MVRT 115
TECHNICAL BINDER

MONTA VISTA HIGH SCHOOL | CUPERTINO, CA

2026

MONTA VISTA ROBOTICS TEAM 115

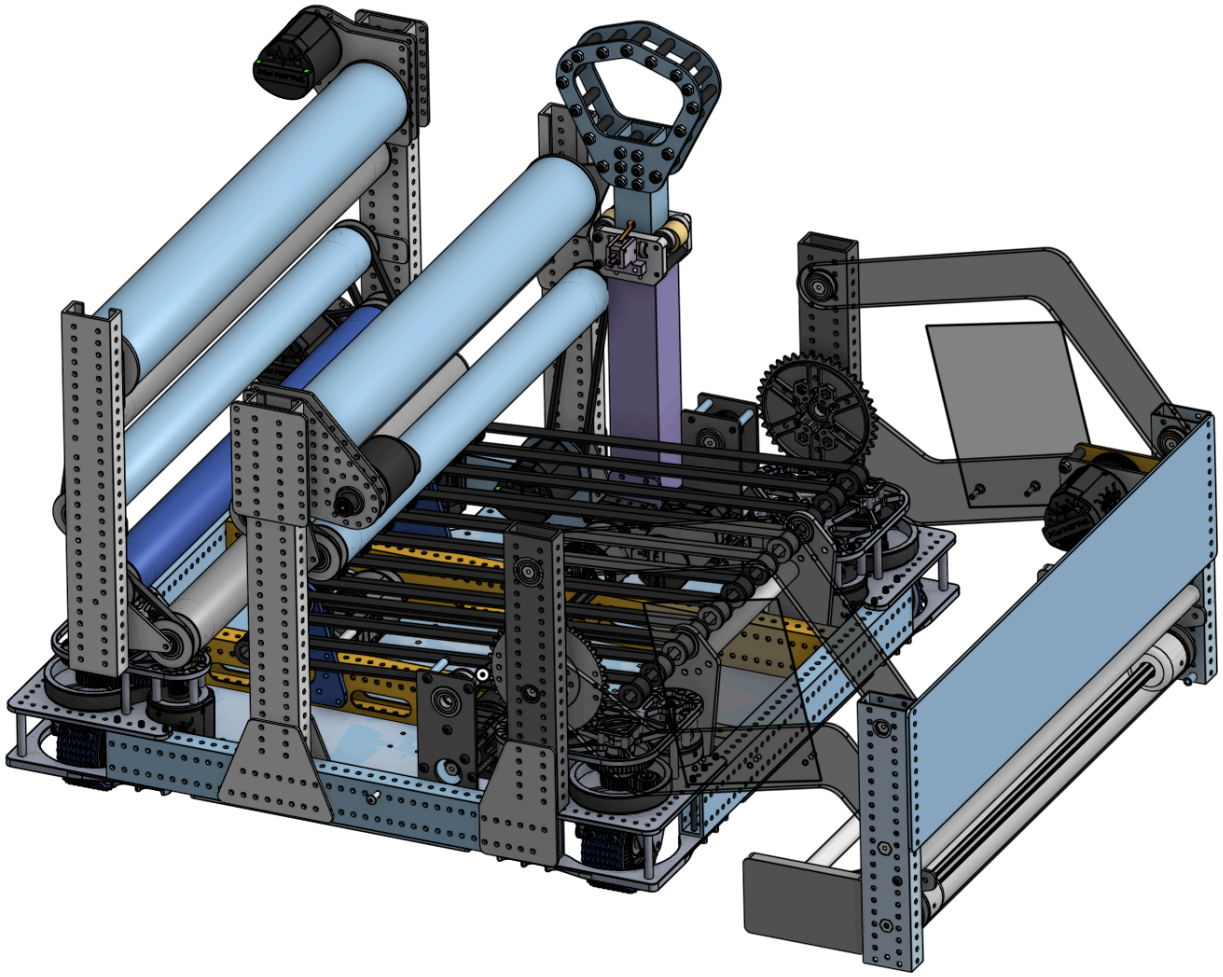


TABLE OF CONTENTS

ANALYSIS 2
- Strategy 2
- Priority List 3

DESIGN 5
- Drivetrain 5
- Intake 6
- Shooter 7
- Climber 7
- Hopper 7

ELECTRICAL 8
- Control Systems 8
- Controller Area Network 8
- Wire Management 8
- Sensors 9
- Camera 9

PROGRAMMING 10
- Swerve Drive 10
- Computer Vision and April Tags 10
- Autonomous Programming 12
- Custom Keypad /MacroPad [Operator Controller] 13
- Elevator 13
- Coral Mechanism 13

ANALYSIS

Strategy

Understanding that success in this game depends on the speed and accuracy alliance members can work together to score fuel in the hub, MVRT prioritized engineering a robot that can efficiently score fuel and be able to pass fuel back to our alliance zone if needed, allowing us to be compatible with wherever our alliance members are able to score fuel. To ensure speed and consistency, our vision system allows us to automate coral scoring, reducing variability in our performance.

Autonomous

- We strategized to be able to score from the sides of the field to be able to fill our intake from the midfield twice in one autonomous period to maximize our fuel scored to gain the advantage going into the teleoperated period

Teleop

- The most valuable task is to efficiently pass fuel into our alliance zone during the periods where hub is inactive, to be able to start scoring

Endgame

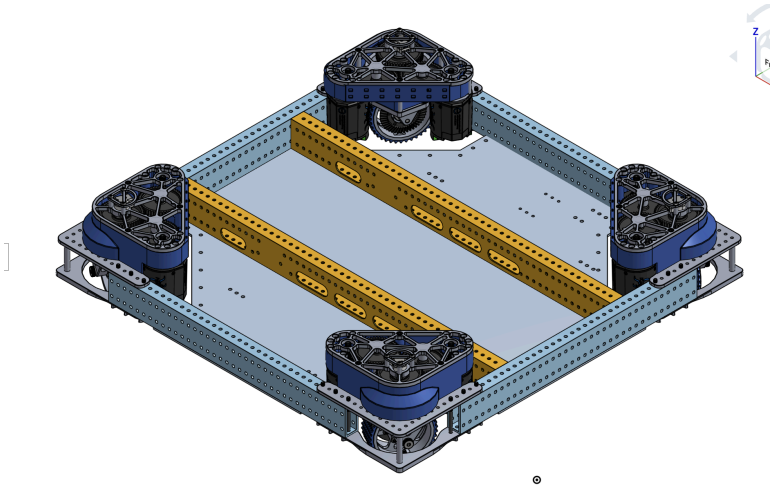
- Our strategy is to continue cycling fuel till the end, as we realized the climb didn't give sufficient points, and would rather try to score more points to ensure a win during our matches

Priority List

1. Drive
    a. Mobility
    b. under trench
    c. Over bump
2. Shooter
    a. Shoot multiple balls at once to score balls quicker
    b. Fixed angle
    c. Odometry alignment
    d. Shoot from anywhere in our alliance zone
3. Fuel manipulation
    a. Fast intake + maximize storage
    b. Efficient indexer to feed from intake to shooter
4. Vision
    a. Updates field position to correct for any odometry issues when shooting
5. Climb
    a. While holding ranking point value, not many teams climb/have an L3 climb, making it impossible to get the ranking point
    b. Easier + more points scored from doing another cycle

DESIGN

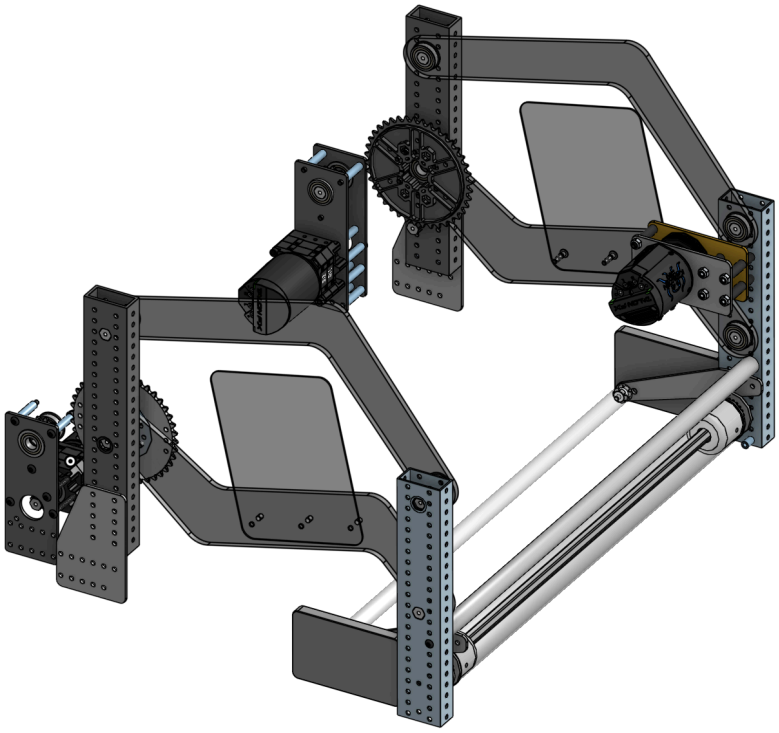
Drivetrain



Our drivetrain is designed to be able to compete with speed and good mobility. Our Mk4i swerve modules utilize Kraken X60 to drive for good speed and mobility while we use Falcon motors for quick steering and mobility. Using the original Mk4i setup, allows us good clearance to the ground, reducing the chance of the bot getting beached on fuel scattered on the ground, should we cross over the bump to get to midfield. The clearance allows the robot to get off the fuel extremely easily.

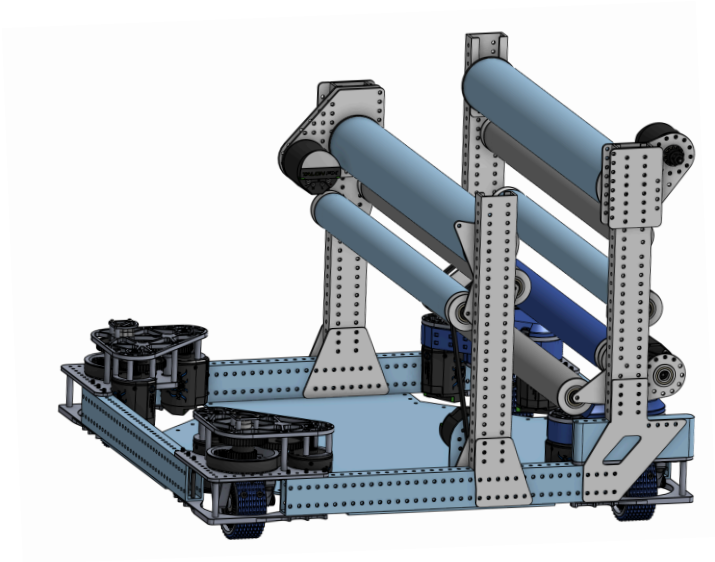
Last year we spent time prototyping swerve guards, creating a heavy 3D printed custom guard, but then ultimately using custom bent polycarbonate for swerve guards. However this year, we decided to use different 3D printed swerve guards made specifically for the Mk4i module we use, in our team color; purple.

Intake



Our intake consists of a 4-bar mechanism to get the ~6-inch balls onto our bot. We have 4 customized polycarb plates with a specific stretched 'Z' shape and a 'C' shape on the top plates so that at the beginning of the match, the intake can be fully compressed, also leaving enough room for 8 preloaded balls while not being too large to take any room away from the hopper and shooter. This shape also allows for maximum extension, giving the most room for the hopper and also staying under the height and extension limit before the match starts. The bottom pivot polycarbonate on the intake is designed to hold panels to prevent the ball from going in between the 2 arms, as we found during testing. Another issue we solved after testing was the ball getting stuck between a 2x1 and the bumper as we lifted the intake back up. This was solved by a simple 3D print to guide the ball back towards the middle. Our intake consists of one big polycarbonate motor attached to a motor that will spin the balls into the robot. The second roller is near the bumper to help guide the ball upward. When creating our Layout Sketch on CAD, we discovered this small gap that would have stalled the balls and could create further issues. The front polycarb plate is made as an extension to the hopper so that the balls do not escape that area. We decided to make aluminium plates instead of 2x1s to simplify the design and use less total space. We found that the more space we have, the more balls we can hold, which will lead to fewer cycles of driving back and forth.

Shooter

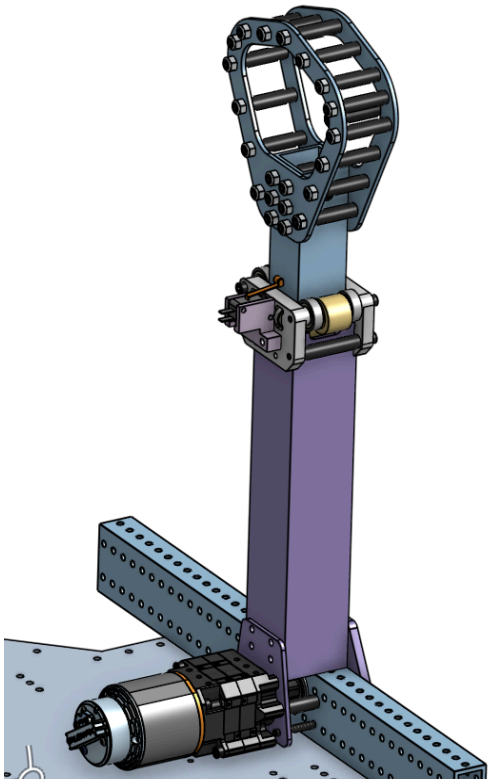


]-Our ***fixed-angle shooter*** has ***two robot-wide steel flywheels***, and ***6 polycarbonate rollers*** to feed the fuel into the flywheels. The flywheels are made of steel to ***increase the moment of inertia*** (which would ensure we can recover the energy lost to shooting the balls quickly) and to make sure the flywheel doesn't bend in the middle (we cannot have it vibrate). We decided each flywheel should get its own motor because we want to ensure the energy lost after shooting a ball is regained as fast as possible to ensure consistency (so we need ***maximum possible power for each flywheel***). We decided to make the shooter ***robot-wide*** because we wanted to ***increase the throughput*** of fuel and didn't want to deal with indexing the fuel. The polycarbonate rollers are connected to each other via belt and pulley, and the flywheels are run by two separate motors via chain and sprocket. We used dead-axles instead of live axles because it made our design and assembly much simpler. The compression of the balls varies between the rollers with the flywheel compression being ~1 inch.

We initially made a prototype shooter, where the shoot angle was 25° and each of the flywheels were connected to their respective motors via two 60T gears. For our final shooter, we ***reduced the shoot angle to 20°*** because it allowed us to ***shoot more consistently***

from closer distances . We also switched from two 60T gears to chain and two sprockets because the 60T gears were out of frame.

Climber



Our climber uses a ***2 in. to 1.5 in. telescoping tube system*** to achieve vertical extension while ensuring that we stay within frame perimeter.. The extension uses constant force springs to allow the inner stage to move up as the rope is unwinded.

A ***UMWPE rope*** is anchored at the top of the climber by tying it to a screw through an aluminum spacer. It then runs down through the telescoping structure and wraps around a hex shaft of the output stage at the base of the climber, where it is secured with a shaft collar. As the robot climbs,

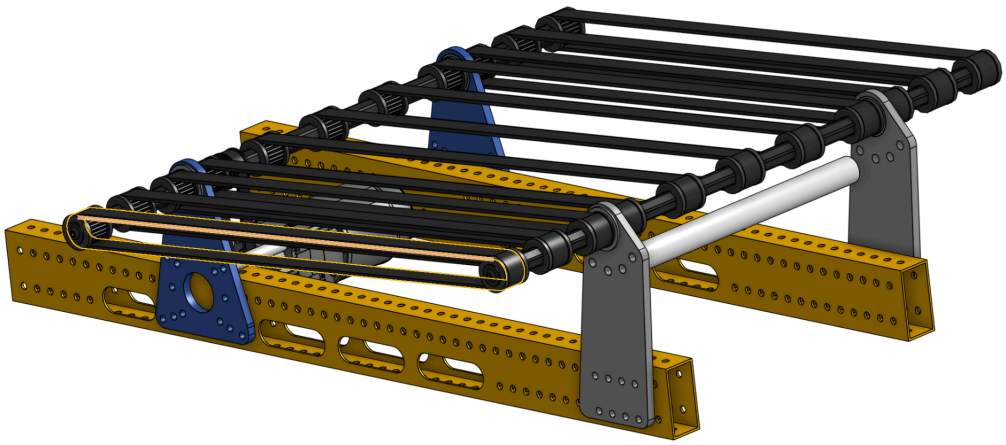
the motor rotates the shaft, pulling the rope inward and drawing the inner telescoping tube downward, which raises the robot off the ground.

The winch is driven by a motor through a ***5:1 × 5:1 × 3:1 gearbox***, providing a total reduction of ***75:1***. This high reduction allows the system to generate sufficient torque to reliably lift the robot while maintaining controlled climbing speed.

At the top of the climber, a ***custom hook geometry*** was designed to maximize area we can engage with the hooks. The widened hook profile increases the capture envelope, allowing the robot to successfully latch onto the bar even with minor alignment error during approach.

The motor input stage and the gearbox output stage are mounted to the bellypan to ensure rigidity. This transfers climbing forces directly into the robot's chassis.

Hopper



Our hopper consists of eleven timing belts on two 19” hex shafts to create a conveyer belt on the bottom of our robot to help feed balls into the shooter. We considered both an active and passive hopper, but decided that having an active way to funnel balls into our shooter would help with consistency and reliability, as opposed to having a passive feeder, which could be prone to jamming. The eleven timing belts and their respective pulleys on the two hex shafts, including the one for the motor, are attached through four ¼” plates. These four plates are attached to our crossrails used to make our robot more structurally stable. This was not originally what we had planned, though. Our first prototype consisted of four 2x1 aluminium tubes that were attached to the bellypan instead of the crossrails. The main issue of this design was only having one timing belt on the outer sides of the tubes, making a long section of the hex shafts unsupported. This caused the hex shaft to sag in the very middle, creating the risk of the hex shaft snapping during competition. This has now been fixed by making each section of the hex shaft shorter. This also caused the tubes to move into the middle of the robot more, where we ultimately decided to switch to plates in order to attach it to the crossrails. We also have many polycarbonate plates surrounding our robot in order to contain all of the balls, which have been cut in a way where the circuit breaker and other needed electronics are still accessible.

ELECTRICAL

Control Systems

- The robot is wired with the latest generation of control systems, including the ***Power Distribution Hub, Mini Power Module, CANivore, RoboRIO 2.0, and the Pigeon 2.0***
- We utilized the Pigeon 2.0, a simple yet powerful device that improves swerve coordination and driving. With an instant boot-up without requiring calibration, it receives precise data that allows for steady movement with little to no drift.

Controller Area Network

- We use a CTRE CANivore device to have a separate CAN bus for the drivetrain and for the manipulators.
- The CANivore increases the reliability of the system by ***reducing the load on both CAN buses***, while also helping ensure that even with a CAN error in the manipulators, the drivetrain is still functional
- The ***swerve CAN Bus, which starts from the RoboRIO***, is terminated through the resistor at the PDH
- The ***second CAN Bus that starts from the CANivore*** is terminated through a 220Ω resistor at the coral manipulator. It is terminated here in order to reduce the number of wires going through the cable carrier while also minimizing the chances of failure in this CAN chain.

Wire Management

- Our robotics system implements a compact yet effective wiring approach, where each ***wire is carefully made to connect each component with the perfect amount***, resulting in clear and neat paths. Terminals in the PDH are clearly labeled for easy identification and maintenance if needed.

- The wires from the intake, as well as the outermost shooter motors, are carefully wrapped in flexible electrical pipe to seamlessly route the wire while protecting against chafing and cuts to the wire as the robot is used.
- ***Swerve subsystems are wired identically*** with curated extensions in a fashion that allows for modules to be interchangeable and switched in the case of one of them failing. This allows for quick maintenance and neatness during competitions.
- To ensure that wires passing through the crossrails do not get damaged in the constant movement during a match, ***there are 3D printed protectors that clip into the crossrail holes to prevent the wires from rubbing against sharp metal edges.***

Sensors

- Each swerve module is equipped with the latest magnetic encoders (CANcoders) that communicate through the CAN bus to send precise rotational and velocity data to allow for consistent auto and teleoperated movement.
- The climber subsystem is equipped with a ***limit switch*** at the bottom of its range, used to calibrate the motor's range of motion while also serving as a failsafe, preventing the motor from pushing the climber past its limits.

Camera

- Throughout the season we tested several different cameras--including the Arducam OV9281, but after having various issues we decided to use ***LimeLight 4*** due to their reliability.
- We have one LimeLight currently mounted on our bot. This LimeLight is mounted on a side bar that is high enough to see past our intake while it is lowered, but is also under the 22" limit set by the trench. The purpose of this LimeLight is to see the April Tags and align the robot so that the shooter is as accurate as possible.

Scouting Applications (App, Dashboard, Bot)

MVRT has built its own ***custom scouting app and dashboard from scratch*** to compile and organize scouting data gathered by our team's scouts during each regional, as well as a ***personalized Discord bot*** to view and summarize the information. The scout app and dashboard are programmed using TypeScript, while the Discord bot is programmed with Python. Our apps are all connected to a single database (Google's Firebase), where all of our information is stored. The data compiled here is retrieved by our Discord bot and used by our Strategy team, both for alliance selections and to ***determine our playstyle heading into each match***. We update all 3 at the beginning of each build season according to the new game, and refine it throughout the season based on the preferences of our Drive and Strategy teams.

Our dashboard is publicly available at https://mvrt115-scout.web.app/dashboard!

Swerve Drive

MVRT uses its state-machine system to make an implementation of Phoenix6's CommandSwerveDrivetrain code, with several modifications to not only give us more control in the customizations, but also modifying the system so it aligns with the rest of the state-machine system.

Throughout most of the teleoperated phase of the game the swerve is completely in control of the driver. However, when the driver intends to score fuel into the hub, the state machine will force the robot to be angled towards the hub, giving the driver less responsibility on aiming and more capability in shooting when at a comfortable place.

Computer Vision and April Tags

Computer Vision System Structure:

Odometry refers to the process of ***estimating the robot's position (pose) on the field over time by tracking wheel rotations and using other onboard sensors*** like gyros. In our case we use two pose estimations and compare them with each other with a common filter to determine our current position on the field. Firstly we have robot odometry, which ***uses the swerve modules' encoder and gyro data from our drivetrain to track the robot's movement and update its pose.***

The other pose estimate we use is gathered from our Vision software. Using our new Limelight camera, we ***gather information from April Tags*** and its ***distance and orientation to determine where we are on the field using Megatag 1 and 2 algorithms.***

Finally, given our two estimates, we run them through the common filter, which ***compares the two odometries and returns the best result on where our bot is on the field.*** This is so we can reliably combine our estimates into one filtered pose and trust both of our estimates simultaneously; the robot odometry is useful when we can't see an April Tag, and the vision odometry is useful to iron out errors from robot odometry such as drift, bad carpets, and colliding with robots and structures. The pose from the common filter is used in our Autonomous phase as well as for our shooter flywheel velocity calculations.

Autonomous Programming

With the introduction of the new state machine system, we've also implemented a new system for autonomous paths. Using the odometry from our common filter, we use ***pose estimation assisted by vision to accurately traverse different paths.*** The individual points on the paths are stored in an action list, which also contains parameters such as max velocity and PID for steady rotational and translational movement. Each item in the action list is either passed through using a timeout duration or when the next target position is reached, for instance for the shooting command.

While we also have some functionality with PathPlanner, the system we used before, because of the more flexibility the new system gives we've ultimately decided to use this new system. Like PathPlanner, the new system will also orient itself to adjust for either the red or blue alliance, so we don't need to make separate points and paths for each alliance. Each point on PathPlanner is completely analogous to the points on our new system, which makes our new system just as, if not more useful than PathPlanner alone.

Climber

In our codebase, ***different phases of the climber's motion are represented by a state***: Disabled, Idle, Travelling, Holding, or Homing. To illustrate, Travelling indicates that the climber is moving, Holding ensures that the climber does not move due to gravity or other factors, and Homing returns the climber to the zero position.

MVRT's climber uses PID, MotionMagic, and FeedForward control to ensure ***consistent, smooth, and efficient motion.*** The climber motor's voltage is calculated using a PID loop, along with Feedforward to account for gravity and static friction, and MotionMagic to reduce jerk.

In addition to these motion controls, using DigitalInput from a limit switch on the bottom of the climber's path signals when the climber has reached its zero position, ***ensuring positional accuracy.***

Shooter

MVRT's shooter consists of two motors dedicated to the flywheels as well as two motors dedicated to feeder wheels. When shooting, the feeder wheels run at a constant speed while the shooter flywheels spin at a speed dictated by multiple factors.

Using the distance from the hub found by Vision and the velocity of the drivetrain, we run the distance and the velocities through a calculator that returns a target velocity for the flywheels. We make sure the motor reaches this calculated speed through a trapezoidal profile kP calculator, ***keeping our flywheel speed consistent and ensuring it does not oscillate,*** allowing a consistent trajectory for a fuel shot anywhere on the field.

In addition to the kP calculator, our team uses state-based programming to maximize shooter score. In the REVVING state, ***feeder wheels will not send balls to the flywheels until the flywheels reach its target speed.*** Once target speed is reached, the state changes to SHOOTING.

Shooter programming also accounts for belts in the robot's hopper, whose motor turns at a constant velocity controlled by PID and velocity FeedForward. In the SHOOTING state, the belts roll fuel toward the shooter's feeder wheels consistently. In case fuel gets stuck at the bottom of the shooter, the UNJAM state calls for the belts to roll backward and unblock the fuel in the hopper.

Intake

MVRT's intake consists of an arm and a roller. There are two motors for the arm, with the left motor following the right motor. Additionally the arm motors use ***DifferentialMotor to help move in sync***, so that the intake arm does not tilt or slant itself while lifting or lowering; as well as the curved PID profile Motion Magic provides in order to move smoothly. Alongside PID, the arms use FeedForward controls ***to account for gravity and other predictable factors that may cause inconsistencies.*** To account for fuel slowing down the

roller, we also run a kP profile on the roller, ***allowing us to run at a steady velocity and thus getting a more stable flow of fuel*** from the intake into the hopper.

Throughout the match the intake arm frequently moves up and down. With our state machine, depending on what state the robot is currently in, the intake will go to a predetermined angle using encoders, which we zero at the start of the match when the intake is up. When we want to shoot fuel into the hub (in the REVVING or SHOOTING state), we raise our intake arm so the fuel in the deadzone in our intake fall nicely into the hopper, ***clearing our robot from balls and score more fuel overall.*** When the robot is not in the REVVING or SHOOTING state, the intake will be lowered so we can constantly collect fuel for our next cycle.