MVRT 115
TECHNICAL BINDER

MONTA VISTA HIGHSCHOOL | CUPERTINO, CA

2026

MONTA VISTA ROBOTICS TEAM 115

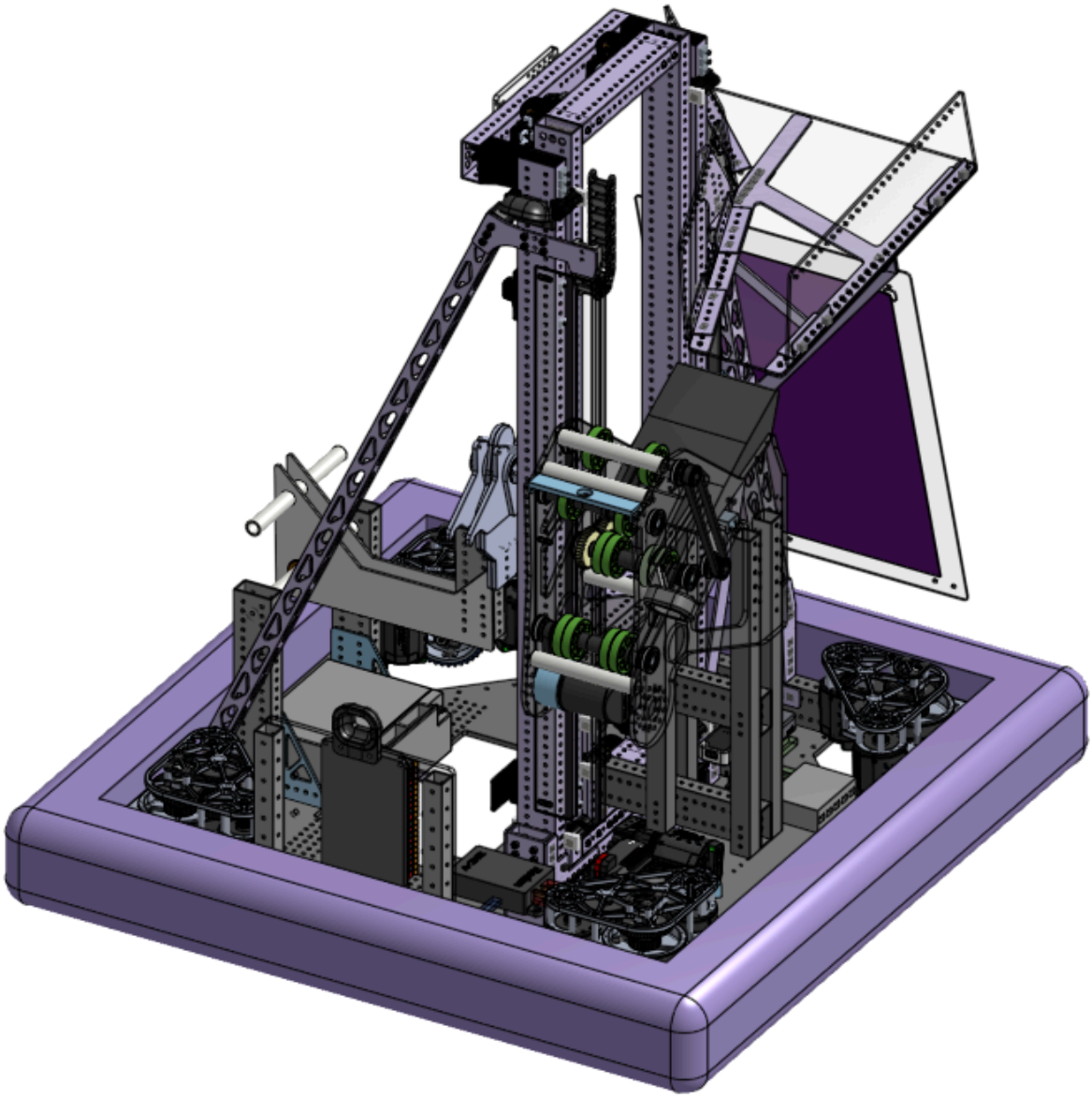


TABLE OF CONTENTS

ANALYSIS

Strategy

Understanding that success in this game depends on the speed and accuracy alliance members can work together to score coral onto the reef, MVRT prioritized engineering a robot that can efficiently score coral on all four levels of the reef, allowing us to be compatible with wherever our alliance members are able to score coral. 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 coral efficiently and effectively in the reef whilst exiting in the autonomous period.

Teleop

- The most valuable task is to efficiently score coral on the reef, starting with the L4 branch in order to maximize points.
- We planned early to ***prioritize efficient cycles*** of coral for our main plan
    - This led us to prioritize our ***coral mechanism***, alongside creating an advanced ***computer vision system*** for localization and automated alignment.

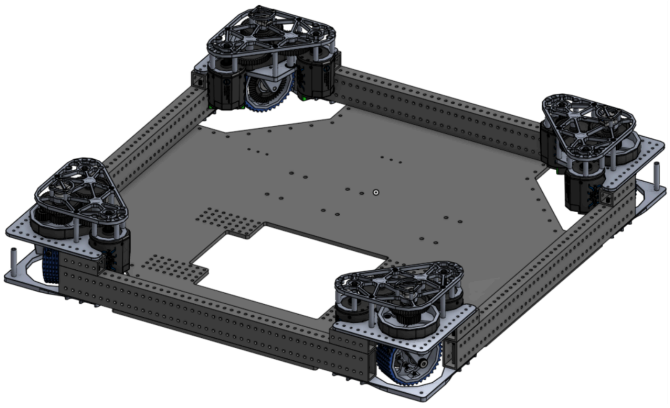
Endgame

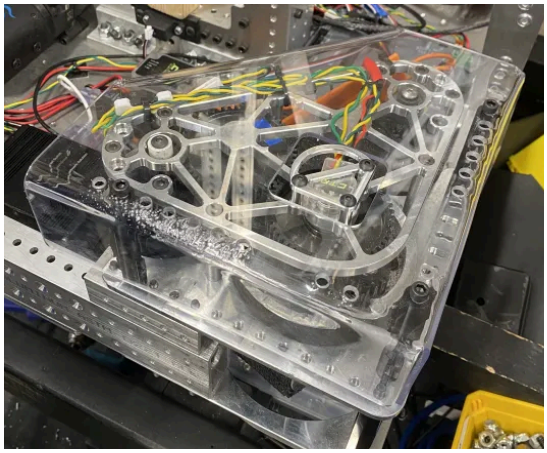
- Our strategy is to ***deep climb*** quickly and efficiently to ensure our alliance receives the climb RP

Priority List

1. Drive
    a. Mobility
    b. Park
2. Elevator
    a. Reach different heights (Coral Station, Processor, all four branches of Reef)
        i. Flexibility to score and reach whatever height is needed
3. Coral Scoring/Manipulation
    a. HP use ramp on robot to feed coral to manipulator
    b. Coral scoring most important for earning points
    c. Coral earns more points during AUTO
4. Vision
    a. Alignment using vision allows efficient and accurate scoring
    b. Maximize number of coral scored per match
5. Deep climb
    a. Earns the most points out of all game actions
    b. Important for Barge RP
6. Ground Intake Algae
    a. Scarcity with algae; most will be on the ground of the field
    b. Have to be able to pick up algae to score it
7. Algae Processor Scoring
    a. Scoring into Processor important for Coopertition bonus
        i. Lowers Coral RP threshold
    b. Other than for Coopertition, any algae scored in the processor is an additional opportunity for opposing alliance's HP to score algae into their net
8. Algae removal from reef
    a. Algae needs to be cleared in order to score more coral on reef
9. Algae Net Scoring
    a. Point value is equivalent to a L3 branch coral during teleop

Drivetrain

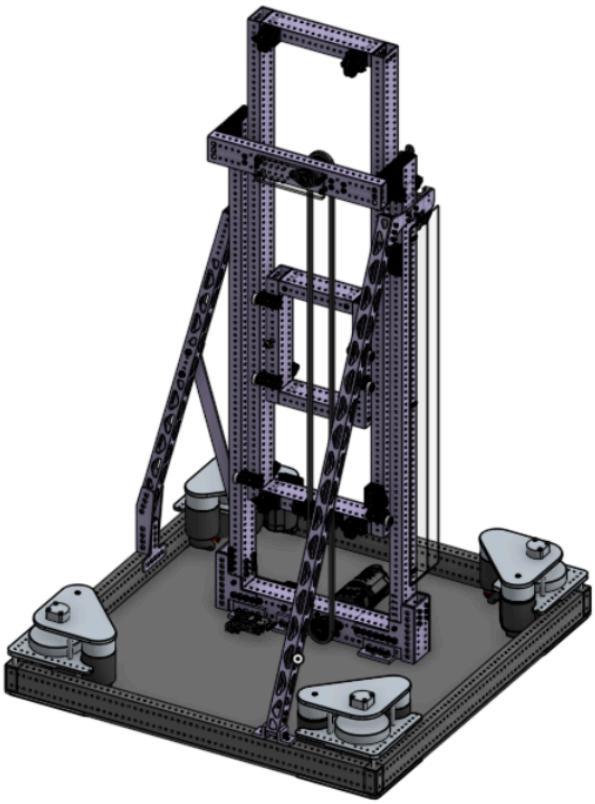




Our drivetrain ensures that we are fast and mobile. With Kraken motors, we are able to move quickly and effectively, and our Mk4i modules allow us to maneuver around tight spaces with ease. We ***modified the original Mk4i design, decreasing the height of each module by one inch to ensure we keep our center of mass as low as possible***. This also has the added benefit of having a low belly pan and bumpers, ***reducing the chance of getting stuck on an algae***.

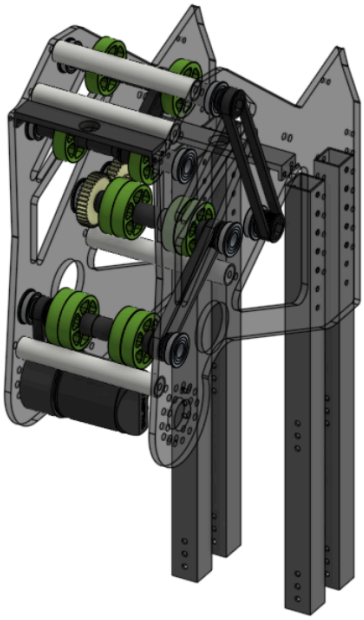
We spent time prototyping different swerve guards to protect the swerve modules from falling coral. We started out with 3D-printed custom guards, but ended with ***bending polycarbonate for swerve protection***, which provides protection while still ***allowing us to see potential problems as well as the CANcoder light***.

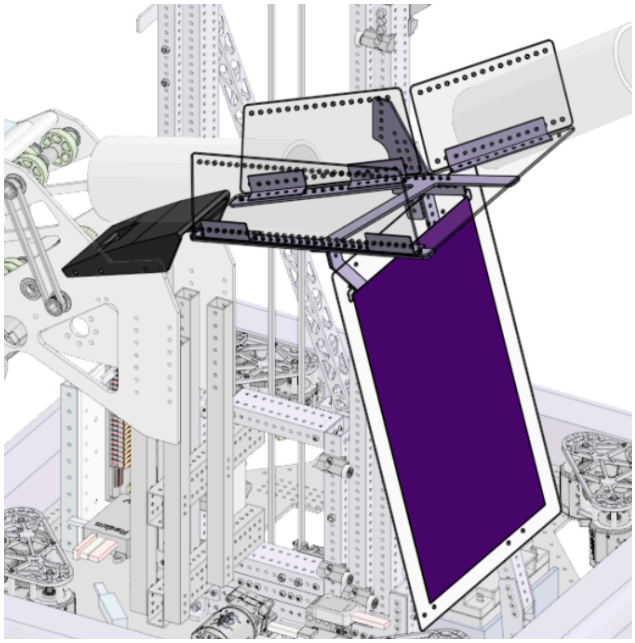
Elevator



Although our elevator is a WCP elevator, having it mesh with the other parts of our robot was difficult since bellypan space was dwindling. By starting just under the maximum starting robot height and setting our coral mechanism slightly higher than the carriage, ***we were able to use a 2 stage elevator to reach all 4 levels of the reef***. Stabilizing the elevator was difficult, since our bellypan space was taken up by electrical components and a climber design (now removed). In addition to the L-brackets mounting it to the belly pan, there are also tubes and gusset reinforcements on either side at the base. There are also ***2 support 1x1s that constrain movement in either direction***, which have been pocketed to minimize weight.

Coral Mechanism and Ramp



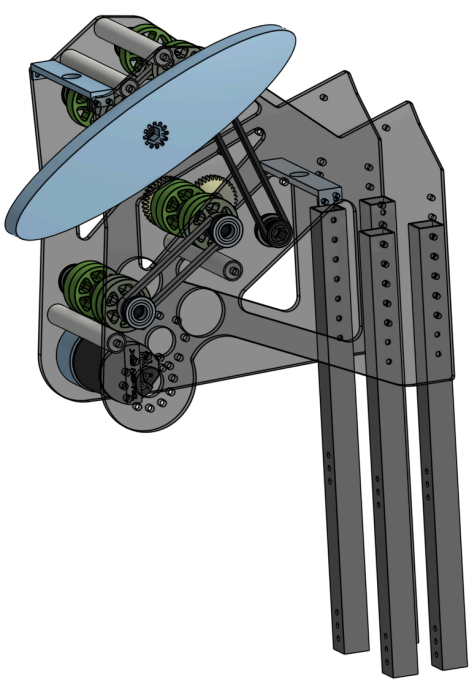


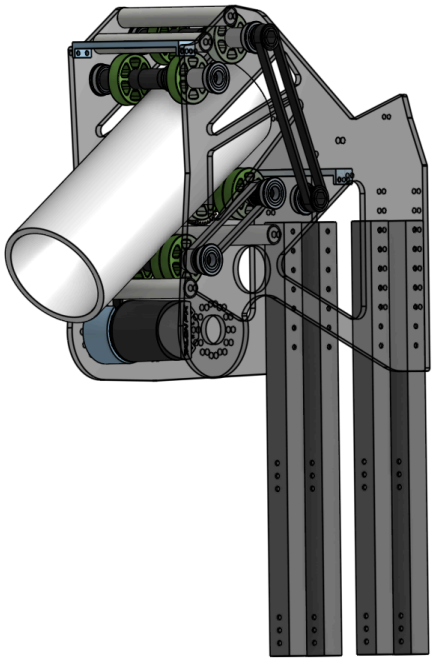
For our coral mechanism, we decided to use upper and lower rollers, to ensure that the coral flows into the mechanism smoothly. We also decided on ***using compliant wheels so that the wheels will compress when the coral comes into contact***, along with gripping the coral so that it doesn't slip out. The top wheels are placed directly above the bottom wheels, so when the coral exits at an angle it is still supported by the bottom wheels and rotates around them to safely change from our mechanism's angle to the reef's angle. Additionally, ***the wheel spacing was carefully chosen to fine-tune the alignment*** that our ramp cannot.

For the ramp, we decided to have it start out wide, and then narrow down, so that when guided by the "rails" it would enter into the coral straight and centered, while also making sure that there is as large a space as possible for the coral to go, making it easier on the human player. ***The ramp is made almost entirely of polycarbonate and has minimal supports to save weight***. It is also extremely effective at reorienting coral no matter which direction it enters, allowing easy compatibility with HP players from other teams.

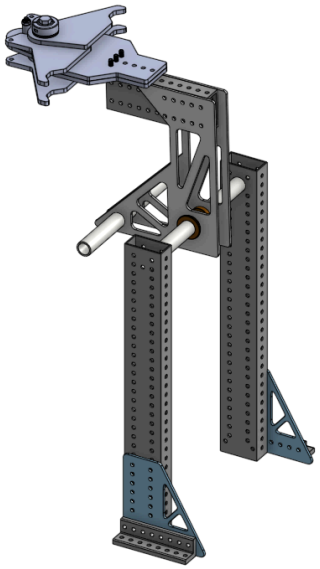
Algae Knocking Mechanism

Inspired by Team 972's algae-removing mechanism, we tested a similar oval design on our robot that uses the existing coral mechanism's motors to spin, removing algae while remaining within the robot's weight limit. We ***experimented with different lengths***: 11 inches and 13 inches, and different placements, testing each design's strength and power. While testing, we noticed that our 3D-printed oval flipper did not have enough friction to get the ball off, so we added sandpaper to its edges. Ultimately, our design was not as efficient as intended and algae removal was low on our priority list, so we removed it from our robot--instead replacing the mechanism with software that ***knocks the algae off*** by extruding the coral out of our mechanism.





Climber Mechanism



Our first climber design involved pushing the robot up from the bottom of the cage using two opposing plates, with a hole in our belly pan to allow the cage to pass through the bottom of our bot. Unfortunately, this design took up too much space, was very heavy, and did not work, so we decided on another design.

The concept behind our current climber design is similar to many other bots, pulling the cage and rotating it to pull the robot away from the ground. This design is much lighter by pocketing material to save weight when possible and takes up much less bellypan space. ***To attach to the cage, at the end of a pivot is a mechanism that allows entry into the cage but prevents the cage from slipping out***. The W-shaped gussets also help with alignment.

ELECTRICAL

Control Systems

- The robot is wired with the latest generation of control systems, including the ***Power Distribution Hub, Radio Power Module, Mini Power Module, CANivore, RoboRIO 2, 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 systemic and neat wire approach, where major and minor power ***wires are routed separately in a "traffic" lane fashion*** with clear and neat paths. The ends on the PDH are clearly labeled for easy identification and maintenance if needed.
- The wires on the elevator going to coral are carefully wrapped in flexible electrical pipe to seamlessly route the wire while protecting chafing and cuts to the wire as the

robots are 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.
- Wires for the ***subsystems on the moving part of the elevator are kept on a sturdy IGUS cable carrier*** to ensure that they do not get damaged in the constant movement during a match

Sensors

- The ***coral subsystem is enhanced with several proximity sensors*** to ensure that it intakes and outtakes efficiently, while also guaranteeing that the coral piece is properly stored while moving.
- The elevator subsystem is also equipped with ***magnetic limit switches*** that are perfectly aligned with the bottom and top of its range. This is used to ***calibrate the motor's range of motion while also serving as a failsafe***, preventing the motor from pushing the elevator beyond its limits.
- 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.

Camera

- Throughout the season we tested several different cameras--including the Arducam OV9281, but after having various issues we decided to use ***Microsoft Lifecams*** due to their reliability.
- We have three Lifecams on our robot: one for computer vision (at the front of our robot), one for viewing the cage and climber during the endgame, and the other to make sure our robot is flush against the reef while scoring.

Orange Pi

- Starting this year (2025) we have switched from using an ASUS MiniPC to an ***Orange Pi*** for our Co-processor for our computer vision localization & alignment system.

We're using a Redux Zinc-V for voltage regulation for the OrangePi, which runs the PhotonVision server for our vision system. We tried both 2D-vision and 3D-vision methodologies and eventually implemented 3D-vision.

LEDs

- We use ***individually addressable LED lights*** to communicate with the drive team about the current status of our robot. With a unique pattern for each action, ***they let the drive team know when it is safe to perform critical actions*** such as driving away from the human player station and scoring coral pieces. A portion of the LEDs also indicate the battery percentage, which lets the drive team know when practicing when to switch out the battery.

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 Phoenix6's CommandSw.erveDriveTrain as our base swerve drive, with added logic for heading correction and speed control.

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, this is handled automatically by the Phoenix6 SwerveLegacyDrivetrain, which uses the swerve modules' encoder and gyro data to track the robot's movement and update its pose.

On the other hand, **Localization** is the broader task of determining the robot's exact location on the field using all available data. While our odometry provides a crude estimate, our localization ***combines this estimate with external references—like AprilTags—to correct for drift or error***. MVRT's localization system runs in the background to enhance our robot's position estimate by detecting AprilTags on the field and adjusting the position estimate accordingly.

MVRT has a localization system constantly running in the background that provides us with ***a constantly updated estimate of our robot's position*** relative to the rest of the field.

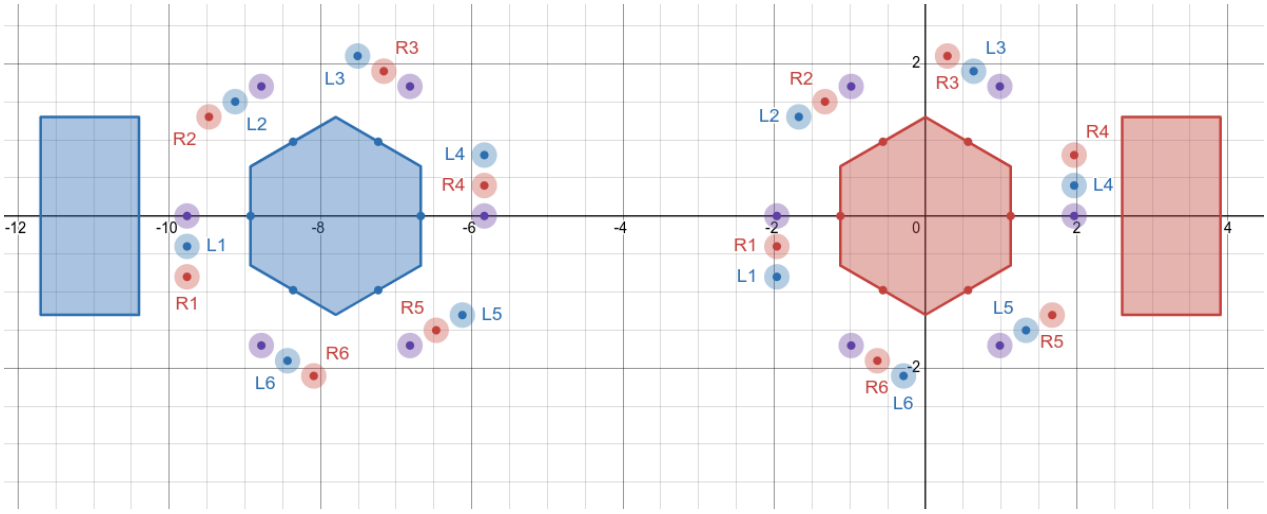
Localization.java: The subsystem for localizing on the field. Constantly running in the background. Updating odometry is automatically handled by our Phoenix6's SwerveLegacyDrivetrain, and so this subsystem solely updates our position based on AprilTags. Below is the main method we use to update our cameraEstimator (a position estimator based on vision alone). Later the cameraEstimator's values are merged with our main poseEstimator (which includes odometry), to give us a final poseEstimation.

```
/**
 * Credit: photonlib-java-examples
 * @return estimated robot pose based on AprilTags
 */
public Optional<EstimatedRobotPose> getEstimatedVisionPose() {
  Optional<EstimatedRobotPose> visionEst = Optional.empty();
  List<PhotonPipelineResult> results = camera.getAllUnreadResults();

  for (var change : results) {
    visionEst = cameraEstimator.update(change);
  }
  return visionEst;
```

```
}
```

Alignment: Alignment is handled through a SwerveRequest in Localization.java, which provides instructions directly to our drivetrain using the outputs of our vision system. We have added logic such that when a button on our Macropad is clicked, the robot ***automatically aligns itself*** with either the left or the right of the nearest Reef, depending on the input from the macropad.



Math for Alignment positions: our team had fun figuring out some hexagon math with the help of ***Desmos*** to calculate the positions on the left/right of the reef we should align to based on the offset of our camera from the center of our coral manipulator (and a certain distance from the edge of the reef)

https://www.desmos.com/calculator/pbvqe59wt5 Our math can be viewed here! :)

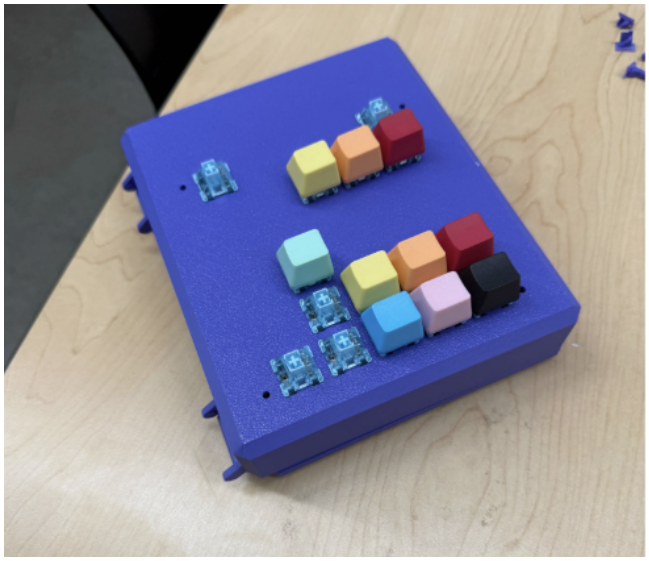
Autonomous Programming

MVRT uses the PathPlanner GUI to draw out each auton path. Computer vision is incorporated into each of our paths, allowing them to be dynamic while still maintaining a high degree of accuracy. During each path, our vision system allows us to ***accurately realign ourselves with the reef*** as we move to score each piece of coral, compensating for any drift

that may have occurred as a result of mechanical inaccuracies and ensuring consistent outcomes.

Our drivetrain is configured using two separately tuned PID controllers during the autonomous period, one for translational velocity and one for rotational velocity, to further ensure smooth motion and consistency. Each path is converted into a .json file that is stored in our codebase, which is then converted into a Command at the beginning of the autonomous period. Each path we draw is ***designed from the perspective of the driver station, allowing it to run on both the red alliance and the blue alliance side without making any software changes.***

Custom Keypad /MacroPad [Operator Controller]



As MVRT shifted toward prioritizing vision, we've come to the conclusion that the xbox controllers we've used up until now simply do not have enough buttons to accommodate our Operator's needs. So, ***this year*** we decided to build a ***custom macropad, from scratch***. Our macropad houses ***up to 30 keys*** and uses Adafruit's 5x6 Neokey PCB as its base. This PCB is wired to a Raspberry Pi Pico, which serves as its processor and is flashed with firmware from QMK, an open source keyboard controller repository. All of this is stored within a custom case that we CAD'ed and 3D printed and can open with a latch on the bottom. Each

of the keys on the keyboard can be configured to blink or shine at any arbitrary color, and are mapped to a unique joystick ID. We use ***7 of these keys*** for our vision alignment routine, as well as a few others for manual controls, including elevator set points.

Elevator

MVRT's elevator builds off of the knowledge we've gained throughout the past few years. Pairing our experiences working with elevators from FIRST's 2023 CHARGED UP season and with the "state"-based programming structure we developed during FIRST's 2024 CRESCENDO season has allowed our elevator to operate ***highly consistently***.

In our codebase, ***each "state represents a different phase of the elevator's motion***; for example, we have states such as "Zeroing", indicating that the elevator is currently retracting to its lowest position, "Zeroed", indicating that it is currently resting at the lowest position, "Towards_Goal", indicating that the elevator is currently in the process of extending, and more. We update this state using commands bound to different joystick buttons. To translate this state into instructions for the elevator motor, we periodically check the current state using a switch statement, and calculate + set a motor speed accordingly.

This speed is calculated using a PID loop to ensure that the path the elevator follows is both ***smooth and speedy***. This calculation is further paired with the outputs from a FeedForward control system that ***compensates for the additional voltage needed to overcome static friction, gravity, and more.*** This combination is what enables our elevator to move at a ***high degree of accuracy*** regardless of what we want it to do, whether it be extending, retracting, or staying completely still at an extended position.

Coral Mechanism

MVRT's coral manipulator software is heavily centered around the on/off states of the 2 proximity sensors located on the manipulator. Our rollers are set to spin while a coral is detected by the first sensor but not the second, ensuring that it is secured within the manipulator. Based on different heights, we set different speeds to the rollers when outtaking the coral, as trough requires a slower outtake speed compared to the L4 coral height. The use of enum states and boolean values for proximity sensor states allows us to keep track of the coral's location and orientation, ***working in sync with the elevator*** to allow us to score at any height we want.