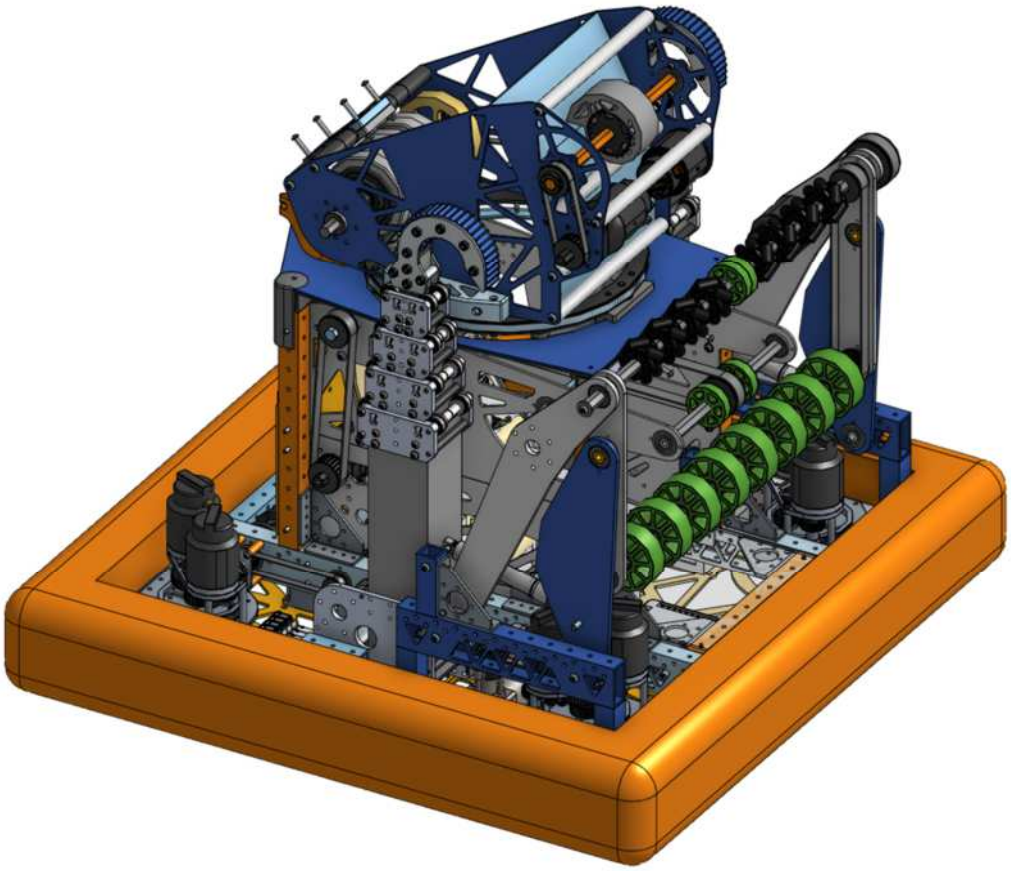
RAPID REACT

PRESENTED BY BOEING



SKADI

WOLVERINES

1757

2021-2022 Technical Binder

FORWARD

Hello, and let us welcome you to FRC Team 1757's 2021-2022 Season. This season has seen tremendous growth in our robot's design and technical ability. During the 2020-2021 season, our team decided to take a break, refocus on what is essential and prepare for the coming year. Building upon a strong recruit class of second-and first-year students along with a dedicated leadership group of veteran High School Seniors and returning Senior Mentors, we have pushed our collective talents to their limits to deliver the competition-worthy robot contained within the pages of this binder. This season was a year of First for our team, including but not limited to our first robot fully designed in CAD using Onshape. The first use of CNC fabricated parts using an Origin Router and the first competition season utilizing RoboPy as our primary software development environment.

Our season started in fall 2022, introducing a new class of over 20 freshmen, sophomores, and juniors to the world of FRC. We showed off the robot at local town events, built a T-Shirt Cannon to raise school spirit at the prep rally, and hosted weekly technical seminars on everything from the engineering process to CAD, Electronics, Pneumatics, Mechanics, and everything in-between. In January, we piled into our classroom on a cold Saturday morning in eager anticipation of this year's game. 5 CAD models, 8 shared Google Drives, 10 weeks, 20 Weekend Build Sessions, 50 Zoom calls, 5158 lines of code, 24,180 discord messages, and many, many cups of coffee later, we are proud to unveil our robot "SKADI" for the 2022 FRC Season.

Our robot is a hunter; it zigs and zags its way around the field, looking for CARGO and shooting with pinpoint precision using its keen eyesight. Therefore it seemed only fitting to name our robot SKADI after the Norse God of Hunting.[1]

We hope you enjoy this brief look at the design process and technical details that went into this robot, and if you have any questions, look for one of our team members in the stands, in the pits, or on the field. We are always ready to share the knowledge we have gained as well as share a few hard-learned lessons along the way.

[1] Ok, so actually we wanted to name it ARTIMIS after the Greek god of Hunting, but there is this small government organization called NASA and they are already using that name for some minor project they are working on. So then we thought “Hey, people like Thor…who the Norse god of Hunting?” and that is a very short explanation of a debate that consumed our team for like 5 days.

TABLE OF CONTENTS

ANALYSIS

GAME ANALYSIS AND STRATEGY

Every FIRST season for our teams starts with a rigorous design identification process to help ourselves understand and make the proper design choices and eliminate time wasted on unnecessary solutions or objectives. This season, we identified that while most points in each match would be scored by putting balls in the HUB, the HANGER POINTS at the end of the match would be necessary to ensure your teams ranking points in qualifications remain competitive in Elimination. We decided to move forward with a balanced design of a highly Accurate UPPER HUB Shooter and A Climber capable of Making it to at least The High Bar, and if we are lucky, the Traverse

Below is an outline of the design requirements we settled on after the First weekend of the build season

SHOOTER/TURRET

1. Reliability (>85%) Shooting into the UPPER HUB
2. Shooting on the Fly to reduce scoring cycle time (Min 8 Cycles per match)
3. Vison to correct Pilot Aim
4. Shooting From outside the TARMAC

INTAKE

1. Intake CARGO on the move
2. Intake 2 CARGO at once without jamming
3. Pick up off the floor
4. Use Computer Vision to assist Pilot in identification of CARGO on the Field
5. Back-drivable to eject wrong color CARGO
6. Retracts When not in use to avoid fouls during Robot-to-Robot Interaction

INDEXER

1. Hold 2 CARGO at one time
2. Use color sensors to identify when the wrong color CARGO is taken in and eject it from the bot
3. Use sensors to automate feeding into Shooter

CLIMBER

1. Climb to Mid Bar – REQUIRED
2. Climb to High Bar – Preferred
3. Climb to Traversal – Only Necessary for Elimination

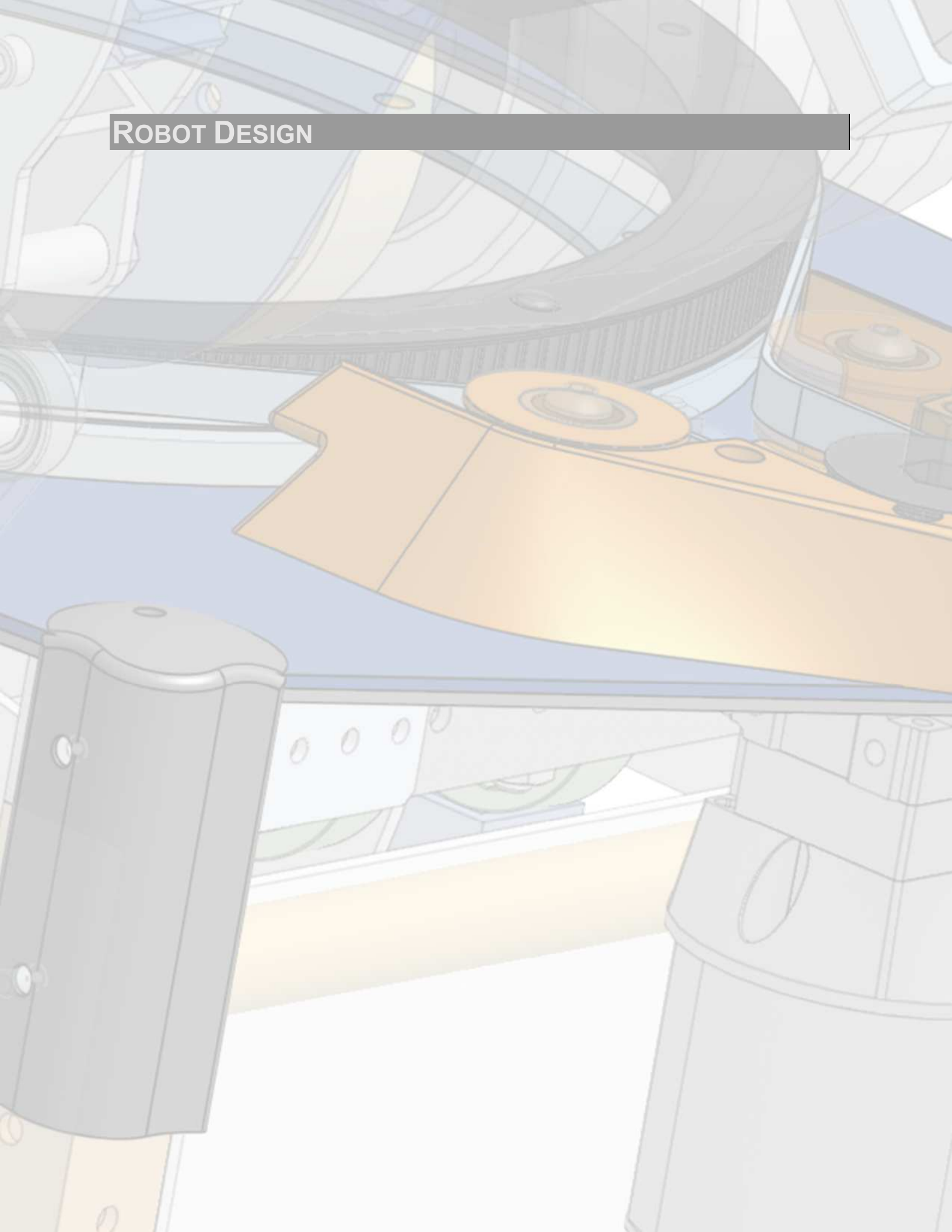
GENERAL DESIGN

Our overall goal for the season was to be a competitive bot in district-level play and qualify for New England Championship. To accomplish this, we need to, at the bare minimum, make it to Elimination at both our district events, hopefully as an Alliance captain or 1st pick.

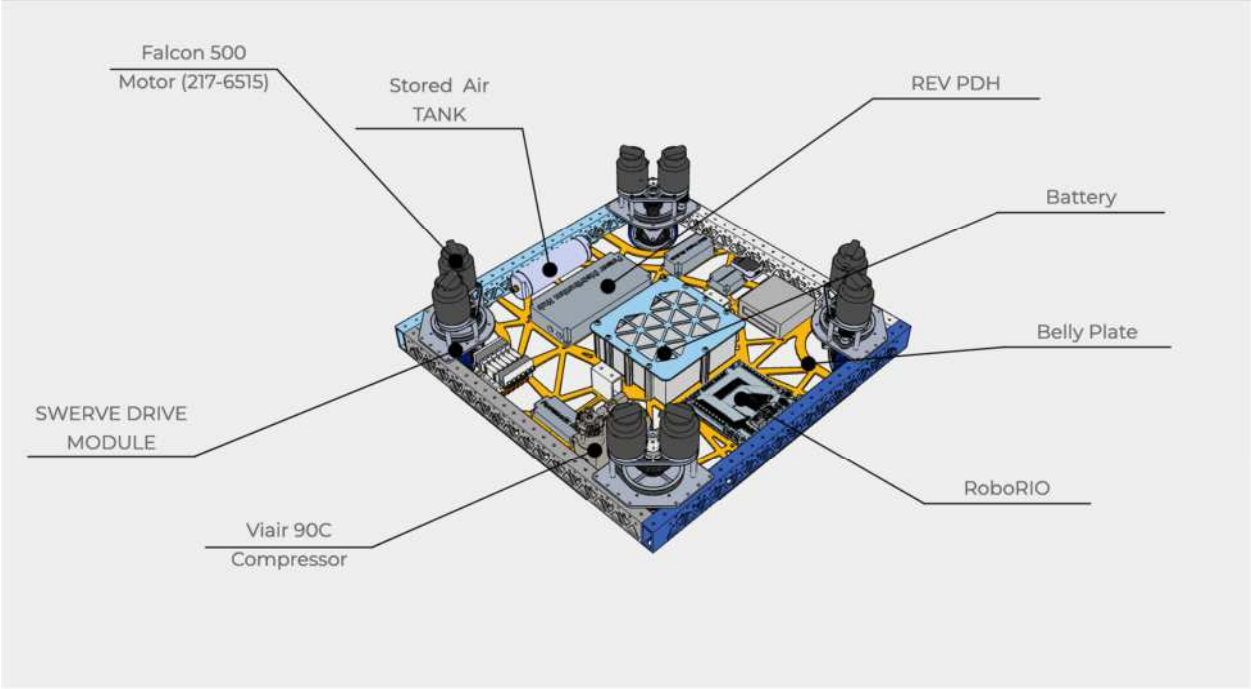
We approached our design as trying to build a highly reliable jack of all trades bot, focusing on gaining one of the two performance based ranking points in either match (Shooting CARGO or HANGER POINTS)

Inspired by the cost-effective production strategies of the Hass Formula 1 racing team and our limited team members and design resources, we prefer to use pre-engineered solutions wherever possible to focus our design resources on critical complex components.

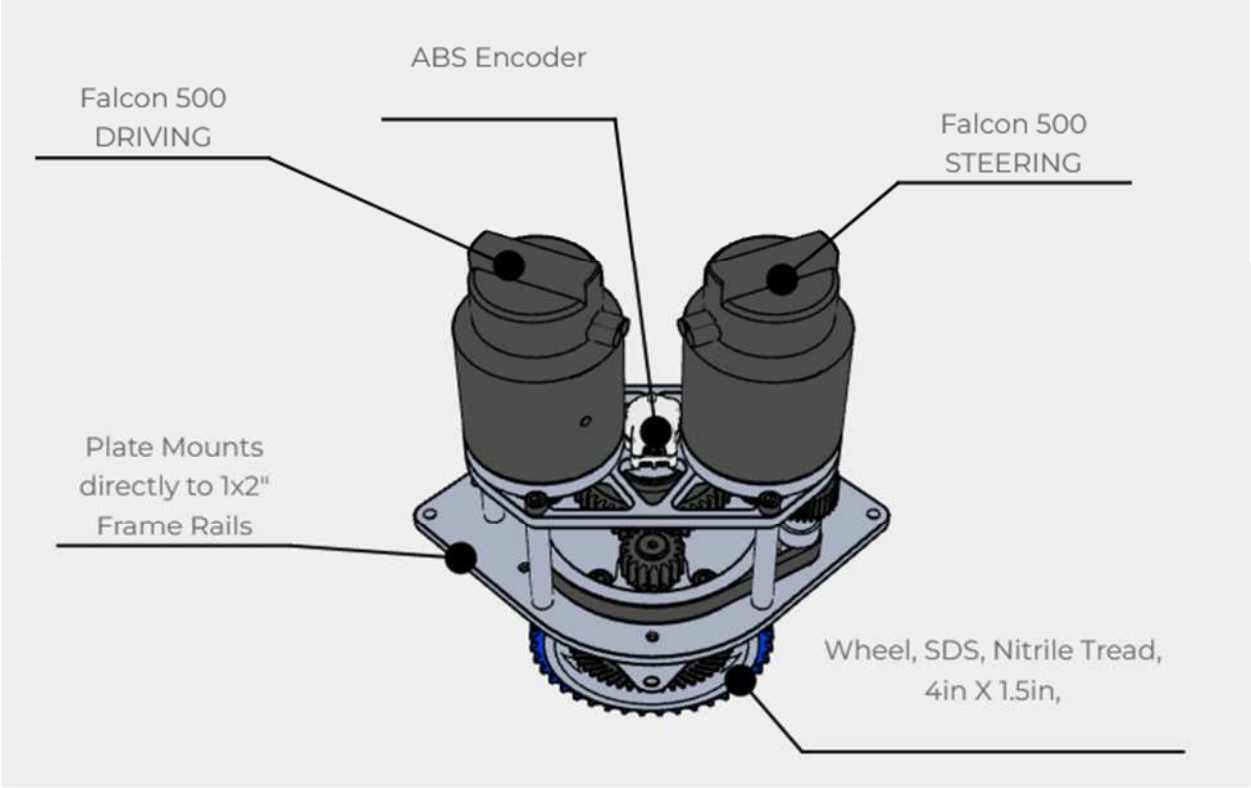
Robot Design



DRIVETRAIN



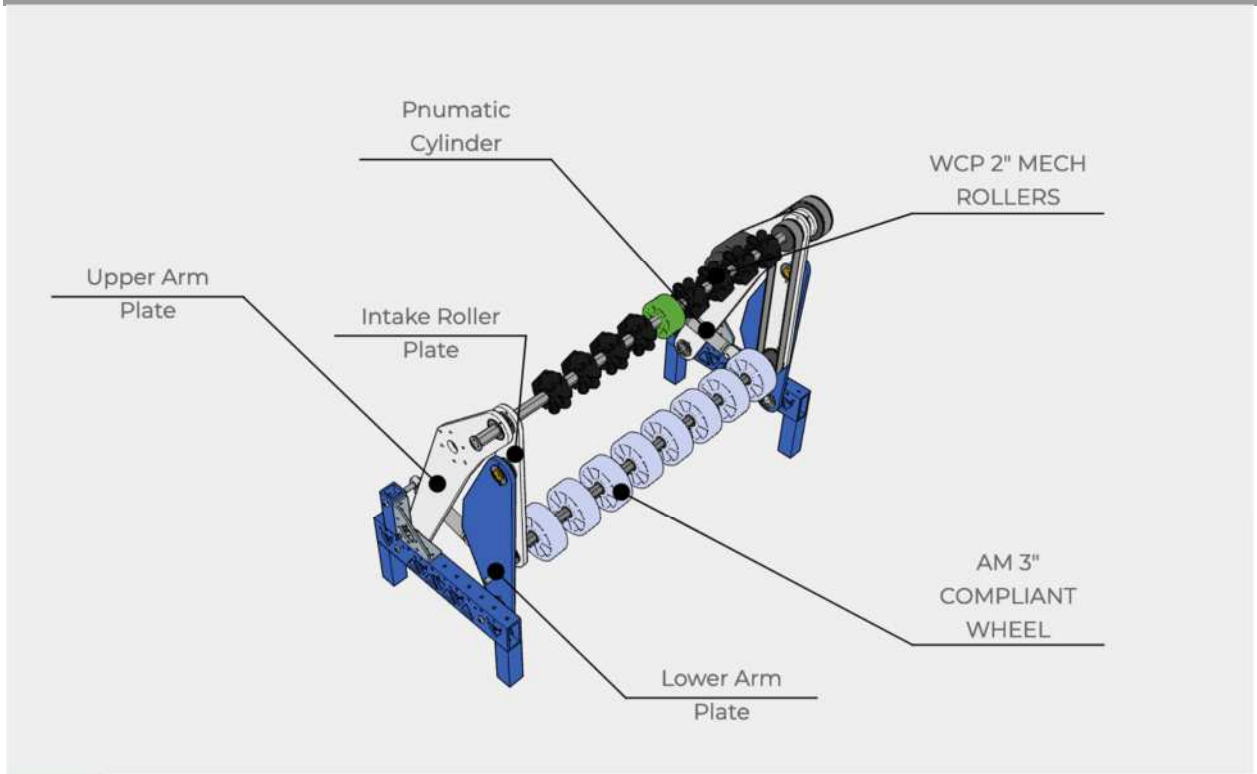
DRIVE GEARBOX



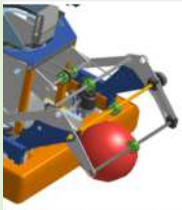
MK4 Swerve Module

Overall Gear Ratio:	**6.75:1**
Unadjusted Free Speed:	**16.3 ft/sec**
Wheel Tread Material :	**Blue Nitrile**
Wheel Tread Pattern:	**Sawtooth**

INTAKE



DESIGN



PROTOTYPE

REFINE

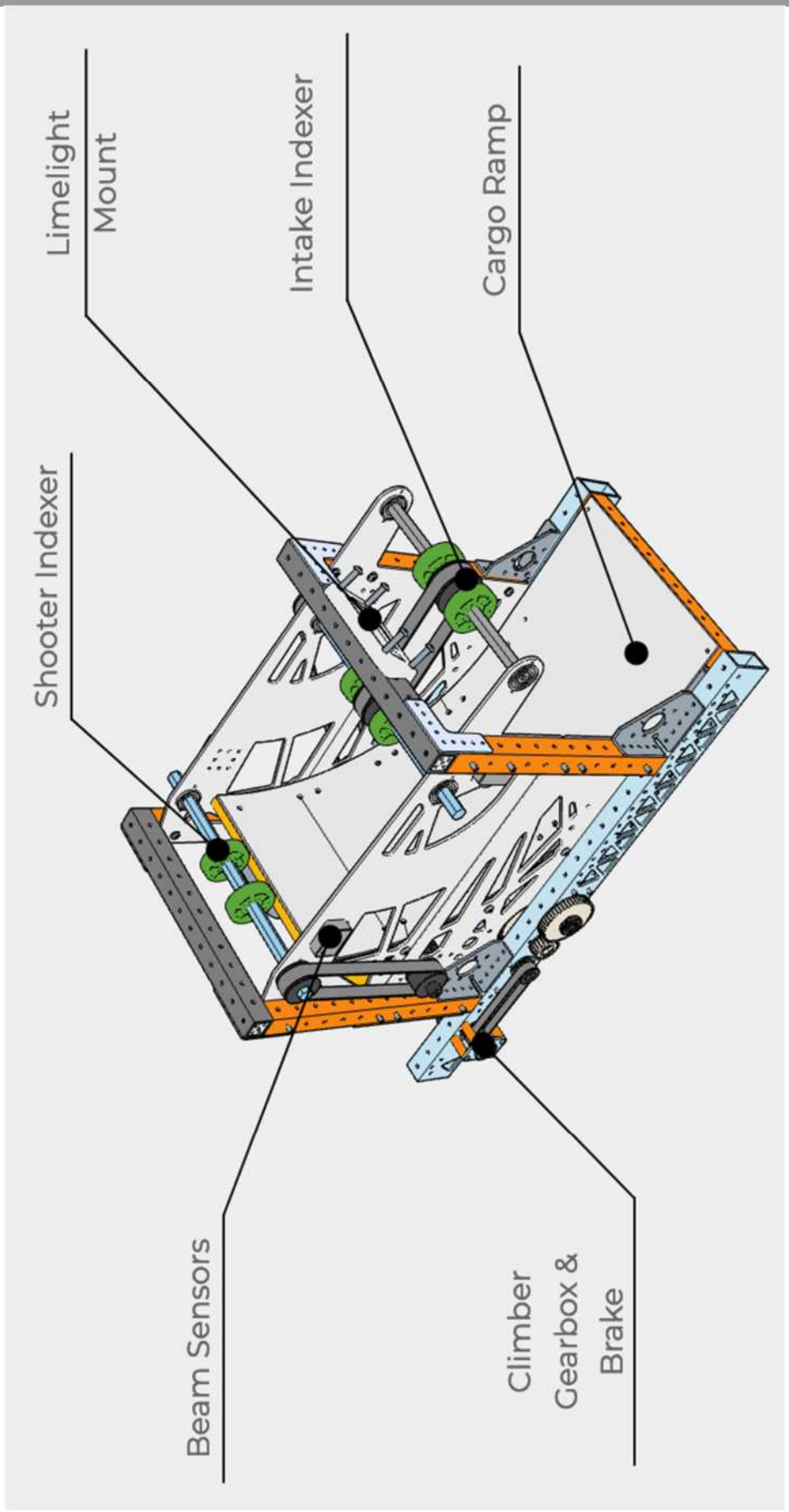
DEPLOY

Our Intake is comprised of a Four-Bar linkage that is pneumatically actuated. We prefer this to a passive intake system that would deploy at match start and stay deployed for several reasons:

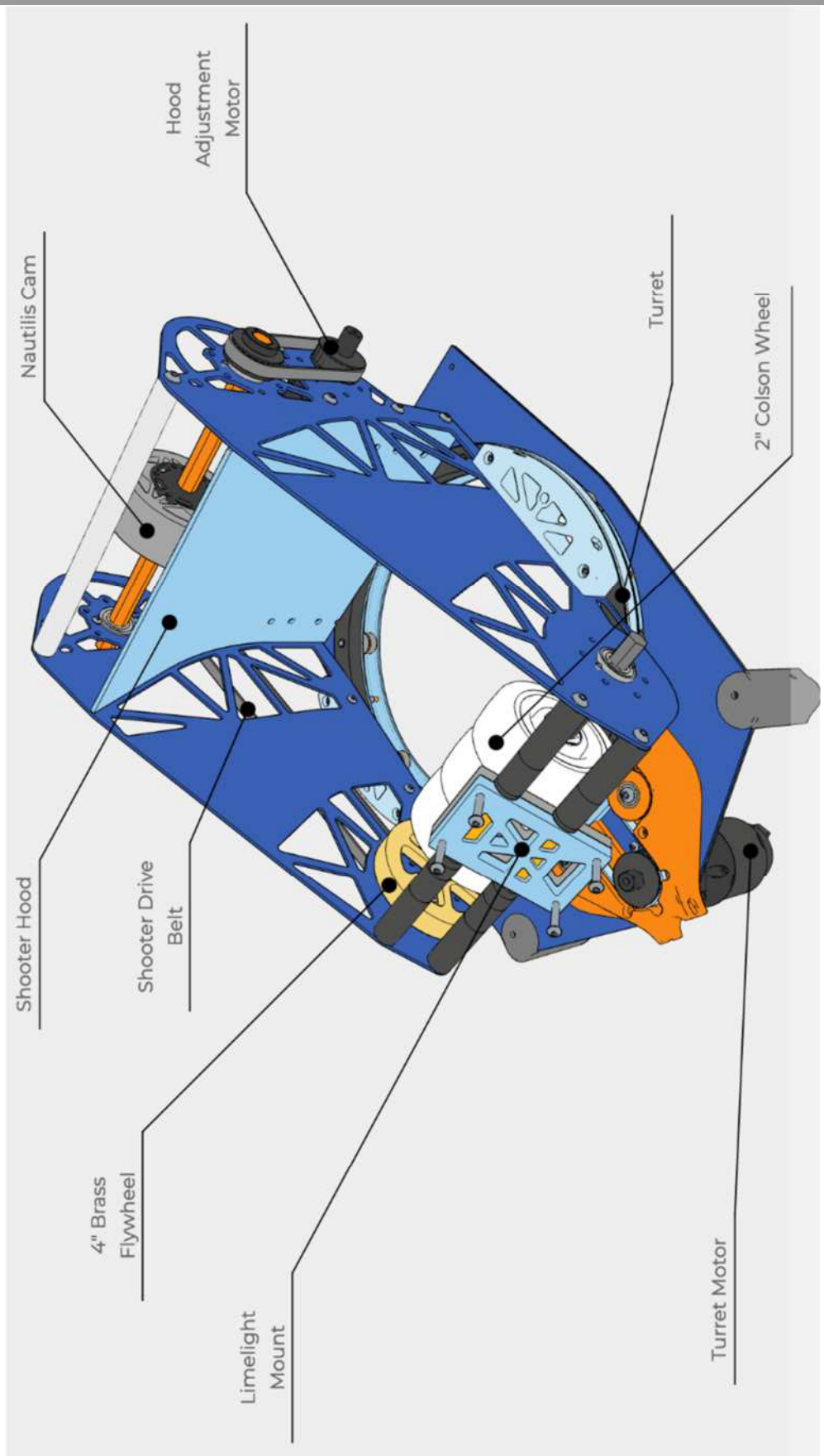
- **Retracted Intake is less susceptible to damage from Robot-to-Robot interaction when not in use.**
- **Smaller robot footprint when traversing the field.**
- **Pneumatic actuators add rigidity to overall design.**
- **Retracted, Intake works as barrier to prevent additional cargo from falling inside of bot and getting trapped.**

After experimenting with multiple roller configurations, we settled on a final design where the lower roller comprises 3" Compliant wheels to really grab the CARGO and suck it into the robot and a second roller consisting of 2" Mechanum Wheels. The mechanum wheels act to self-center the cargo as it travels through the Intake and into the indexer subsystem. This flexibility allows us to pick up balls from anywhere along with the front bumper of the robot

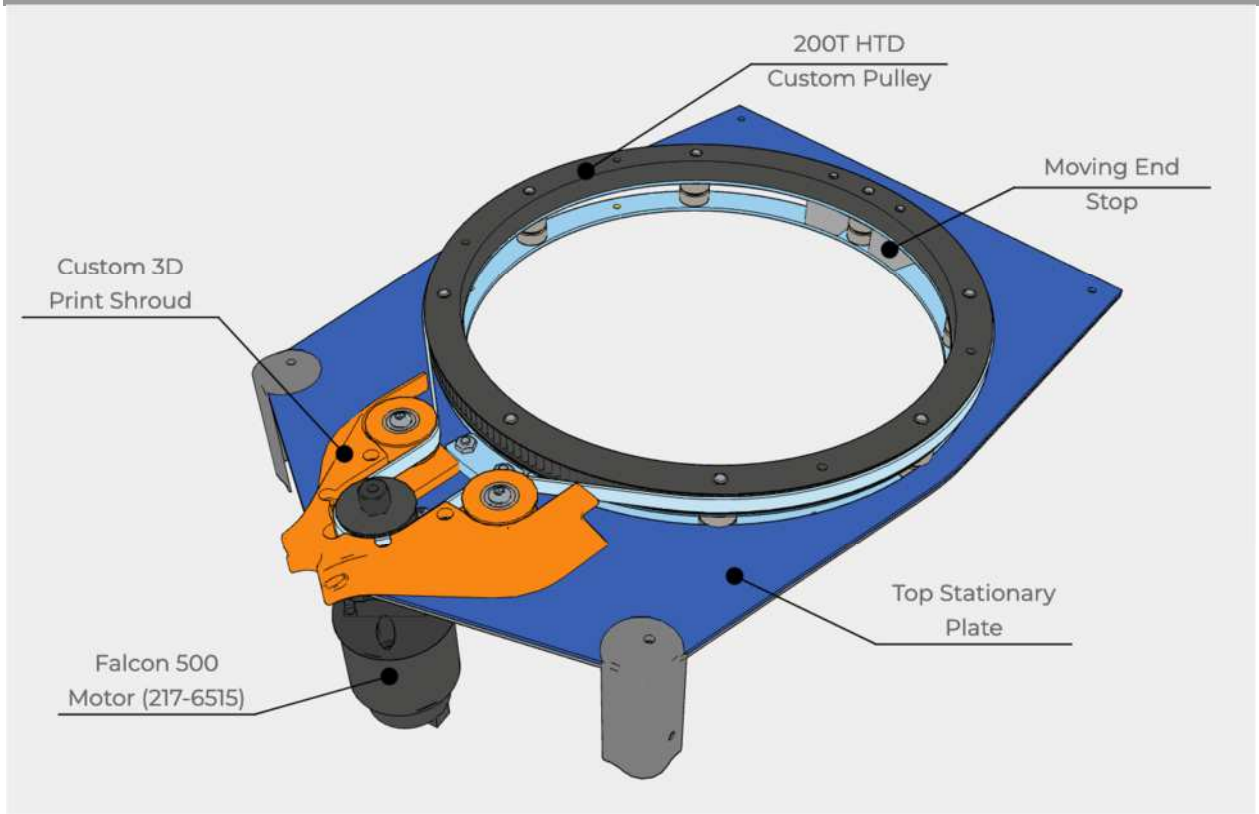
INDEXER


Limelight Mount
Intake Indexer
Cargo Ramp
Shooter Indexer
Beam Sensors
Climber Gearbox & Brake

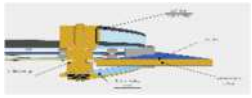
SHOOTER


Hood Adjustment Motor
Nautilis Cam
Turret
2" Colson Wheel
Shooter Hood
Shooter Drive Belt
4" Brass Flywheel
Limelight Mount
Turret Motor

TURRET



DESIGN

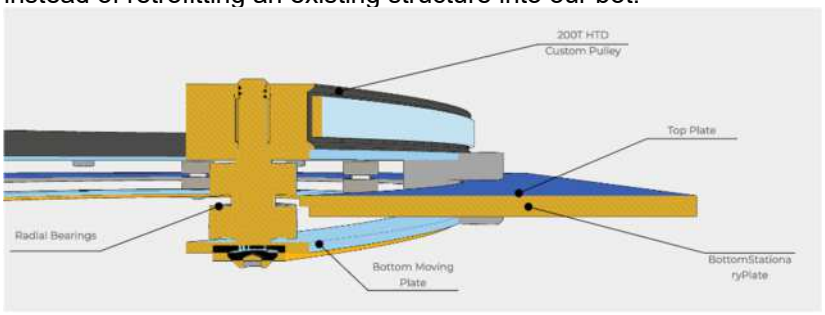


Our team has found a turret an invaluable tool in any FRC bot with a shooter. It allows for extra software compensation for the Pilot's aim beyond what they can achieve through driving alone. This year we designed and implemented a turret designed from the ground up to suit our design requirements.

PROTOTYPE

During the 2019 season, we had a bad experience using a pre-designed turret design from an FRC vendor. Determined to learn from our mistakes, we worked hard to reverse engineer our turret design. This approach gave us the flexibility to design the turret around our robot instead of retrofitting an existing structure into our bot.

REFINE

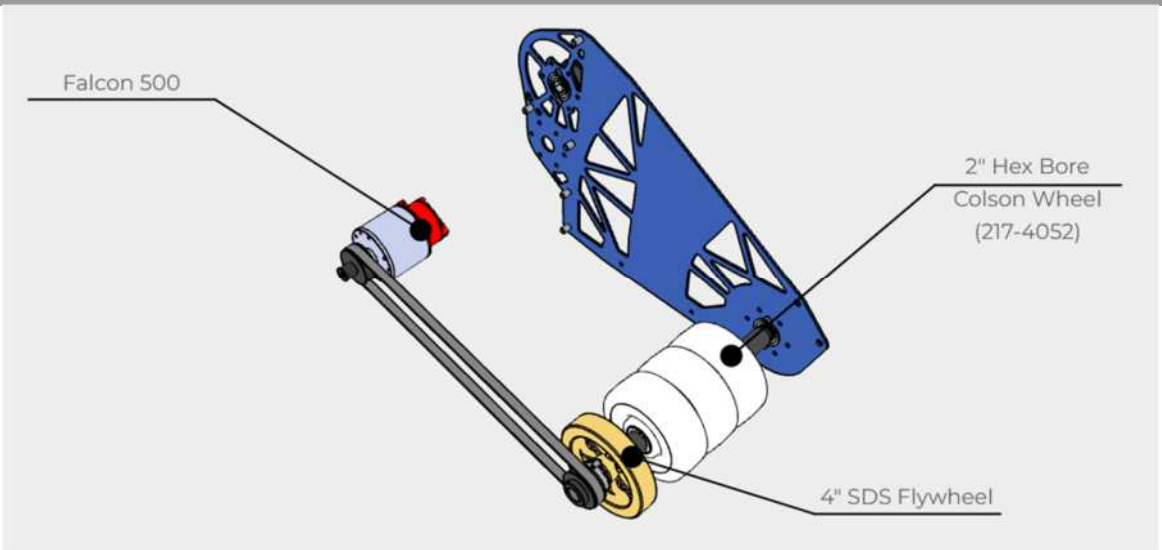


Turret uses roller bearings around two rings of slightly different diameters. It uses a Falcon 500 Motor for rotation control with the integrated encoder and has a 320° range of rotation before running into its limit switches and mechanical hard stops.

DEPLOY

We had to design a custom 3D printed shroud to protect the cable umbilical running up to power the Shooter and associated hardware from becoming caught and frayed inside the main turret drive belt.

FLYWHEEL



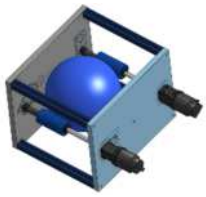
We use 3 Standard 4" Ø Colson Wheels on a 1/2" steel hex Shaft for our Shooter. In addition, we have a 4" Brass Flywheel on the shaft to increase the rotational inertia of the system. This increased rotational inertial means we don't have to pause between shots for our flywheels to come back up to speed a problem we have had in past flywheel shooter designs from previous seasons.

POWER:	**FALCON 500**
GEARBOX:	**Direct Drive**
MAX RPM:	**≈6300 RPM**
OPERATING RPM:	**4800-6000 RPM**

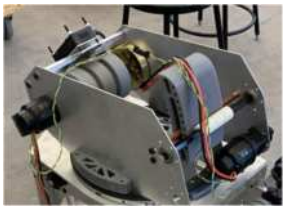
HOOD

DESIGN

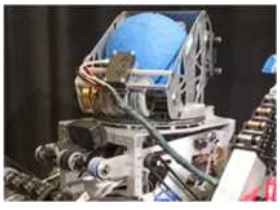
PROTOTYPE



REFINE



DEPLOY

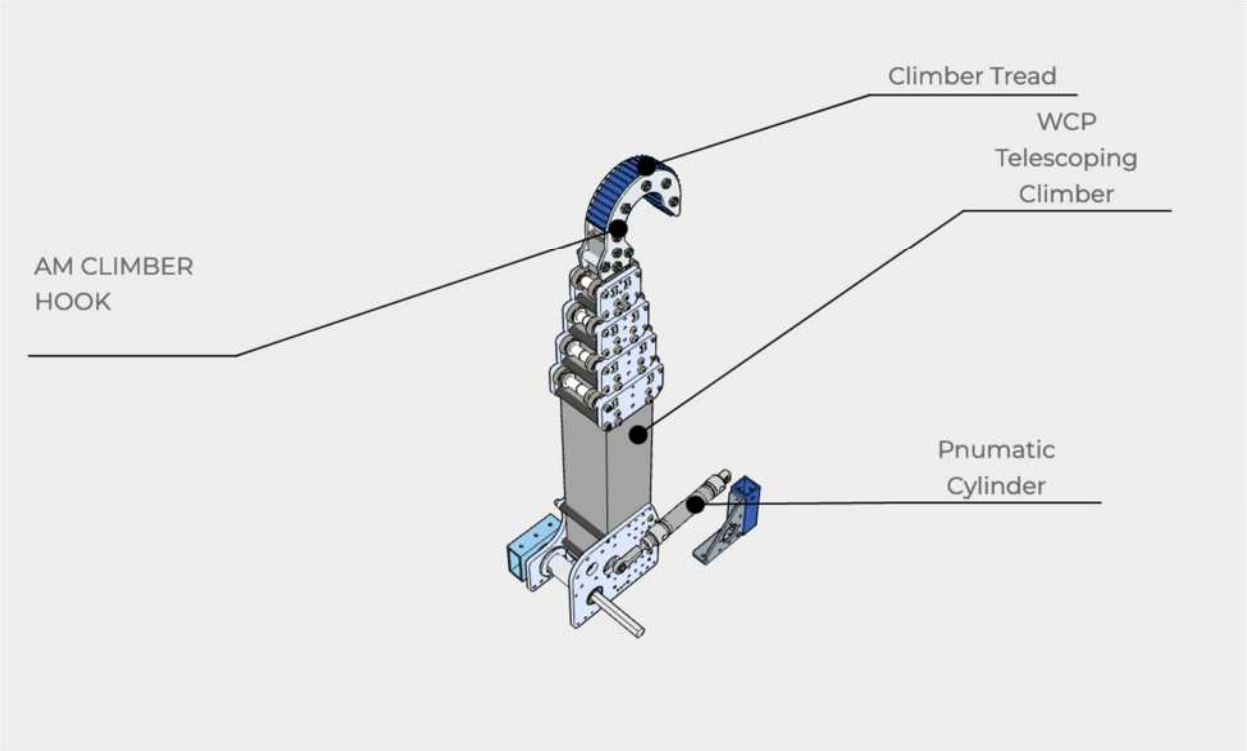


We went through two major design iterations with our adjustable hood design. The first design featured a custom 3D printed Hood which featured splines along the back surface which meshed into small sprocket gears. This allowed for us to have a fixed arc and adjust the release point of the Shooter however this design ultimately proved to fragile for implementation, either the spur gears would slip along the splines of the hood, or the force of the ball traveling along the surface of the Shooter would cause the spur gear teeth to fail.

Looking for inspiration from other FRC teams we stumbled across a few teams that were using a Nautilus-like cam to deflect a piece of polycarbonate to change the release angle of their shoot. We thought this was a brilliant idea and quickly worked on prototyping our own implementation. Satisfied with the success of that design we refined and strengthened it before final implementation on the robot.

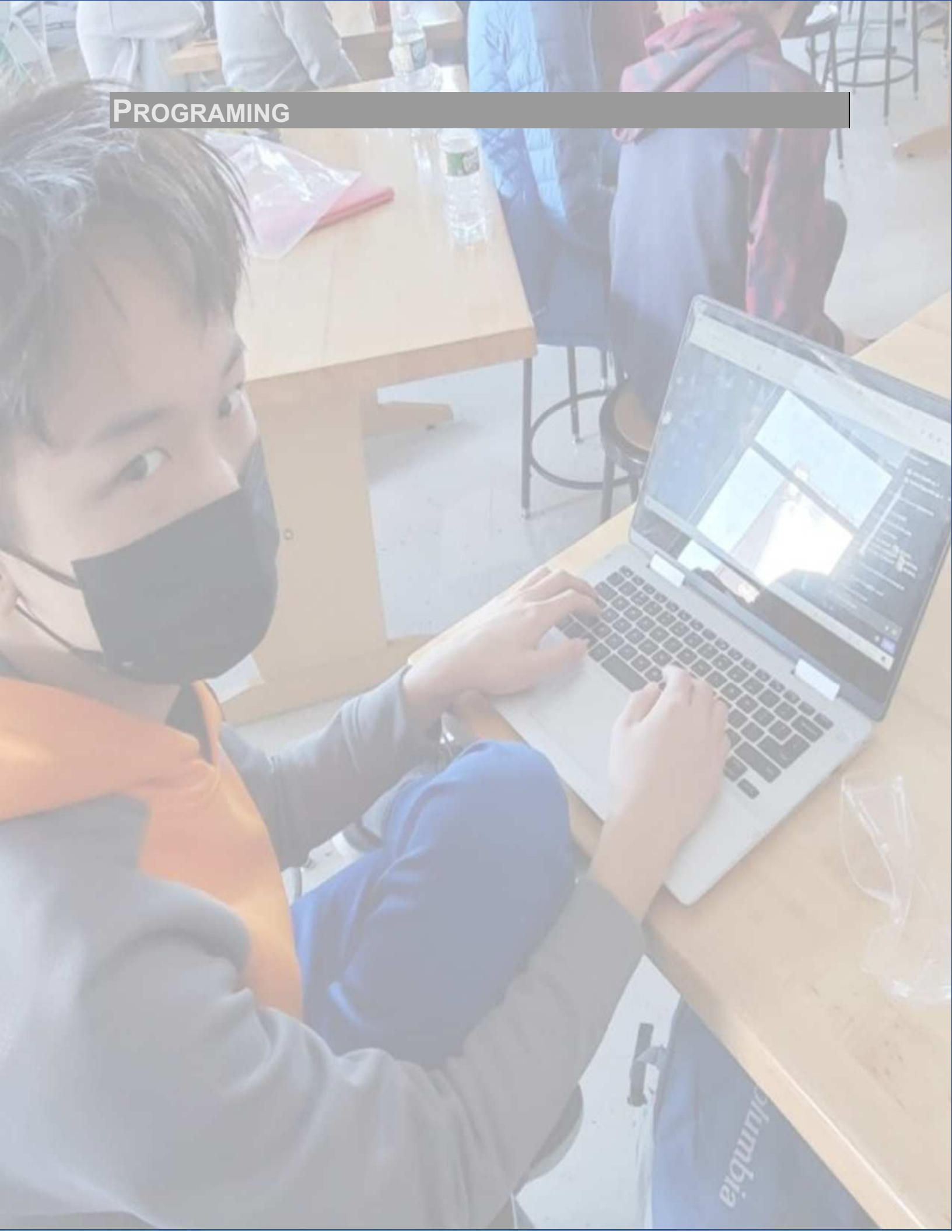
Overall the Cam design provides up to 9° of variability to our Shooter which in conjunction with shooter power helps to maximizes range and satisfactory backspin on shots. Within a a 3-4.5m arc around the Upper Hub our Shooter has a lab measured accuracy of 85%

CLIMBERS



We chose to go with a pre-Engineered system for our climbers using the telescoping arm system from West Coast Products. These climbers are extremely durable and our powered constant force springs in the extend and by a cable pulled winch in the retract. The arms are attached on pivots located as close as possible to the robot center of gravity.

PROGRAMING



Conceptually our code base is using a command / subsystem model. This provides separation between the individual elements of our robot and allows gradual testing to occur. Due to the low barrier for entry python provides, our robot is equipped with RobotPy. There are 5 subsystems in our robot code: drive, Shooter, indexer, Intake, and vision subsystems. Each of them can be removed and edited separately.

```python
self.odometry.update(
    self.gyro.getRotation2d(),
    self.frontLeftModule.getState(),
    self.frontRightModule.getState(),
    self.backLeftModule.getState(),
    self.backRightModule.getState(),
)
robotPose = self.odometry.getPose()

visionPose = Pose2d(
    *SmartDashboard.getNumberArray(
        constants.kRobotVisionPoseArrayKeys.valueKey, robotPoseArray
    )
)
weightedPose = Pose2d(
    visionPose.X() * constants.kRobotVisionPoseWeight
    + robotPose.X() * (1 - constants.kRobotVisionPoseWeight),
    visionPose.Y() * constants.kRobotVisionPoseWeight
    + robotPose.Y() * (1 - constants.kRobotVisionPoseWeight),
    visionPose.rotation() * constants.kRobotVisionPoseWeight
    + robotPose.rotation() * (1 - constants.kRobotVisionPoseWeight),
)
self.odometry.resetPosition(weightedPose, self.gyro.getRotation2d())
```

DRIVE TRAIN

Using wpilib built-in functions and a plethora of examples on swerve drive mechanics, getting our drive train running was a standardized process. On top of this, we update our odometer from wheel speeds in order to get our position on the field. We use a NavX sensor to get our robot orientation which is reset to align with the full field. This allows field relative movement and absolute rotation to be possible for a driver, reducing the mental load.

```python
if self.state == self.Mode.FeedForward:
    self.indexerMotor.set(
        ControlMode.Velocity,
        constants.kIndexerSpeed
        * constants.kIndexerGearRatio
        * constants.kTalonVelocityPerRPM,
    )
    self.stagingMotor.set(
        ControlMode.Velocity,
        constants.kStagingSpeed
        * constants.kStagingGearRatio
        * constants.kTalonVelocityPerRPM,
    )
elif self.state == self.Mode.Holding:
    if not (self.indexerSensor() == 0 and self.stagingSensor() == 0):
        self.indexerMotor.set(
            ControlMode.Velocity,
            constants.kIndexerSpeed
            * constants.kIndexerGearRatio
            * constants.kTalonVelocityPerRPM,
        )
        self.stagingMotor.set(
            ControlMode.Velocity,
            -constants.kStagingSpeed
            * constants.kStagingGearRatio
            * constants.kTalonVelocityPerRPM,
        )
    else:
        self.indexerMotor.set(ControlMode.Velocity, 0)
        self.stagingMotor.set(ControlMode.Velocity, 0)
elif self.state == self.Mode.Reversed:
    self.indexerMotor.set(
        ControlMode.Velocity,
        -constants.kIndexerSpeed
        * constants.kIndexerGearRatio
        * constants.kTalonVelocityPerRPM,
    )
    self.stagingMotor.set(
        ControlMode.Velocity,
        -constants.kStagingSpeed
        * constants.kStagingGearRatio
        * constants.kTalonVelocityPerRPM,
    )
elif self.state == self.Mode.Off:
    self.indexerMotor.neutralOutput()
    self.stagingMotor.neutralOutput()
```

INTAKE

Internally we hold a state machine for the Intake — deployed, retracted, off, and reversed — each of these states is then converted into the appropriate solenoid action and motor action. Commands are used on top to change between states on a button press.

INDEXER

The indexer system is coupled closely to the Intake, but functions separately. As with the Intake, there are multiple states — holding, off, reversed, and feeding to Shooter — one notable distinction is that the action of "shooting" a ball is controlled by the indexer, as the action isn't the shooter system firing but the indexer feeding the ball into the Shooter. Using the data from two break beam sensors, we can also determine if running the motors during our "holding" state is necessary if the balls are already where they need to be.

```python
def optimizeAngle(currentAngle: Rotation2d, targetAngle: Rotation2d) -> Rotation2d:
    currentAngle = currentAngle.radians()

    closestFullRotation = (
        floor(abs(currentAngle / tau)) * (-1 if currentAngle < 0 else 1) * tau
    )

    currentOptimalAngle = targetAngle.radians() + closestFullRotation - currentAngle

    potentialNewAngles = [
        currentOptimalAngle,
        currentOptimalAngle - tau,
        currentOptimalAngle + tau,
    ]  # closest other options

    deltaAngle = tau  # max possible error, a full rotation!
    for potentialAngle in potentialNewAngles:
        if abs(deltaAngle) > abs(potentialAngle):
            deltaAngle = potentialAngle

    return Rotation2d(deltaAngle + currentAngle)
```

SHOOTER

The Shooter can be broken up into three separate parts: the hood, the flywheel, and the turret. Using data from the vision system, we have relative angles that we can specifically rotate the turret to. Furthermore, using our robot's odometry, in the event that our vision system cannot detect the position of the hub, a fallback estimation can be used assuming an ideal system. This allows for progressive correction as we move around the field. Our turret will also only move when necessary as indicated by a ball ready to be fired in our indexer. Using sampled data collected from multiple points, a curve of our hood angles and wheel speeds are fed distances calculated from vision in order to have the capability to shoot from a variety of ranges.

VISION

The vision system is responsible for two main components: asserting the position of the odometer and determining hub distance and relative angle. Using our turret mounted Limelight 2+, a relative horizontal and vertical angle can be achieved. Using the vertical angle, our distance to the center of the target can be calculated and our relative horizontal angle can be derived using the direct values. Using the data from our gyro and the distance and relative angle of our turret and vision system, a positional estimation on the field can be achieved which can be used as a corrective measure progressively onto the odometer.

AUTONOMOUS

With our odometer position and each command, sequentially we can create a path using Pathweaver for our holonomic system and follow it absolutely with specific timing between waiting for each element. This is controlled by a sequential command group that determines the order in which to function; the autonomous aiming also allows for an easy and accurate shot from most of the fie

ENGINEERING TEAM

WHS Faculty Advisors

James Looney & Raul Madera

Team Captain

Eric Yamaguchi*

Team Business Lead

Lizzy Collins*

Team Technical Lead

Charley Marsland

Senior Mentors

Dwight Meglan | **Chris Aloisio°** | **Steve Harrington°**

Mentors

Anthony Gelsomini | **Manny Barros°** | **Sean Lendrum°** | **Mark Holthouse** *Mentor Emeritus*

STUDENT MEMBERS

Safety Captain

Finn Prendergast*

Lead Programmer

Luke Maxwell

CAD Lead

Landon Bayer

Ivan Cai | **Kevin Chin** | **Tyler Dong** | **Mit Ganson**

Joey Gundal | **Jason Guo** | **Momin Humayon*** | **Baili Jiang**

Jacob Kaplan | **Declan MacDonald** | **Liam McWeeney** | **Vincent Milinazzo**

Alex Ng | **Jeffrey Pan** | **Nina Pappas** | **Claire Peng**

William Qu* | **Rachel Qu** | **John Santosuosso** | **Ali Tariq**

Alex Theofilou | **Jessica Wang** | **Ray Wang**

Westwood Robotics, Inc
Board Of Directors

Steve Harrington° *President* | **Catherine Tseng** *Treasurer* | **Manny Barros°** | **Cheten Gopal** *Clerk* | **Jack Tseng**

**Denotes High School Seniors* | *°Denotes FRC Alumni*

SPONSORS

Heartlander Surgical	Prime Motor Group	Plymouth Rock Assurance	Fidelity
Universal Robotics	Roche Brothers	Mathworks	Deadham Savings
		Needham Bank	

VENDORS

Andymark	West Coast Products	Vex Pro	Swerve Drive Specialties
Discord	Google Work	Onshape	Robo Promos
McMaster-Carr	The Home Depot	Cross the Road Electronics	MS Visual Studio
Python	GitHub	RobotPy	Java