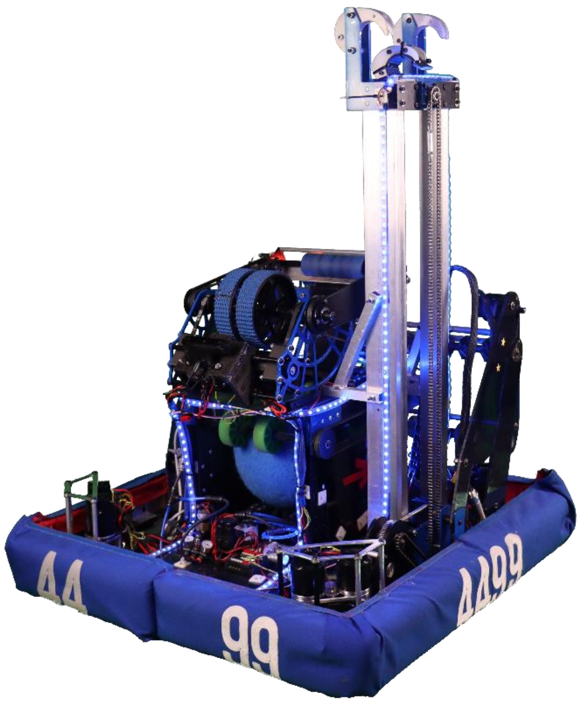


THE HIGHLANDERS

#4499

2022

TECHNICAL BINDER

CONTENTS

ANALYSIS 3

ROBOT DESIGN 4

- Drive Train 4
- Intake 5
- Feeder 6
- Shooter 8
- Climber 9

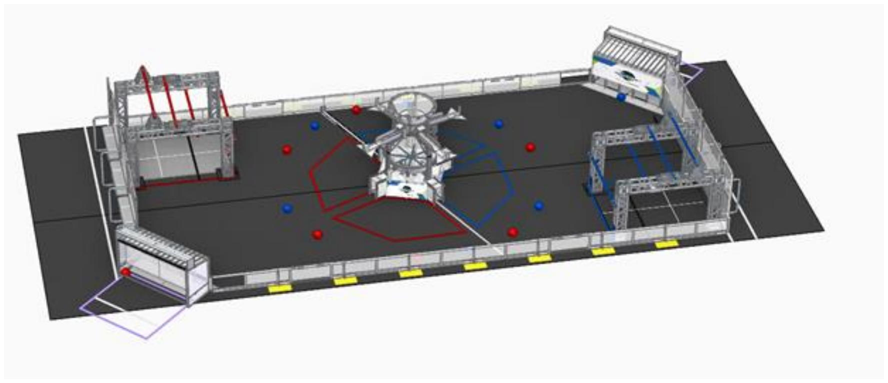
PROGRAMMING 10

- Autonomous 10
- Teleop 10
  - Overview 10
  - Swerve Drive Control 10
  - Odometry 11
  - Generating the Velocities 11
  - Simulation 12
- Vision 13
- Pathing tool 13
  - Path Creation 14
  - More About the Wheel Paths 16
  - File System 17
  - Path File Format 18
  - Path Display 19
  - Python Libraries 20

ANALYSIS

This year's game called for an all-around proficient robot that can intake, shoot, and climb. All of these mechanisms need to connect effectively so we can cycle as much cargo as possible. With a more open style field this year and a scoring zone in the middle of the field, we have more angles to score from, as a result of this, the team agreed that we need a robot that can maneuver quickly and precisely. Due to the lack of safe zones, we need a robot that is able to dynamically adjust to actions of other robots. Having a robot that is able to climb is crucial because points gained by a climb are consistent and can't be defended upon. They also provide ranking points.

Our mechanisms priorities are below.



Shooter

1. Consistent aim
2. Quick shots
3. Adjustable Hood in order to make shots from varying distances

Feeder

1. Hold 2 balls
2. Quick transfer
3. Low to the ground to lower the center of mass

Intake

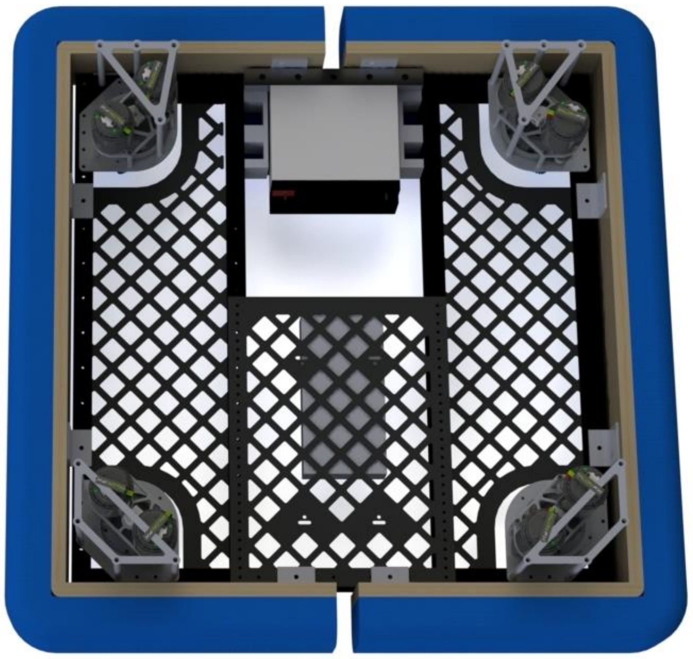
1. Durable (can handle running in to walls and other robots running in to it)
2. Touch it own it (obtain balls with inconsistent variables)

Climbing

1. Climb to bar 2 in less than 10 seconds
2. Climb to bar 4 in 20 seconds
3. Compact

ROBOT DESIGN

DRIVE TRAIN



The drive base is optimized for movement around the central hub. Swerve drive was the best option to move quickly and accurately around the central Hub. With its simple mechanical integration, the team was able to devote more resources to developing the next subsystems early on.

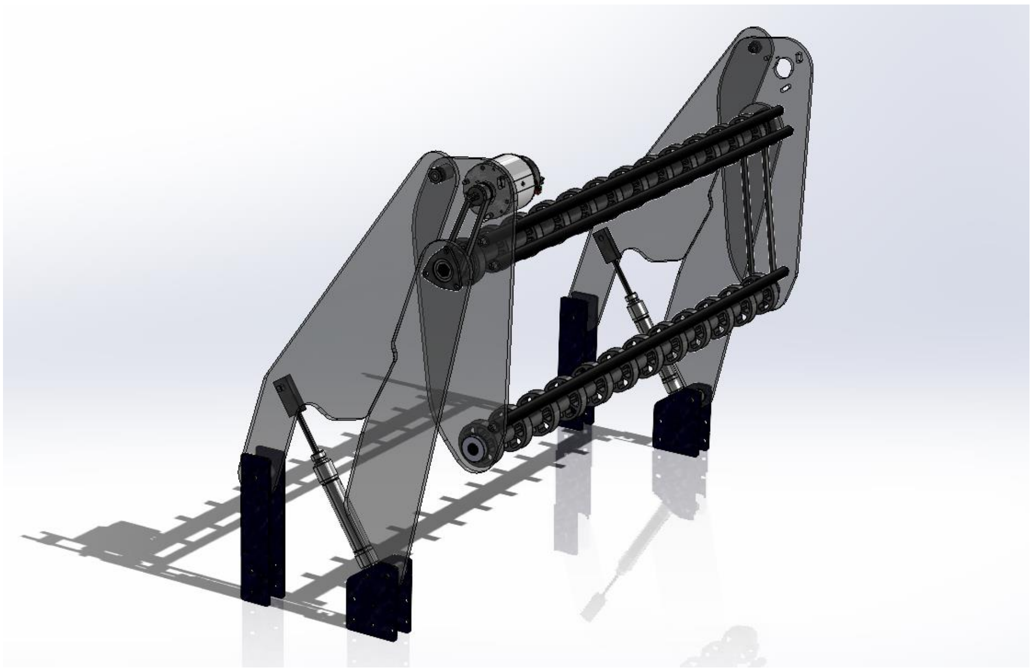
Chassis

- 4 tubes make up the outer chassis frame with standardized holes for easy subsystem addition.
- 2 inner custom tubes with weight savings for electronics routing
- Bumpers allow for ⅛" gussets to be mounted on the outside tubes
- Square design allows for easy swerve drive integration and simplifies the programming process

Swerve Drive

- 6.75:1 gearing for a max speed of 16.3 ft/sec
- 4" diameter wheels with blue nitrile treads
- MK4- L2 modules offer both a fast speed and agile movement
- Maximizes room for electronics and subsystems
- Simple to mount

INTAKE



Intake

The team chose a design that is durable and can obtain balls in any situation. This year's intake consists of 2 robot-width horizontal rollers on a pneumatically actuated four-bar to lift up to 2 balls over the bumper and into our feeder. This system allows us to run the intake into a wall without it breaking.

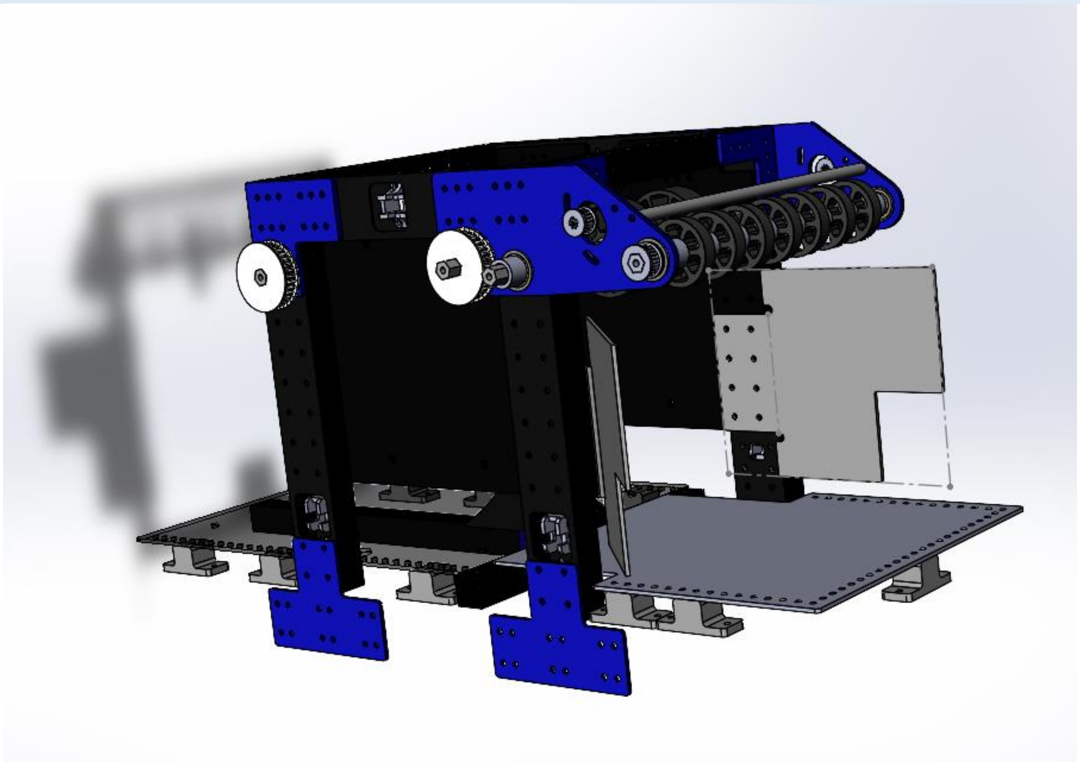
Horizontal Rollers

- 2 belt-driven aluminum rollers powered by 1 Falcon 500 motor spin at 25 feet per second which allows intaking at robot max speed
- 2" black flex wheels provide compliance when intaking balls
- 2" mecanum wheels on each side of back rollers assist in centering balls
- 3 additional standoffs run across the intake to add rigidity

Four Bar System

- ¼" thick polycarbonate plates provide durability to withstand impacts
- Non-parallel four-bar optimally positions rollers for ground pickup and storage
- ¼" Bore 2 ½" Stroke Pneumatic Cylinders on each side actuate to stow/deploy
- Can pull back into the robot for maneuverability around defenders

FEEDER



The feeder receives up to two balls from the intake. The intake centers the ball and the feeder indexes them to just before the shooter's flywheel. In order to prevent jamming we have added flaps to the side of the feeder to help center the ball.

Our feeder was specifically designed to give us the option of a dual intake to add on is we had spare time. The ability of intaking balls from both sides, this gives the driver more freedom when driving around the field.

At the center of the feeder there is a double sided ramp (3D printed) which gives the ball approximately 2" inches of compression using 3 inch compliant wheels to control when balls are released to the shooter.

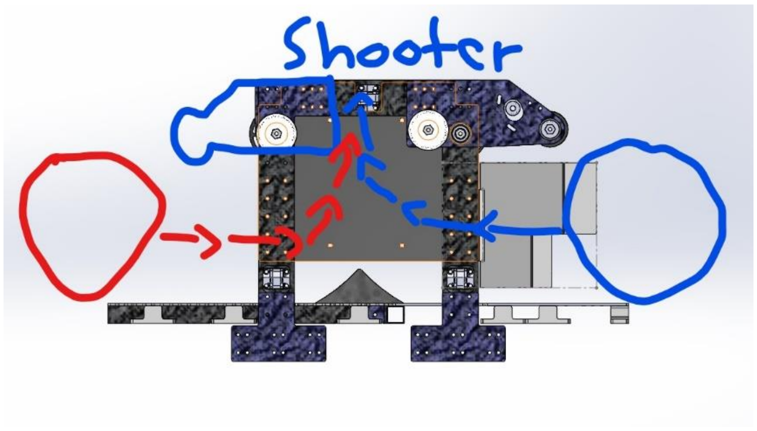
Feeder specifications

- The indexer wheels - 18 to 32 tooth pulley run by a 80 tooth belt
- Indexing wheels - 3" green compliant wheels which were used for more grip and compression of the ball to prevent slipping.
- Motor to front feeder rollers - 18 to 24 tooth pulley run by a 60 tooth belt
- Front feeder rollers - 2" compliant wheels also used for gripping the ball.
- Control – Beam Breaks, both at the entrance and at the end of the feeder
    - Allows programs to track where balls are in the system

Iterations:
We have gone through two iterations of the feeder.

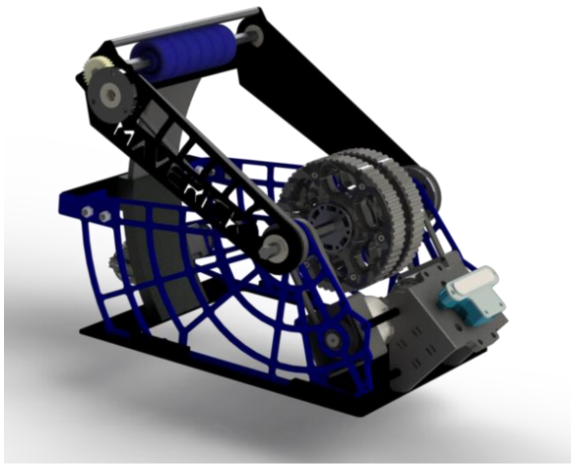
Our original design had a belt system that received the balls from the intake and transferred the ball to the indexing wheel. The indexing wheel thereby transferred the balls by moving them along the ramp and then transferring them to the shooter. The main problem with this design was that it wasn’t as fast and presented many problems such as our ramp (which was made out of thin sheet metal), would flex and not maintain the amount of compression we wanted. The mounting solution in this design was also not stable causing the robot feeder system to be unstable.

Our new feeder design took into account all the above-mentioned problems with the first iteration. The ramp was replaced with a flat plane. A speed bump was integrated in this plane so that when the ball moved through the plain and reached the speed bump it would move uphill and be captured by the two indexing wheels, where the ball would be held until it needed to be transferred to the shooter.



Ball path

SHOOTER



The shooter includes a flywheel and an adjustable hood with a backspin roller to allow the robot to shoot from multiple field locations. An Oak-D stereo camera tracks the reflective tape on the goal to position the robot during the shooting sequence.

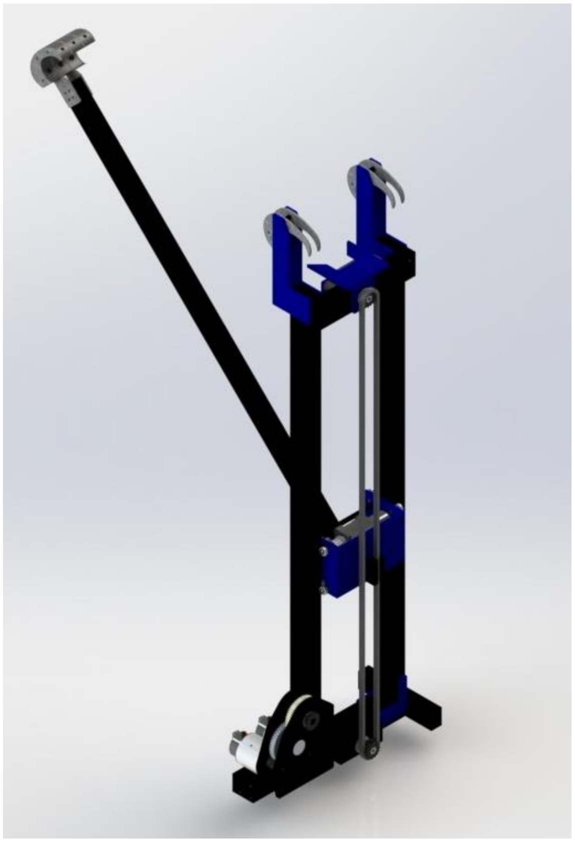
Flywheel

- Powered by 2 Falcon 500 motors with a 1:1 belt reduction
- Flywheels are 2 6” diameter wheels with blue nitrile tread for better traction
- PID used to set motor rpm
- Custom in-house made plates with weight savings

Adjustable Hood

- Allows shooting from 5° to 30° above horizontal
- Powered by 1 Falcon 500 motor with 135:1 reduction
- 2” diameter wheels on the backspin roller powered by the flywheel with a 1.94:1 reduction

CLIMBER



The climber is made up of a two stage elevator and a rotating arm to reach the high and traversal rungs. Vertical tubes create the structure for the elevator and act as rails for the carriage bearings. Structural tubes attached to the feeder transfer the climbing load to the chassis and feeder structure.

Elevator

- Powered by 2 Falcon 500 motors with 10:1 reduction
- Raises carriage to max extension height
- Extends to max height with load of 150 lbs in less than 1 second
- Gearbox attached to carriage with #25H chain
- One way ratcheting hooks mounted on ⅛” wall 1” x 2” tubes to hold robot up after the match ends
- Mounted on chassis tubes and feeder superstructure to keep robot rigidity
- Mounted in the center of mass of the robot to reduce side to side swing

Carriage

- Constrained to the elevator using ⅝” OD ¼”ID and ½” OD ¼” ID bearings

Rotating Arm

- Pivoted by a single Neo 550 motor with a single stage pulley reduction and a 2 stage versa planetary for a total reduction of 114.3:1
- Hooks have blue nitrile treads over the rung area to reduce swinging by increasing friction
  1”x1½” ⅛ wall tube to reduce bending when hitting hard stop

PROGRAMMING

AUTONOMOUS

Our autonomous plan is for the following:

2 ball – move off the line, intake 1 then shoot 2

3 ball – shoot 1, move off the line intake 1 then move to ball 3, shoot 2

4 ball - shoot 1, move off the line intake 1 then move to ball 3, shoot 2 then move to ball 4 then shoot

4 ball - shoot 1, move off the line intake 1 then move to ball 3, shoot 2 then move to ball 4, pause to intake ball from human player then shoot

TELEOP

OVERVIEW

Constant acceleration interpolation is an algorithm developed for several robotics purposes, but works extremely well for autonomous pathing of a swerve drive in FRC. The idea behind the constant acceleration interpolation algorithm is to maximize velocity and minimize acceleration changes during an autonomous path. The algorithm accepts a list of points in a JSON format generated by our path tool, that includes information such as time, x, y, and theta values at a given point, as well as a few other pieces of information, and returns a velocity in the x direction, a velocity in the y direction, and a velocity in theta(how much theta should change in a second).

SWERVE DRIVE CONTROL

When programming our swerve drive, we realized that it would be best to program it in a way that would easily allow us to program for our autonomous. To do so, we programmed our swerve drive to accept three values, velocities in the x, y, and theta directions. Using the x and y velocity, we determine the angle that each wheel must face, as well as what velocity the wheels should be spinning.

ODOMETRY

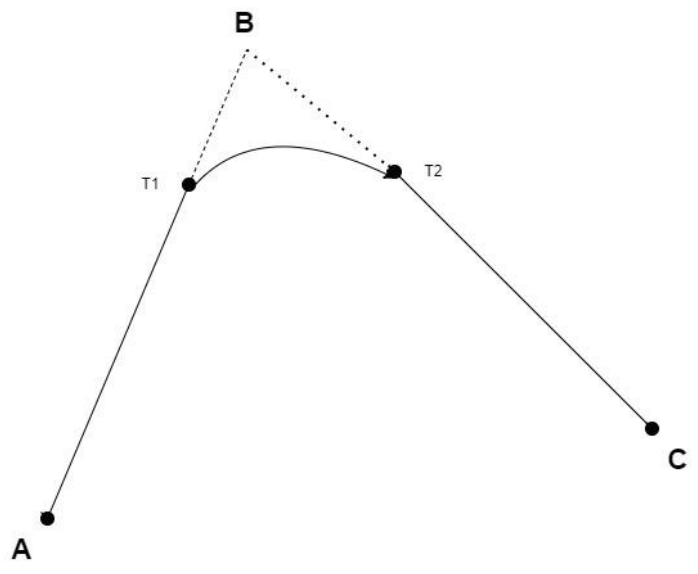
In order to complete a autonomous path, the robot must first know where on the field it is. We originally began by using just the SwerveDriveOdometry class created by wpilib, but later realized that our odometry could become more accurate by adding different sources of information to better our estimation of where we were on the field. One of these sources of information was a prediction of where the robot was on the field. We created this estimation when running the continuos acceleration algorithm, which returns x, y, and theta velocities. By multiplying these velocities by the loop time of the code, we were able to find our change in position, and adding that to our previously estimated position gave us a valid estimate of where we could be on the field. By averaging this with the prebuilt odometry class, we found much greater accuracy in our pathing and odometry.

In addition to the odometry class and prediction, we also used our camera to determine where on the field we are. Our camera has been programmed to return a distance to the target in the x and y directions, and when paired with knowledge of our gyro angle, allows us to do a little bit of math to determine where on the field we are. Averaging this estimate with our previous odometry and velocity based estimate, we determine where our robot is to a far greater accuracy.

GENERATING THE VELOCITIES

As the robot follows the path, it uses information from 3 points in the path, the previous point, the current point, and the next point. The current point is defined as the point closest to the robot in time, and the previous and next points are defined according to this point. Once these points have been determined, its time to do a little math and send velocities to the robot.

In this algorithm, there are essentially three segments, the segment before point defined at time T1, the segment in between the point at T1 and another point defined at time T2, and the segment after the point at T2.



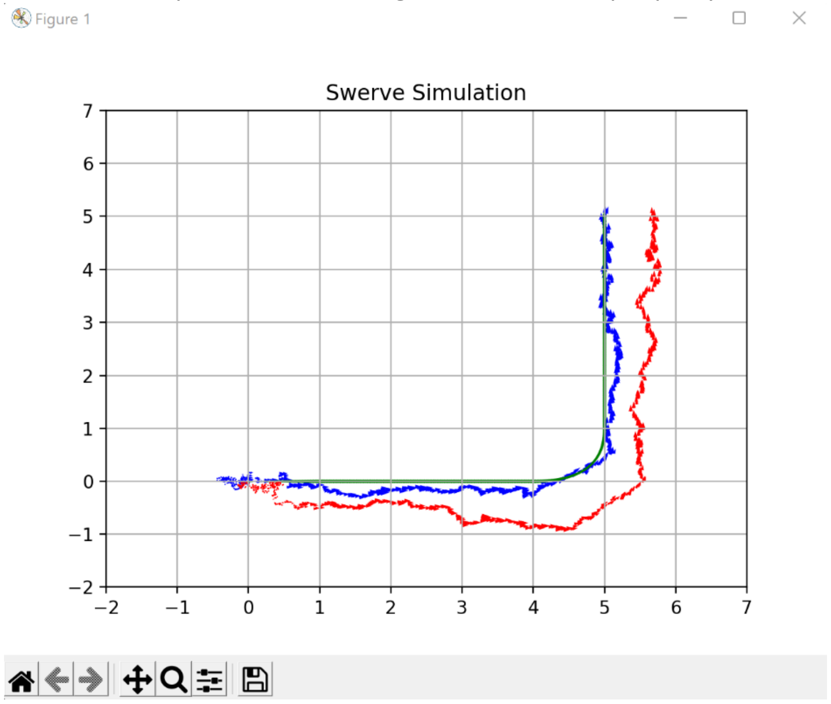
Times T1 and T2 are defined as a percentage of the total time of the line segment. For example, if the time between A and B is 10 seconds, and T1 was at 80% of the total time, T1 is at 8 seconds. If the time between B and C was also 10 seconds, T2 would be at 12 seconds.

Now, we have all the information we need to begin determining the velocity of the robot throughout the path. In segment 1(the segment between A and T1), the robot can continue at a continuous velocity. This velocity can be determined by taking the x, y, and theta value at B and subtracting them from the x, y , and theta values at A, and dividing this by the time between A and B. The robot can follow this velocity until T1.

Once the robot reaches T1, the interpolation portion of the algorithm begins. Between T1 and T2, we want to be able to change our velocity from what is required before T1 to what is required after T2. In order to do this smoothly, we need to change our acceleration constantly during the times between T1 and T2. We can calculate the change in acceleration by dividing the difference in velocity between the two points by the time between the two points. By multiplying this acceleration by our time step, we can gradually change the velocity of the robot to curve around point B on the way to point C.

SIMULATION

Due to the variety of reasons that path following could fail when running on the robot(odometry, incorrect math, etc.), we decided to test our algorithm using a simulation. This simulation was structured similarly to the robot code, running a for loop to simulate timing, and the actual algorithm accepting a JSON list. By testing our algorithm coding in a simulation, we were able to prove that our algorithm worked properly, before testing with the real robot.

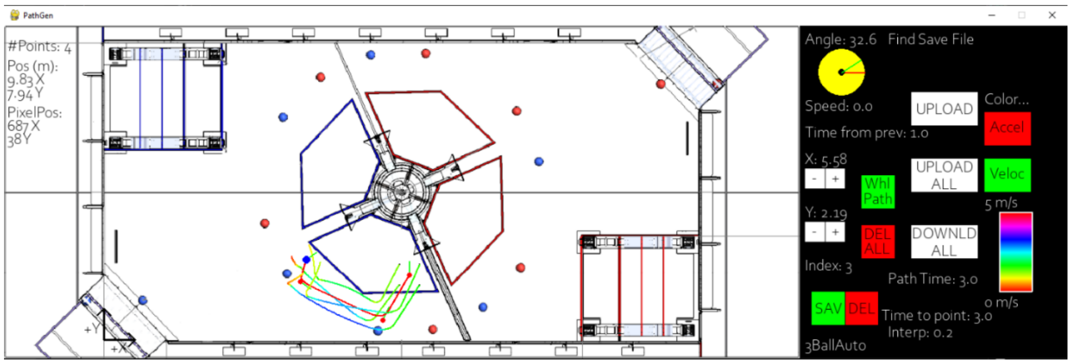


On the right side, a picture of our simulation is depicted. The green line represents the robot perfectly following the path. We also tested our algorithm by adding noise to our simulation

VISION

PATHING TOOL

PathGen is a versatile tool used to create, edit, and optimize autonomous robot paths in real time. PathGen is a custom tool that we designed to help with defining and customizing our robots path for autonomous.



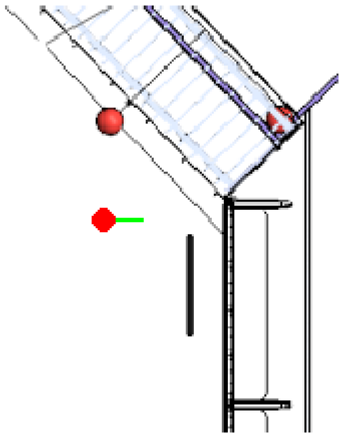
#Points: 4
Pos (m):
9.83 X
7.94 Y
PixelPos:
687 X
38 Y

Angle: 32.6
Speed: 0.0
Time from prev: 1.0
X: 5.58
Y: 2.19
Index: 3
Path Time: 3.0
Time to point: 3.0
Interp: 0.2
3BallAuto

An example path “3BallAuto” from the PathGen editor

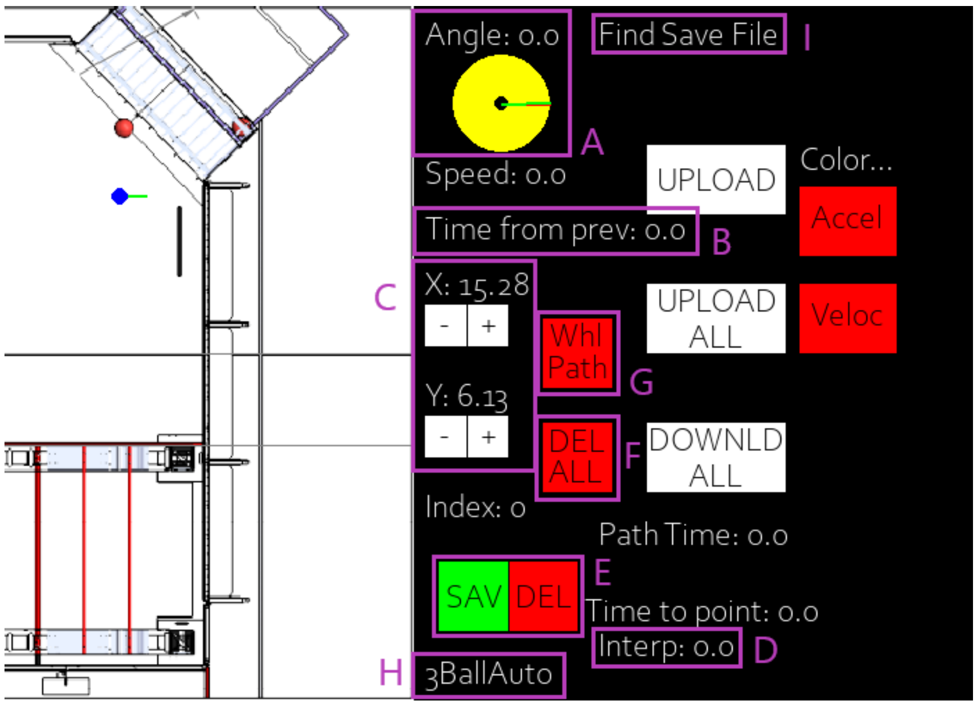
PATH CREATION

To create a path simply click on the image of the game field and a path point will be placed.



Point placed near a terminal on the field

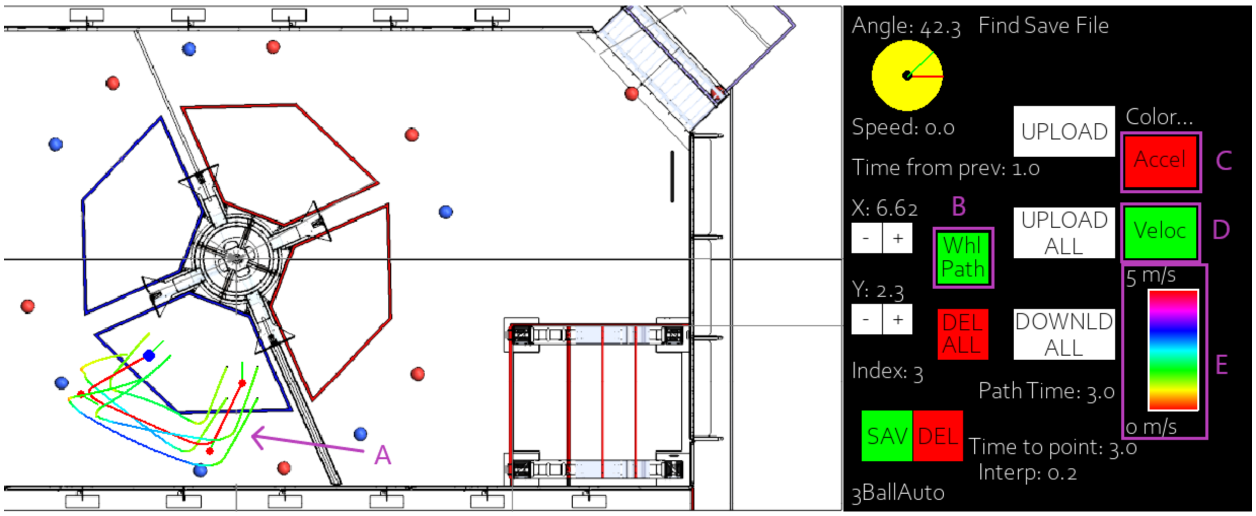
Click on an existing point to edit that point. You will see the point editor appear on the right side of the program.



Label	Name	Function
A	Angle Input	Set the angle of the selected path point, either by clicking on angle dial or through text input
B	Delta Time Input	Set the time between previous path point and selected path point
C	X and Y Input	Set the (x,y) coordinate of the selected path point (in meters)
D	Interpolation Input	Set the interpolation factor of the selected path point (more information later)
E	Save / Delete	Delete a path point or save the entire path locally
F	Delete All	Delete the entire path
G	Toggle Wheel Paths	Toggles visibility of the path of all four robot wheels (more information below)
H	Save Name	Set the filename that the path is saved under
I	Load Save	Load a locally saved path into the editor

MORE ABOUT THE WHEEL PATHS

When the Wheel Paths button toggled on (shown by changing colors to green), four new paths, one for each of the robot's wheels, will appear. Furthermore, when either the Color Acceleration or the Color Velocity Button is toggled on the color of the four wheel paths will show the acceleration/velocity of the wheel at that point.

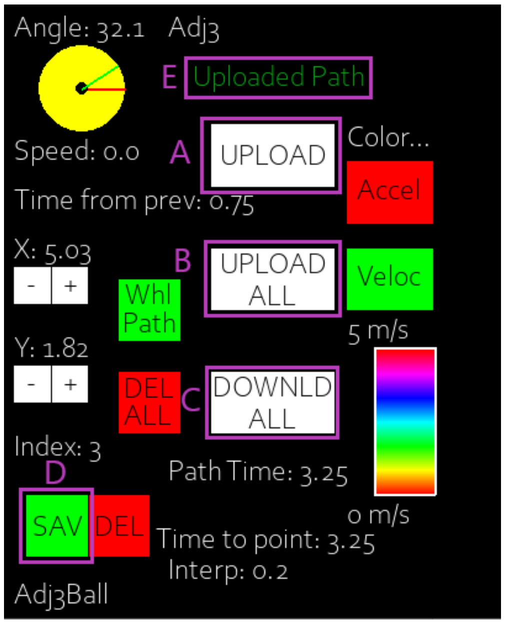


Label	Name	Function
A	Wheel Paths	Shows the path of each wheel throughout the autonomous path. Wheel paths above are colored to reflect instantaneous velocity
B	Wheel Paths Button	Toggles the visibility of the wheel paths
C	Color Acceleration Button	Toggles the wheel path coloring to reflect instantaneous acceleration
D	Color Velocity Button	Toggles the wheel path coloring to reflect instantaneous velocity
E	Color Key	Key for the selected coloring mode, values outside of shown range are displayed as black

FILE SYSTEM

The file system on PathGen saves autonomous paths as JSON objects in a folder labeled "*json-paths*". PathGen also has the capability to upload paths directly to the "*deploy*" folder on the roboRIO. **Please Note:** User file permissions for the "*deploy*" folder must be changed such that lvuser can read and write; use "chmod 777 -R /lvuser/deploy" to change this.

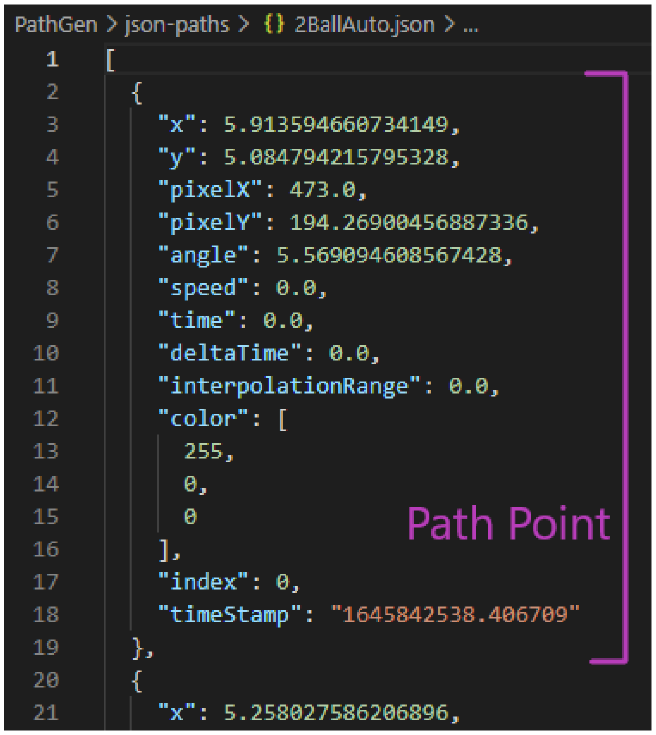
Furthermore, in case of unwanted changes or lost paths, all paths on board the roboRIO can be downloaded back to PathGen and saved again locally.



Label	Name	Function
A	Upload Button	Saves the current path locally and uploads it to the roboRIO
B	Upload All Button	Saves the current path locally and uploads all saved paths to the roboRIO
C	Download All Button	Downloads all paths on the roboRIO and saves them locally
D	Save Button	Saves the current path locally
E	Status Message	Displays status of selected operations (e.g. "Upload failed" or "Download all successful")

PATH FILE FORMAT

Paths are saved as JSON objects consisting of a list of path points each containing metadata. An example JSON path point as seen in this image.



```
[
  {
    "x": 5.913594660734149,
    "y": 5.084794215795328,
    "pixelX": 473.0,
    "pixelY": 194.26900456887336,
    "angle": 5.569094608567428,
    "speed": 0.0,
    "time": 0.0,
    "deltaTime": 0.0,
    "interpolationRange": 0.0,
    "color": [
      255,
      0,
      0
    ],
    "index": 0,
    "timeStamp": "1645842538.406709"
  },
  {
    "x": 5.258027586206896,
```

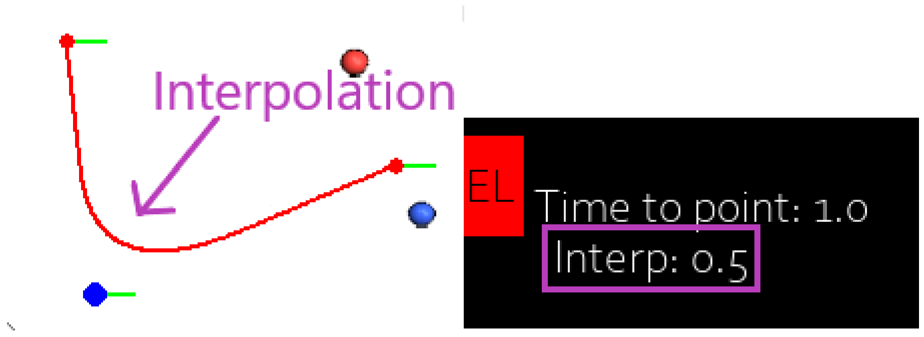
Field	Value Represented
x	Field X coordinate of the path point
y	Field Y coordinate of the path point
angle	Angle of the path point
time	Time of the path point since the beginning of the path in seconds
deltaTime	Time from the previous path point to this path point
interpolationRange	Percent interpolation of the path point as a vertex in the path (used to smooth corners of the path)
index	Index of the path point
timeStamp	Timestamp of when the path was last saved, in Epoch Time

Note: Some fields contain information that is only relevant to the PathGen program, not the robot.

Also, development is in progress on using an mqtt server to send robot odometry values back to PathGen to display the robot's position on the field in real time.

PATH DISPLAY

The displayed robot path shown in the program is generated through the use of a *Constant Acceleration Interpolation Algorithm*, which smooths corners by gradually adjusting the robot velocity between line segments of the path. The weight of this "corner smoothing" can be adjusted with the "Interp" field on the point editor. This value ranges from 0 (no smoothing) to 0.5 (maximum smoothing). The system of displaying the robot path in the PathGen program directly mirrors the actual robot code that makes the robot follow a given path.



Time to point: 1.0

Interp: 0.5

Example of an interpolation curve in a path, with the interpolation factor set to maximum

PYTHON LIBRARIES

This project is written in **Python 3.10.0** and utilizes Pygame for the user interface. Here is the list of all libraries currently being used in the program:

Library	Version
pygame	2.1.0
paramiko	2.8.1
paho-mqtt	1.6.1
bcrypt	3.2.0
cryptography	36.0.0
cffi	1.15.0
pycparser	2.21
PyNaCl	1.4.0
scp	0.14.1
six	1.16.0
syscolors	0.0.5

All of the required libraries are included in *requirements.txt* for easy installation.

PathGen is available for download at: https://github.com/HighlandersFRC/2022-Robot