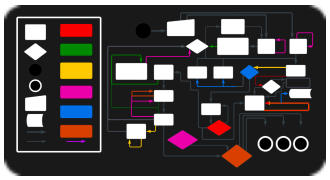
Control Award Sponsored by Arm - Wolfpack Machina 18438

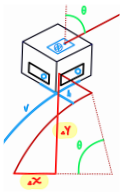
Autonomous Objectives



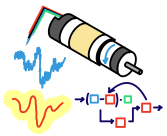
State machine diagram of our autonomous detailing our 11 types of failsafes. See the full diagram at our pit, in our engineering notebook.

Our objective was to create a high-scoring and reliable Auto by adaptively scoring 6 cones on the safe or contested high junction each match and reacting to interference from other robots. We concluded this was the **most consistent** way to end up in a picking position (due to tie-breaker one) and be strong in eliminations. We met these objectives using our key algorithms to **increase our speed and redundancy** in the most critical parts of the auto, and our array of sensors to **adapt to different field conditions** by triggering our many fail-safe routines. With these fail-safes, our Auto can handle 248,832 possible permutations of the game.

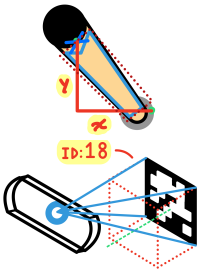
Sensors Used



Two **Rev Hex encoders** and the Control Hub's **Gyro (IMU)** combine to give us odometry that's accurate to 2cm over 1800cm



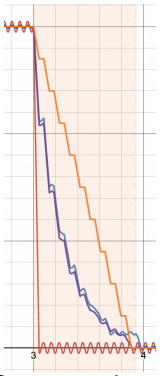
Eight **Motor Encoders** and **Current Draw Sensors** give us information about the state of our motors, which we use for everything from controlling our slides in Tele-Op to triggering fail-safes in Autonomous



Two **Logitech Webcams** enable us to determine the distance to junctions to automatically aim our depositor, and use the 'AprilTags for OpenCV' library to identify the correct parking zone in the autonomous phase

Two **Infrared Break-beams** detect cones in our claw (for automatic intaking and failsafes) with high reliability and faster polling than I²C sensors

This year, we **focused on using digital and built-in sensors**, like encoders, break beams, and current draw sensors, because our testing found them to be dramatically faster than I²C sensors, improving the accuracy of our control algorithms. We found we can get the **same information more efficiently using these sensors** than we can with more complex sensors. For example, we use current draw sensing to detect when our intake slides have hit the cone stack in Auto, break-beams to assist the drivers in Tele-Op by automatically closing the intake claw at the right moment, and encoders to detect if a mechanism is stuck and activate failsafes, for instance the unjamming sequence of our slides.

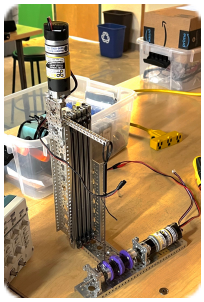


Desmos testing of 3 different noise reduction filters

To improve the usefulness and reliability of our sensors, we **implemented several different signal noise reduction filters**, such as Rolling Average, Low-Pass, and Alpha-Beta. We test each filter to determine the best option for each application. These filters eliminate erroneous results, smooths our control of mechanisms, and overall increases accuracy and reliability by **improving signal-to-noise ratios** and ensuring the precision of our sensor reading reflects their true accuracy.

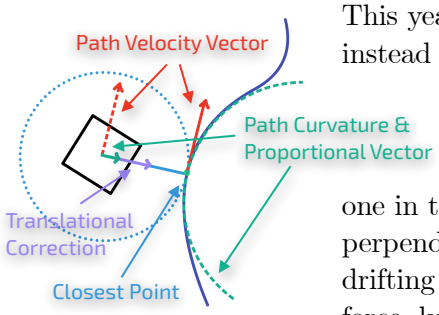
Key Algorithms

We start development by **analyzing multiple ideas** using Desmos animated graphs and other computer simulations. We then create an **initial proof of concept** for the most promising ideas using our robot or testbed models. Finally, we implement code from the best solution into our competition code branch. This multi-step process - based on the **industry standard of design and code reviews** - allows us to discover potential issues early on in our development process when they are far easier to fix. This saves us time and increases the **reliability and maintainability** of our code - and we're super proud of the algorithms this has let us create:



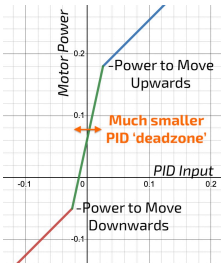
Testbed models like these slides allow us to develop and test in a controlled environment

Custom Spline-Based Movement:

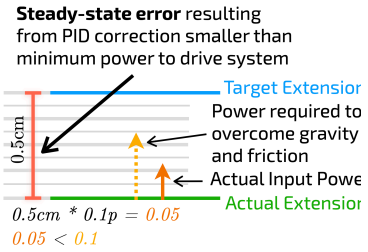


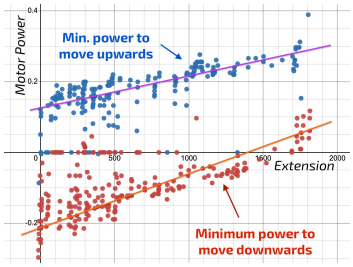
This year, we developed a **fully custom navigation and movement system** instead of using a pre-packaged library. We use two-wheel arc odometry to localize the robot and a dynamic movement function that can **follow any parametrically defined path** (lines, arcs, bezier curves, etc.), **leveraging lambda functions** to easily pass in complicated paths. Driving direction is the sum of 3 vectors: one in the direction of the velocity vector at the point along the path closest to the robot, one perpendicular to the direction of travel proportional to the distance from the path (to correct for drifting off the path), and one proportional to the curvature of the path (to model centripetal force, keeping us on the path through tight curves).

Measured Feedforward PID:



Measured feedforward **fixes steady-state PID errors** without complex analytics. By logging the motor power it takes for the subsystem to move at different positions, then taking a regression of these data, we are able to create a function that allows tiny errors to correspond to enough power to overcome static friction and correct the error. This effectively eliminates steady state error, allowing our depositor slides accuracy to improve from ~12.0 to ~0.5 ticks of error on average.





Datapoints from automated CSV logger and line of best fit used to approximate response of slides system

Vision Auto Aim:

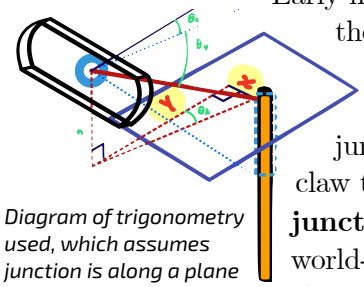
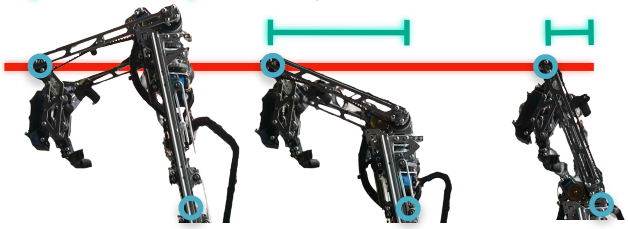


Diagram of trigonometry used, which assumes junction is along a plane 'h' cm below the camera

Early in the season, we were brainstorming ways to align the robot with the pole when we realized that **looking down through the cone was a great way of aiming**. We developed a system that first uses a camera and the OpenCV vision library to detect the junction and then uses the known variables of depositor height, claw tilt angle, camera field-of-view to **find the distance from the junction** (essentially, transforming camera perspective-space pixels to world-space coordinates). This makes aiming incredibly dynamic and accurate, **handling tipped junctions** and **allowing for a wider range** of robot positions.

Analytical Inverse Kinematics:



Even though it extends from the robot different distances, the endpoint of the depositor stays at the same height

The proof-of-concept of our Auto-Aim system moved the entire robot to align with the junction, however the high inertia of the robot (and thus response delay) made aiming very slow. We realized that by **syncing the rotation of the depositor arm with the extension of the slides**, the depositor claw could extend outwards from the robot while maintaining the same height. Using Desmos to test and model our equations, we developed an inverse kinematics controller that could find the necessary arm and slide positions to extend a given distance from the robot. The addition of a turret gave us the extra degree of freedom needed for **full 3D control of the position of the depositor claw.**

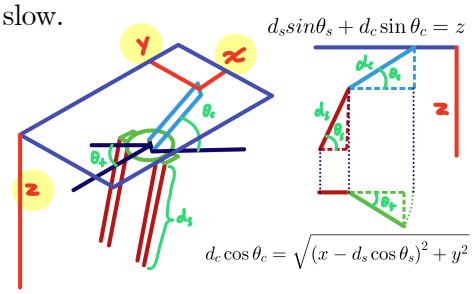


Diagram of our system and the basic systems of equations we solved. Slides in red, turret in green, and arm in blue.

Driver Controlled Enhancements

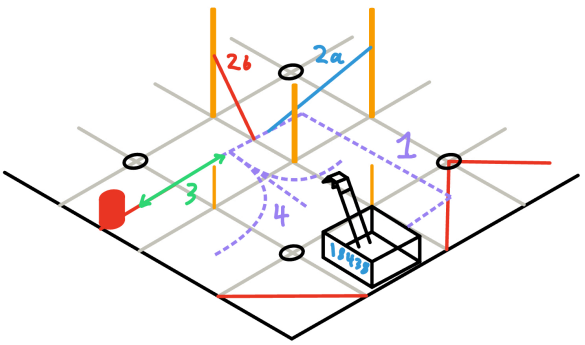
Our **driver enhancement philosophy** is driven by **human factors engineering**: making controls intuitive to **reduce cognitive load on the drivers** and allow them to focus on strategy of the game. For starters, our robot control is field centric, so input is relative to the driver, not the robot. Taking the concepts from our custom movement algorithm, we also model the effective path the robot is taking to determine the "curvature" and centripetal force to keep the robot on the path. Thus, directional driver input will always cause the robot to move in that direction, regardless of any existing momentum, making the robot more controllable at high speed.

Furthermore, our driver inputs follow this same philosophy: each **button maps to an action or preprogrammed sequence** built with the intention of automating subsystems without taking any control from the drivers. Driver inputs can easily override automated controls such as automatic intaking (utilizing break-beams) and adjust preset depositor positions on-the-fly (using Measured Feedforward PID).

Engineering Portfolio References

See pages 11 and 15 in our portfolio and 44-56 in our engineering notebook.

Autonomous Program Diagram



1. Move to scoring position
2. Score our preload cone on either the safe (2a) or contested junction (2b)
3. Extend our intake slides, using current detection to sense the cone stack (We then repeat steps 2 and 3 to score the rest of the starter stack)
4. Park in the zone indicated by our AprilTag signal sleeve