbonate plates.

Intake Mk2 incorporated a 2:1 pulley reduction for greater speed and used exact center-to-center distances to prevent over-tensioned belts from grinding. It also used a ramp to load from the human player station and a larger polycarbonate roller to eliminate the dead zone.

Funnel & Chute







The funnel and the chute are feeding mechanisms that guide the power cells from the intake to the shooter. The funnel intakes a maximum of 5 power cells on a flat plain with a wide opening that narrows down to the width of the chute. It uses poly cord pulleys to move the balls towards the narrow end. After passing through the narrow opening, the power cells are caught by the compliant wheel on the bottom of the chute. Then, the ball is pulled upward by poly cord rollers where it is finally transferred to the shooter.

A key detail to maintain consistency is to conform to the dimensions of the ball. The chute was purposely made to be as wide as the ball on one axis and then smaller than the ball in another so that it could compress the balls and maintain a friction-driven grip.The funnel’s integration to the chute was difficult, but in the end, we fastened PE plates to block the gaps where balls would jam. This is not pictured in the CAD, as the plates’ bending made the shapes highly organic.

SPECS

FUNNEL

- Driven by NEO 550 with 36:1 reduction with VersaPlanetary gearbox
    - The pulley attachment on the neo had teeth that were 3mm to maintain consistency with the intake
        - Is more efficient with larger teeth because larger teeth skip less on loose pulleys
- 3D-printed polycord pulleys

CHUTE

- Driven by NEO with a 35:1 reduction with VersaPlanetary gearbox
- Constructed with polyethylene walls for maximum sliding
- 3D-printed polycord rollers
- Sturdily fastened to chassis
    - Allowed for other mechanisms to be positioned on its 1 in. x 1 in. channel skeleton

DEVELOPMENT

The first iteration of our prototype was a platform with flat poly cord and two planks of hardwood that angled from a wide entryway to narrow exit.

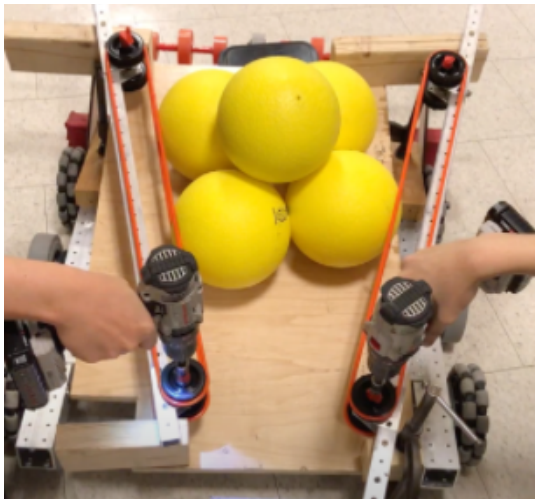
This iteration was inefficient because the balls kept jamming into each other due to the contrasting forces of friction, like if you were to turn two adjacent gears in the same direction. Furthermore, the polycord kept moving towards the looser tension on a specific part of the PVC rollers.

We had a second iteration of the same polycord platform but no angled side planks. Instead, we put one of the planks in front of where the area the balls were supposed to roll towards to see if they would climb up the wall.

The balls ended up just slipping off to the sides because there was not enough tension to roll them up the wall. It would have been better if it was an enclosed space so that there was greater grip, but because the balls were only being moved by one roller on the bottom, it was not very effective. It may have also worked if the polycord platform was on top of the balls so the roller on top could roll the balls up the side plank. The bottom platform would still have to allow the balls to move without catching.

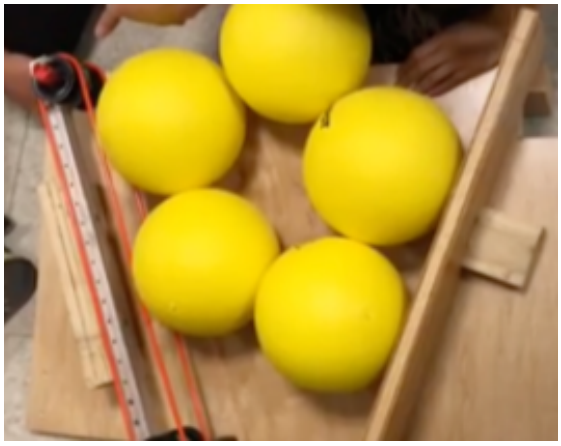
A video found online from another team led to our third iteration. We tweaked the prototype and added an angled wooden, slippery platform and two angled, tubular polycord rollers on the sides.This iteration was quite effective because gravity moved the balls

downwards and kept them from randomly bouncing off like in the first iteration. The polycord on the sides also fed the balls consistently into the into the narrower opening. One roller rolled the balls down at high speed and the other roller rolled the balls up at a lower speed. This kept the balls from jamming. Even with the improvements made by this design and the consistency it initially provided, the polycord eventually started slipping off. This may have been due to the fact that this iteration put a lot of strain on the polycord and caused them to stretch out. It may also have been due to the flexing of the shafts driving the pulleys.



We made another partial iteration that formed the beginnings of the chute.This transported the fuel vertically. It didn't cause

any jams and seemed reliable due to its rigidity. Only one ball could go through this at once.



We wanted to test out whether a plank of wood would be a sufficient secondary “wall” for the balls to be contained within. This turned out to be better than the two-roller configuration. The power cells were transported best on one moving surface and one slippery surface. Jam rate was very low for this mechanism.

In this iteration, we constructed the chute. There were many configurations for this because we needed to test many different ways to position the wheels. Initially, we use two compliant wheels in the front and one in the

There was many a “snap, pop, and crackle” with this one. It jammed frequently; surprisingly, no one went insane.The main issue with this configuration was the fact that the radii of compliant wheels were too small and when the balls pushed against the wheels, they would jam into the 1 in. x 1 in. channel. With some trial and error, we found a position where these two parts integrated well. The compliant wheels would be low enough to come in contact with the power cells but high enough so that the power cells rarely touched the 1 by 1 (which would cause a jam).

WOOF

A.K.A. Wheel of (of) Fortune, control panel mechanism

The Woof, designed to spin the control panel field element at approximately 60 rpm, consists of a vertically-mounted gearbox running a shaft with three compliant wheels. We attempted to create a mechanism that was light but highly effective.

SPECS

- NEO 550 with a 20:1 reduction with VersaPlanetary Gearbox
- 3D-printed mount
    - Clamps onto the climber

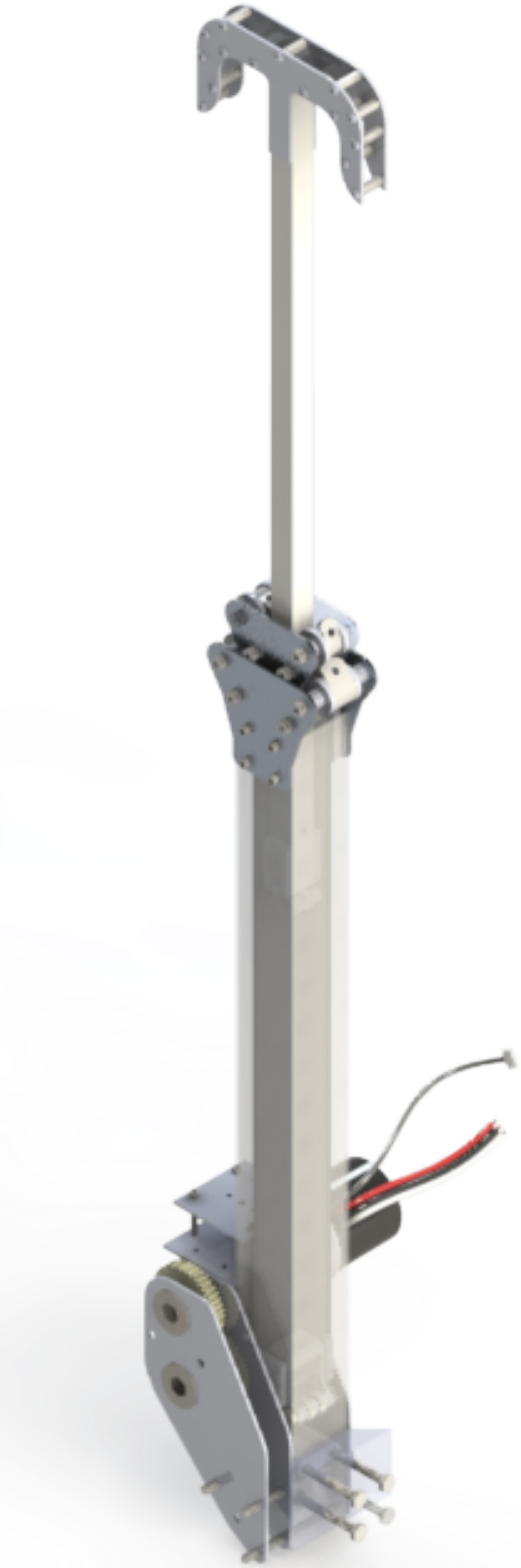
DEVELOPMENT

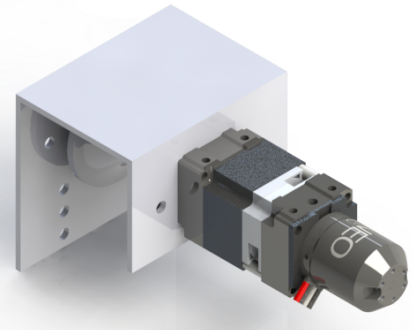
A previous iteration had two sets of wheels that allowed our robot to contact the control panel at two points, which would help with aligning.

Using the combination of 4:1 and 5:1 planetary gear sets, we geared down a NEO 550 motor to spin at a desired speed. The closest reduction, to attain ~60 rpm, for 2-⅞” T81 50A BaneBot wheels was 20:1. However, we eventually chose to use 4” compliant wheels, which would equate to a 27:1 reduction (closest reduction to be under 60 rpm for a VersaPlanetary would be 25:1).

Several issues were overlooked. Mainly, a significant part of the mechanism overlapped with the control panel. The initial iteration was also rather heavy, and changing to a simpler and lighter design freed up more weight for mechanisms with greater importance (as the control panel was relatively low on our priority list). The Woof was fairly quick and consistent at the Palmetto Regional, but took significant damage from malfunctioning autons and aggressive driving.

Climber & Yo-Yo





The climber was designed to be a lightweight mechanism that would enable our 2020 robot to climb quickly and sturdily onto the generator switch during the endgame period.

The Yo-Yo, consisting of a rotating concave wheel within a U-shaped metal hook, was designed to attach

to the top of the lift and shift the robot along the rung of the generator switch.

SPECS

CLIMBER

- 3 stage telescoping PVC lift
- 2x 4.950 lb Constant Force Spring driving the middle lift stage
- 2x 2.630 lb Constant Force Spring driving the top lift
- Driven by a NEO with a 20:1 reduction with Versaplanetary Gearbox
    - Impact Resistant Pancake Piston + Versaplanetary Ratchet Gear served as a brake
        - Piston served as a pawl

YO-YO

- Attached to a Neo 550 geared down to 40:1 with 2 versaplanetary gear kits: 4:1 and 10:1
- Versaplanetary Universal Female Output Shaft Kit

DEVELOPMENT

PROTOTYPING AND DESIGN

Prototyping of the telescoping arm was initially built with a single moving inner stage and aluminum tube stock. The stages were actuated with two constant force springs looped around shoulder bolts that also bore bearings to keep the inner stage in place. The design was inspired by 2056's climber from 2016, though simplifications were made for ease of manufacturing.

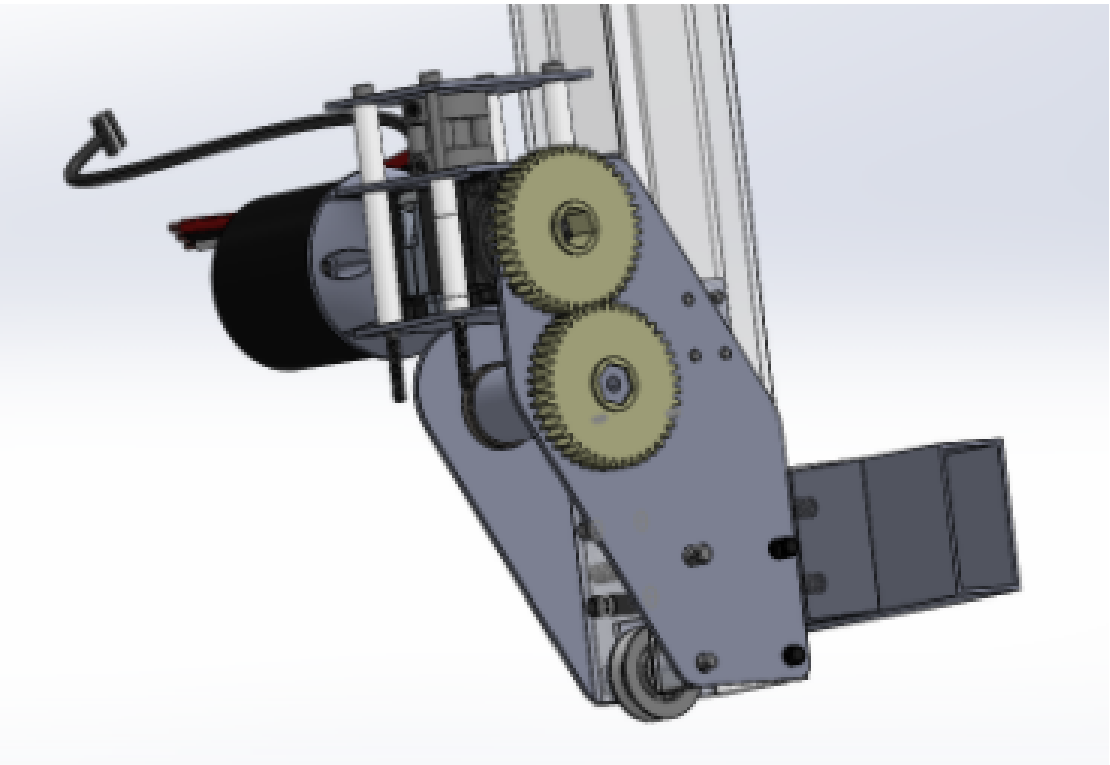
It was tested sparingly, as we did not have the proper conditions nor time to set up a more intricate testing environment, and attached the arm to a dolly and drove it with a drill.

The initial design of the Yo-Yo consisted of two sets of wheels under a long, notched, U-shaped tube, but due to structural instability and weight, this was converted to a single wheel and a slim hook such that the robot could grip the generator switch close to the central handle. The initial prototype for this design used 4” compliant wheels and bearings on an axle in a 2” by 4” channel with one face cut off. When it was tested, it functioned well on a level or slightly tilted rung, though we never got around to attaching a significant amount of weight to it or trying an extreme tilt.

CONSTRUCTION

The climber, mounted to the chute, was one of the last mechanisms to be mounted onto the robot. Some aluminum gussets were machined, but our mills were frequently busy, so we created many of the parts by hand.We made some mistakes along the way (like cutting PVC with a chopsaw; it should always be hacksawed to prevent cracking).We also didn't account for the power cells passing

underneath the gearbox, which required us to quickly redesign the gearbox mount plate and relocate the gearbox higher.



The concave wheel of the Yo-Yo was 3D printed out of PC-ABS. After comparing some additional traction materials (including tread, the kit of parts foam, belts, and flex seal spray), we concluded that rubber bands were the most effective.

PERFORMANCE EVALUATION

It was a shockingly lightweight mechanism due to the use of PVC, clocking in at

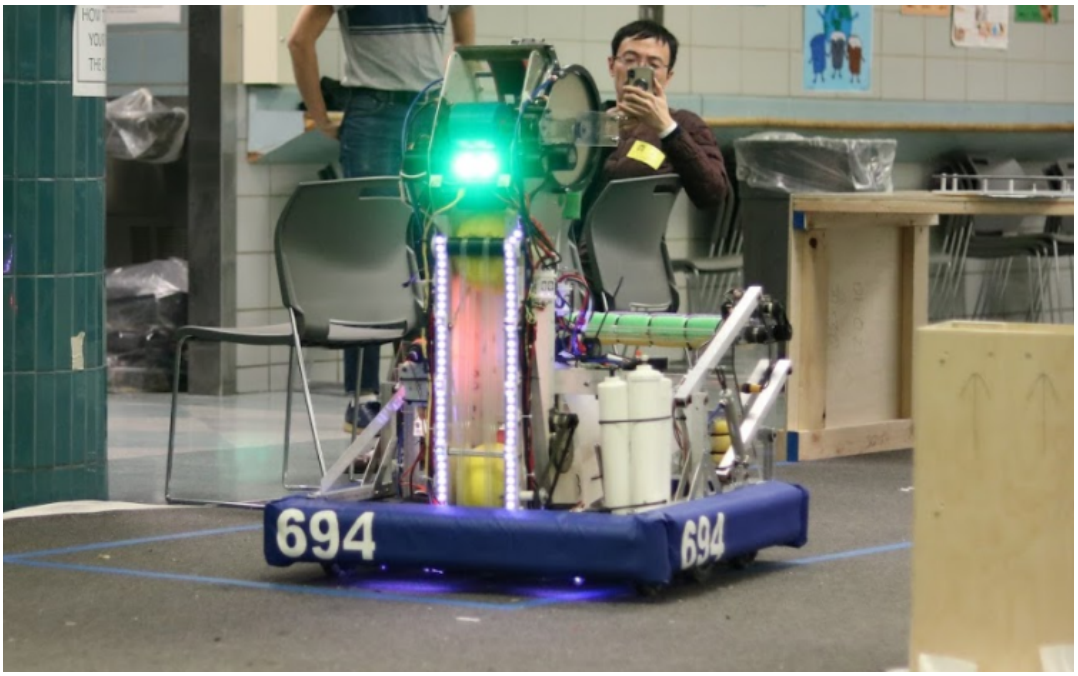
approximately 7 lbs.

The climber proved very robust, especially when used as a hook delivery mechanism. It withstood the impact of the rung and the swinging of the robot through its 15 matches and showed no significant structural issues, even when hit by an alliance partner in the air and rotating in a full circle. However, the noose that we had tied onto the hook came loose, and we remedied the issue by tying two bowlines- one for functionality and the other as backup. The rope coming loose proved to be problematic, as the design of the climber made it difficult to retrieve the rope without dismantling the entire mechanism.

The Yo-Yo was never tested at competition because the retracted hood of the shooter necessitated a redesign. In the meantime, a two-sided hook worked well at the Palmetto Regional, especially during eliminations because our alliance partner, 4020, had a mechanism that shifted along the rung.


694
694
SOFTWARE
DESIGN
694
694

Limelight

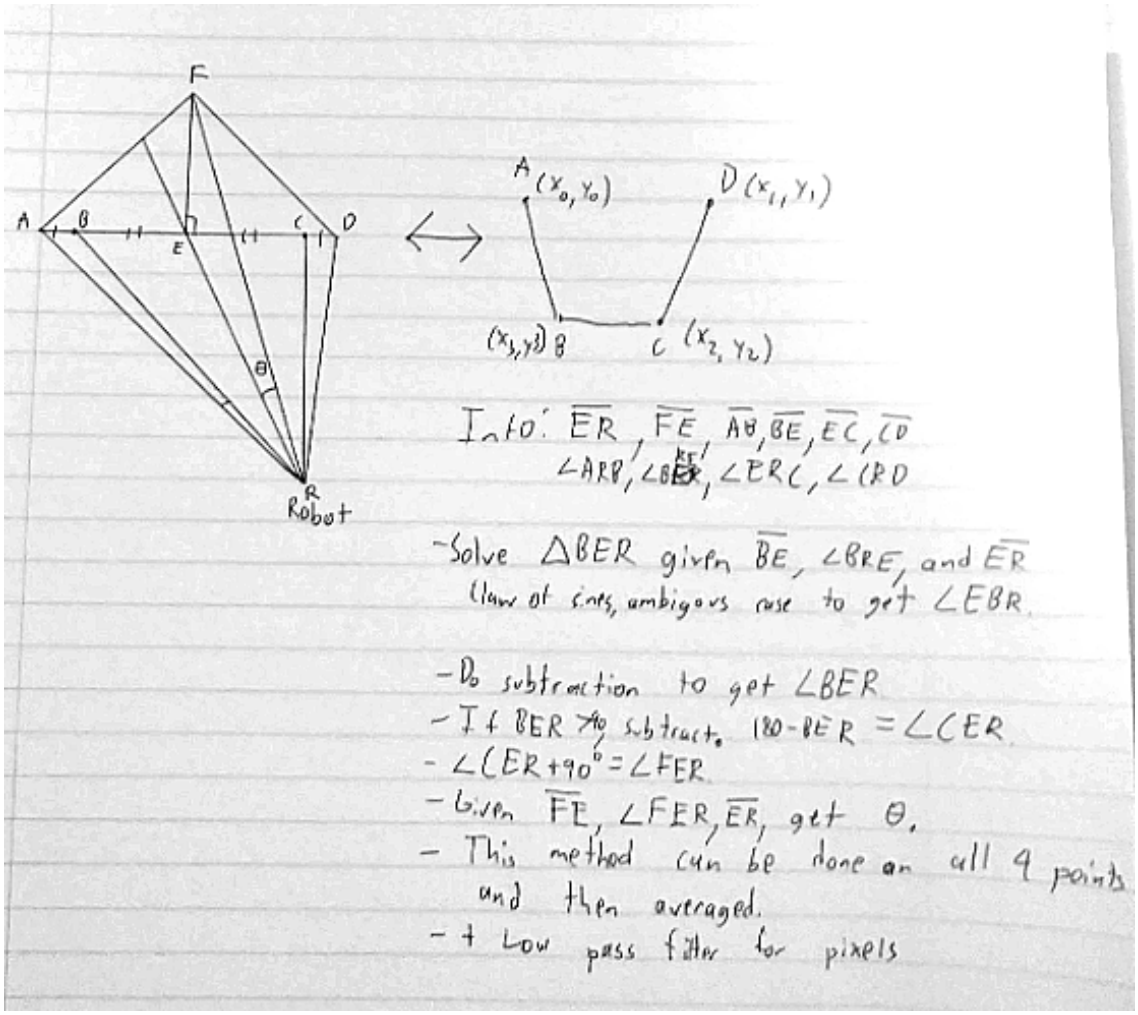


The limelight was crucial when it came to aligning our robot with the target. The target for the 2020 game has a piece of reflexite tape around the bottom half of the target. This makes it easy for the limelight to filter for the target (which is trivial). It's important that the limelight is able to shine all of its LEDs at the target unobstructed. We ignored this in 2019, which led to significant tracking issues. Applying a filter to the data received can help (but not completely) mitigate shakiness or noise.

The limelight was used to measure two types of information about the target: the angle the shooter was to the target (which can be solved with a simple PID loop) and the distance to the target (which can be found using the height of the target and the limelight).

However, the limelight cannot see the target unless tilted upwards. If you tilt the limelight upwards, you must record its angle precisely, which led to issues with the limelight's pitch changing throughout competitions. The last issue with the limelight was the latency / update rate that comes with CV. As the limelight needs to process the images coming to the camera, there will be a built in delay when retrieving information. This led to issues when PID tuning the limelight. In order to solve this, we used the encoder values in the PID loop while occasionally polling the limelight to update the target values. This way, we get the latency of encoders with the versatility of the limelight.

Inner Goal Alignment



In order to accomplish inner goal alignment, we can utilize basic geometry or a computer vision algorithm known as solvePNP, which has yet to be perfected. The diagram above demonstrates how inner goal alignment can be accomplished with basic geometry.

BROWNOUT PROTECTION

Brownout protection prevents brownouts, which occur when subsystems draw too much current and the battery can’t keep up.

To prevent this, we take subsystems in order of importance and limit the current of a single subsystem. Then, we check whether that helped with the brownout, and, if not, it moves on to the second subsystem and so forth. After the current returns to a deemed safe level, subsystems are turned back on in reverse order.

MOTOR STALLING

Motor stalling occurs when there is not enough voltage to be distributed throughout

the many subsystems and functionalities of the robot.

Checks are used to determine if the voltage level has dropped below a threshold for a constant amount of time. If it has, it runs through a list that contains all of the subsystems and starts disabling them from least to most important until the voltage level rises above said threshold. Once it is above that threshold for a constant amount of time, the subsystems that were disabled are turned back on in opposite order

PID Tuning

Automatic PID tuning was accomplished using the **Relay (Åström–Hägglund) method.**

An simplification of this method is:

- Replace where the PID Controller is used with a Bang Bang Controller
    - This will cause the system to oscillate back and forth.
        - This is generally bad, but it will give us crucial data about how to tune the robot.
        - If the system is not able to handle these oscillations, you may get inaccurate data.
- By measuring the period and amplitude of the oscillations, you can calculate optimal PID values.

An implementation of this method can be found within StuyLib. This was very helpful when it came to tuning the Shooter, which requires specific P, I, and D, values.

PID Auto Tuner was able to find very good values relatively quickly. This may be because of the stability of the shooter, which makes PID more consistent. Auto PID tuning ran into some issues with the drivetrain, however. The shaking and momentum of the drivetrain led to Auto PID being unable to gather useful data. The shaking and momentum issues can however be avoided by applying a filter to the movement of the robot. While decreasing the precision of movements, it increased the consistency of movements, making Auto PID more feasible. It is important to keep the same exact filter and configuration between the AutoPID and the PID controller you use in order to gain usable data.

Filters & Drivesystem

Filters were an accidental recreation of something called a Digital Filter (which is very common in electronics), and there are many college courses on it.

The most uncommon part of this is the use of Interfaces and Lambdas. If you are familiar with this, feel free to skip the explanation, but for the engineers here, here's an explanation (circa Jeanne Boyarsky):

INTERFACES

Sometimes you want to write code that works with other code. For example, you want to write a routine that drives 5 feet and then turns left 90 degrees. However, the robot isn't designed yet. So the code to drive and turn doesn't exist yet. No worries. You can do that with interfaces. The interface can have a method like drive(int numFeet) and turn(int degrees). Now you can write the routine and

reference the interface. Then later, someone can write the actual drive and turn methods! It gets even better. If someone clever comes along and figures out how to drive faster or turn more accurately, the implementation can change without affecting your code/routine!

LAMBDAS

Sometimes you want to call code written by someone else. It's great when someone has already written code to do something complex and you just have to fill in code to be run "later". The trick is you want to specify something that happens in the middle. Imagine you have to bubble in questions like on standardized tests. There's already a library that knows how to fill in bubbles. But it doesn't know the answer. You can write code that looks like this: answerQuestion (x -> "A"). This code lets all the bubble filling in logic work. The code will call your code (x->"A") at the right time and know what letter to circle in. This

is called “Deferred execution”. It means that your x-> “A” logic will run later

STREAMS

So, what is a Filter? Well, in order to understand what a Filter is, we need to understand what a Stream is. Let's take a look at the class IStream:

```
public interface IStream {

    /**
     * @return next value in the stream
     */
    public double get();
}
```

So an IStream is basically a way to get a series of numbers. What that means is that every time you call .get(), you get the next number in the series. Let's imagine a situation: you have a controller and you want to get how much the driver is pressing the trigger, you can make that an IStream.

```
Drivetrain robot = ...;
Gamepad controller = ...;

IStream speed = () -> controller.getRightTrigger();
IStream turn = () -> controller.getLeftX();

while(...) {
   robot.arcadeDrive(speed.get(), turn.get());
}
```

The values for driving are just numbers, which means we can make a Stream for them. We have one stream controlling the speed of the robot, and another controlling how much it turns. The Streams get their numbers from the gamepad. This may seem redundant, but it's going to make the filters work a lot better.

FILTERS

Using streams, we can now build Filters. Let's take a look at what the IFilter class looks like:

```
public interface IFilter {

   /**
    * Get next value in Filter based on the next value given
    */
   public double get(double next);
}
```

So the idea behind filters is that you give it a value from a Stream of values, and it gives you a filtered value. It modifies the Stream one by one. This is made easier by the FilteredIStream class which takes a IStream and a IFilter and makes a Stream that automatically filters the results.

For example, it's common to square values sent to the drivertain as it gives you more precise values at lower speed, but still gives you the ability to go at full speed. But filters let you do so much more than that. You can have a filter that averages the last 10 values in the Stream to filter out noise, or a filter that limits acceleration, the possibilities are endless. For an example for a square filter, here is what the code will look like.

```
IFilter square = (x) -> x * x;

double a = square.get(3); // 9
double b = square.get(2); // 4
double c = square.get(b); // 16
```

If we wanted to make an IStream that squared all of its values, it would look like:

```
IFilter square = (x) -> x * x;

// Stream that always returns 2
IStream raw_stream = () -> 2;

IStream stream = new FilteredIStream(raw_stream, square);

double a = raw_stream.get(); // 2
double b = stream.get(); // 4
```

DRIVESYSTEM

When controlling the drivetrain, we tried to stick with Arcade Drive and Curvature Drive as much as possible. Arcade Drive takes in a speed variable and a turning variable and moves the drivetrain accordingly. Curvature Drive is identical to Arcade Drive, except it limits turning depending on how fast the robots are moving. This makes it easier to turn as all movements are curved, whereas with Arcade Drive it's easier to spin out. An important thing to note is that you can not turn in place with Curvature Drive. There are other more subtle differences between Arcade Drive

and Curvature Drive, but the only important thing is that with both, you send speed and angle values to the drivetrain.

This split is important because it lets us break down control into two simple components, Speed and Angle. Using the terms Speed and Angle can lead to some awkward grammar, but they both have 5 letters, so it makes code look nice.

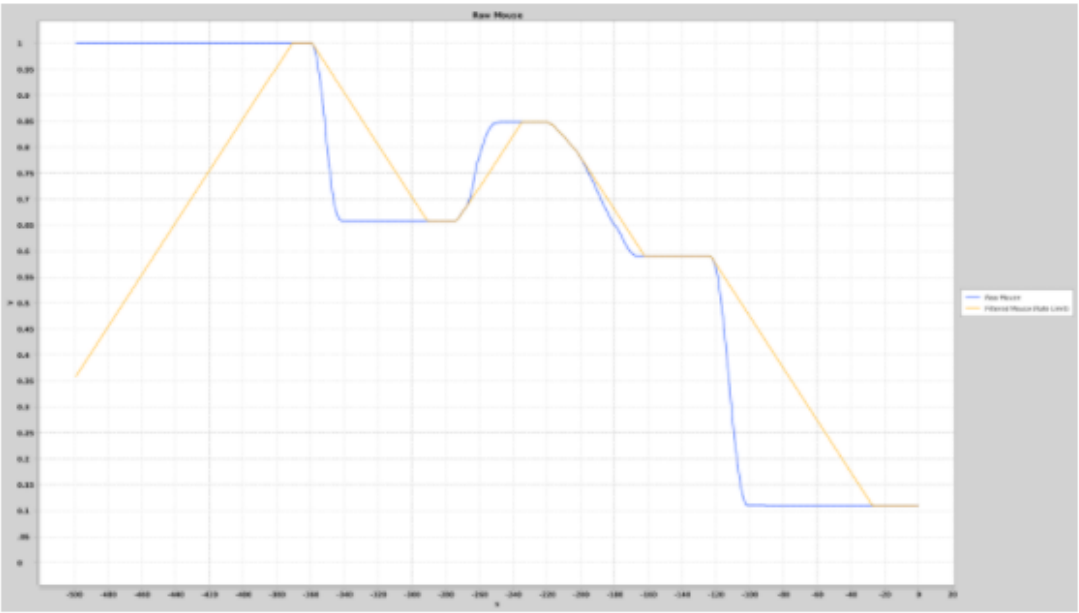
For the **Speed** of the robot, we take the **Right Trigger** and subtract the **Left Trigger**, this makes it so **Right Trigger** goes forward, and **Left Trigger** goes backwards. We can apply filters to this later.

For the Angle value, we take the **X Position** of the **Left Stick**. This is a natural place for the driver to turn with, and it frees the right hand to press the face buttons. We can apply filters to this later.

FILTERING ALGORITHMS

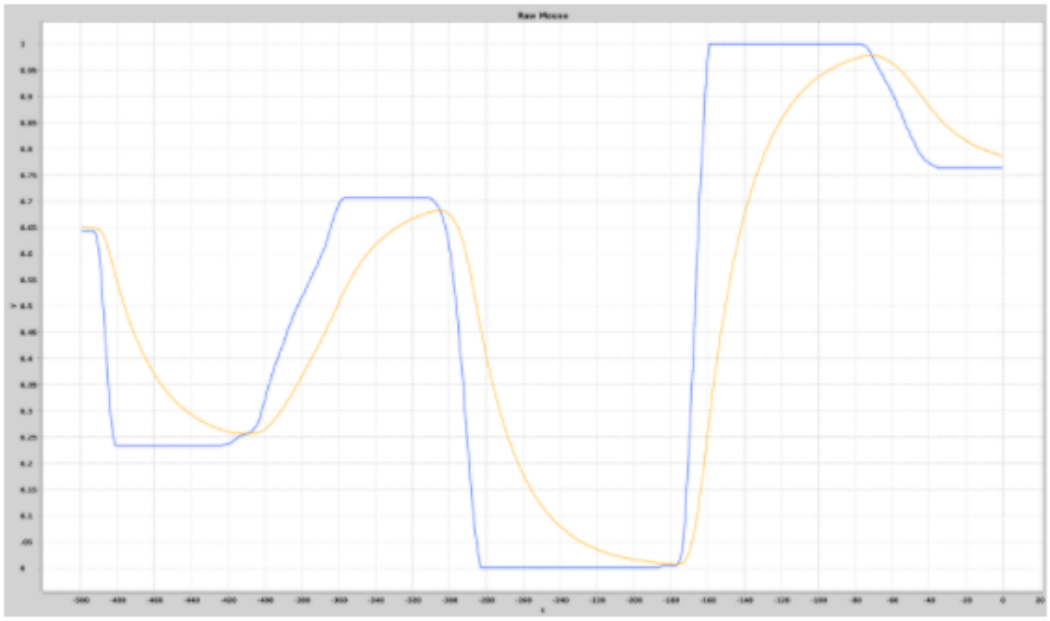
From here on out, we are going to leave the code behind us. We have created some graphs that are going to help demonstrate some of the filters, what they do, what they're good at, and why you would want to use them. All of these Filters are accessible through StuyLib and you are free to use them through the instructions above.

RATE LIMIT



Rate limiting is the simplest of the filters. All this filter does is limit the amount that a Stream can change from one value to the next. This is standard WPI ramping. It's useful for reducing power usage, but can make it difficult to control as a certain level of control is lost. We rarely used Rate Limiting as other Filters were more useful in this situation.

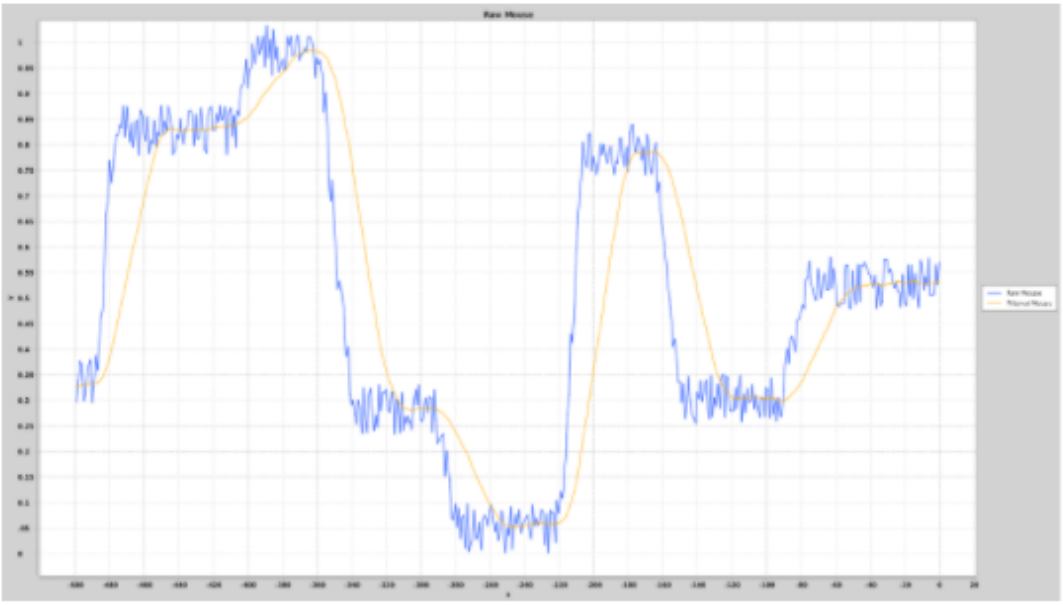
LOW PASS FILTER



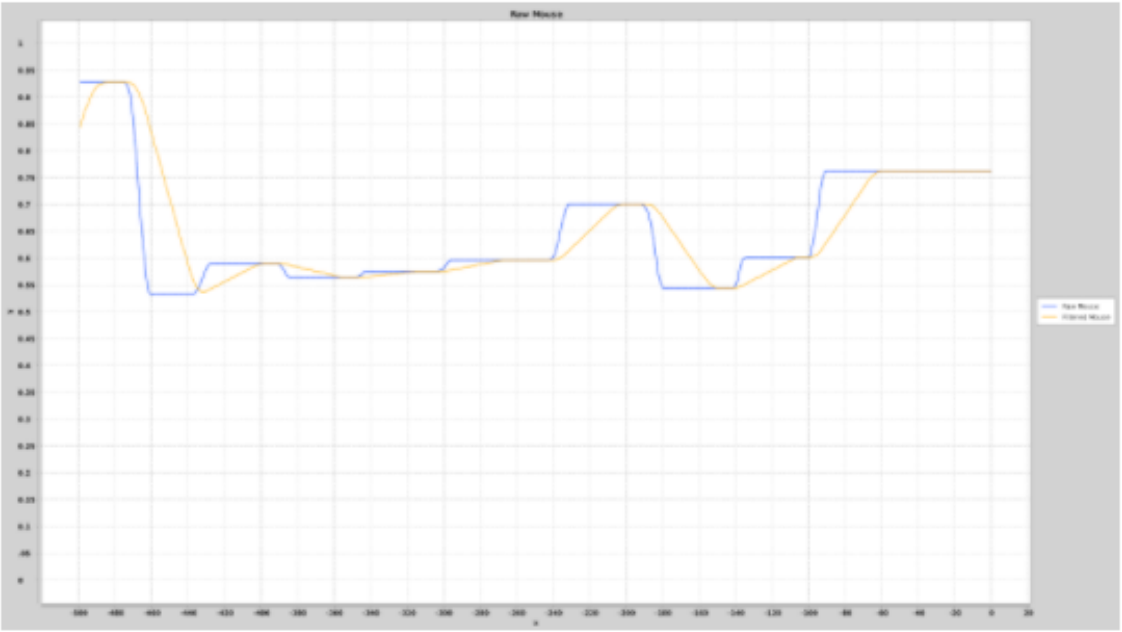
Low Pass Filters are very common in sound editing, being used to filter out high frequencies/noise. This was by far the most consistent and useful filter during our season. A Low Pass filter basically takes the distance

from the last value and the current value and finds a place in the middle, creating this curve-like effect. When tuned correctly, Low Pass Filters can make driving easier for the driver as it handles acceleration while still giving a higher level of responsiveness than Rate Limits. There is a little bit of jerk at the beginning of acceleration, but we found that this was helpful as without it, the driver wasn't

MOVING AVERAGE

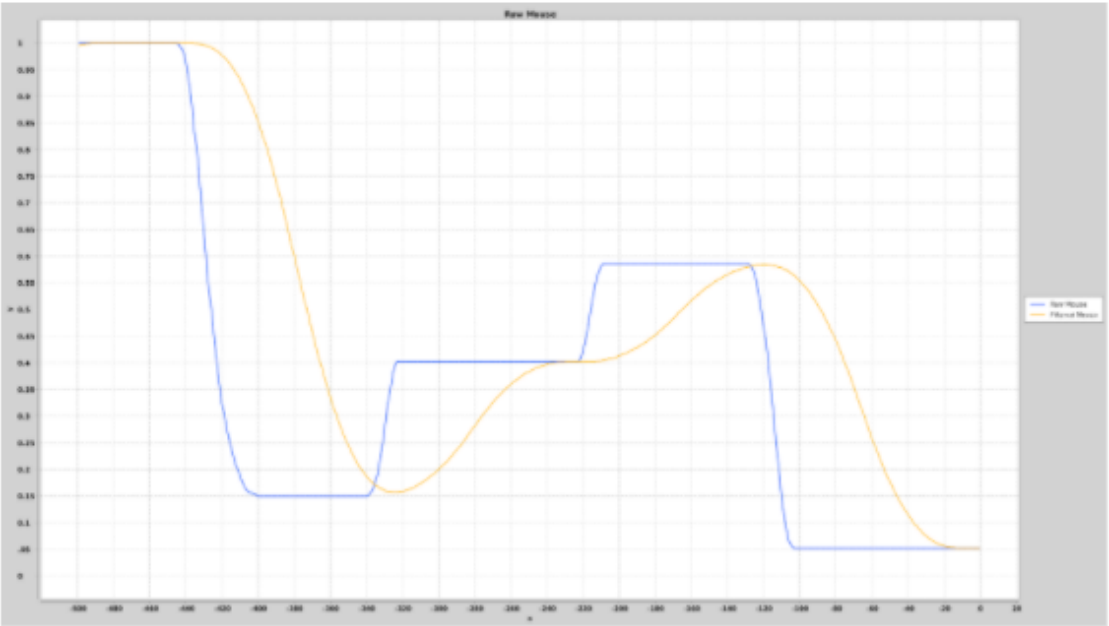


Moving Averages were not too useful for driving, but were extremely useful when it came to sensor data. As sensors are unrelated to jerk/acceleration, Moving Averages



capitalize on this and provide smoother values without as much lag. A Moving Average averages the last X values in a series and returns the result. It suffers with user data, but is helpful for reducing noise.

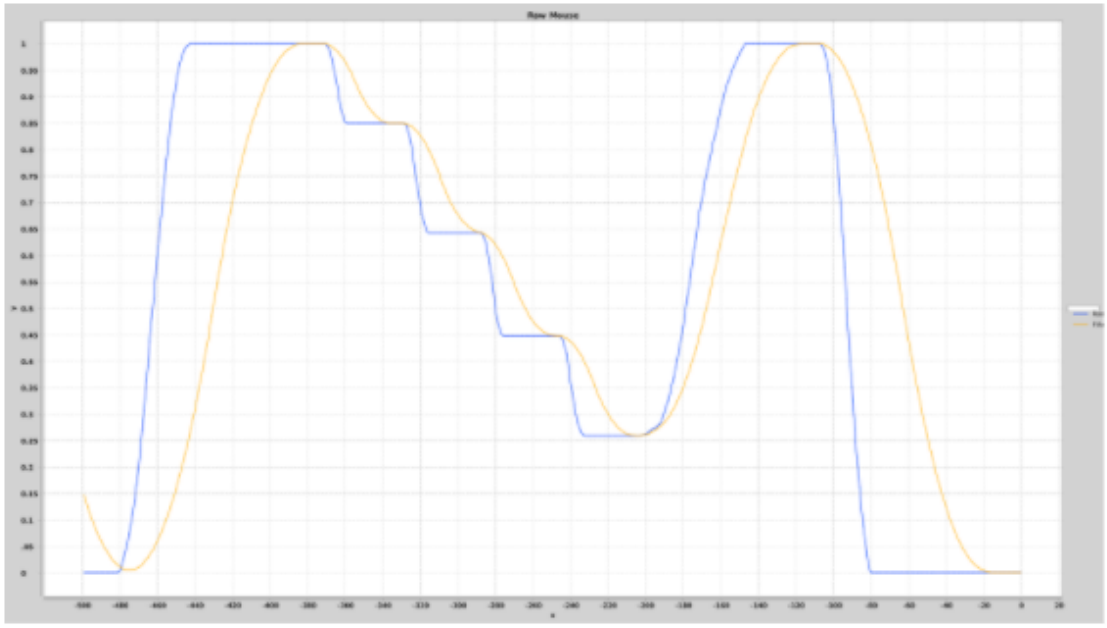
DOUBLE MOVING AVERAGE



Theoretically, applying two Moving Averages ontop of eachother provides a perfect SCurve. An SCurve, as you can see above, forms when a graph takes the shape of an S, meaning there is an extremely low jerk from beginning to end. This is very desirable, as it would mean that tipping the robot by moving back and forth would be impossible. However, it was extremely hard to control. When the driver would accelerate, it would take a second for it to start moving, so the driver would press harder, and then it would rapidly accelerate. This led to our robot in one case speeding towards the drivestation, which would have nearly been a disaster if it weren't for a pole blocking the robot.

SPEED PROFILE

The hypothetical issue with using a Double Moving Average for SCurves is that it performs an SCurve when making slight adjustments like from 0.1 to 0.2. If our goal is to keep jerks under a certain threshold, this is unnecessary.



This filter took about a week of daydreaming during algebra class to workout, but when we finally tested it, it suffered from the same issue as double moving average. The calculations for this class are the most complicated of any filter on this list, and more research is needed to see what it can be used for. Unusual behavior is demonstrated when random data is given to the filter.

S-CURVES WITHOUT CRASHING

When the driver manually ramps the robot while using a Low Pass Filter, it creates an SCurve that the driver is able to react to. This is why Low Pass Filters were the most common filter for us to use during the season.


FIRST
FIRST
STUYVESANT
694
ROBOTICS
694
STUYVESANT
694
ROBOTICS
694


A&B
Goldman Sachs