

SOFTWARE DESIGN PROCESS



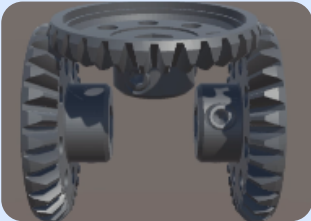
BRAINSTORM



What **challenges** did we face in previous seasons?

Brainstorm solutions while collaborating with design subteam for sensor implementation


PLAN & SIMULATE



Create **simulations** to test theoretical algorithms implementation

Enables **concurrent workflow** with design team and limits errors

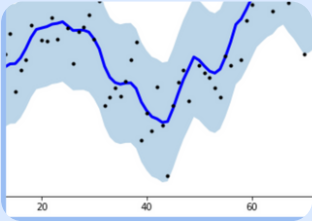
TESTING



Over **multiple trials**, test speed and consistency of both individual and full subsystem controls

Compare improvements of newly-coded algorithms to previous methods

ITERATE



Evaluate options and **reiterate**; consider complexity and benefits

Connect with industry mentors to learn new algorithms

AUTONOMOUS

OBJECTIVE: Score both pre-loaded pixels and cycle from the pixel stack

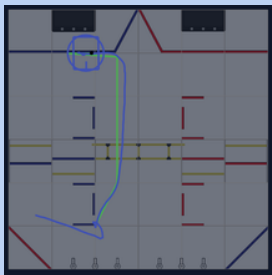
PURE PURSUIT ALGORITHM

Problem: Needed robot to **follow efficient paths** that could be quickly developed

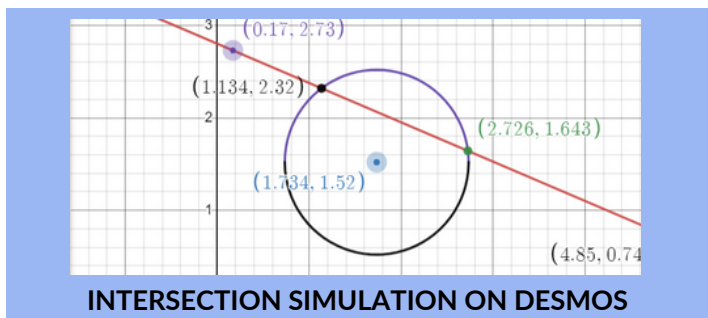
Initial Solution: Roadrunner path library was **limited to basic trajectories** and relatively slow development time

Solution: Created a custom **Pure Pursuit algorithm** for efficient paths and streamlined development

- Programmed pathfinding algorithm that generates optimal pathfinding between waypoints
- **Created a Desmos simulation** to test our derived equations for finding intersections
- Used **another custom simulator** to evaluate the waypoints of our trajectory.



Pure Pursuit allows us to generate complex curved robot paths (**blue**) with simple waypoint lines (**green**)



$$i_x = \frac{r_x + mr_y - bm + \sqrt{(-m^2 r_x^2) + (2mr_y - 2bm)r_x - r_y^2 + 2br_y + lm^2 + l - b^2}}{m^2 + 1}$$

$i_x = 2.72640916626$

$$i_{2x} = \frac{r_x + mr_y - bm - \sqrt{(-m^2 r_x^2) + (2mr_y - 2bm)r_x - r_y^2 + 2br_y + lm^2 + l - b^2}}{m^2 + 1}$$

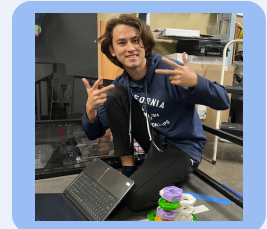
$i_{2x} = 1.13408000532$

INTERSECTION EQUATION

IMPACT 1.5x faster path following than Roadrunner library; allows for more **complex autonomous**

JEREMY'S STORY - PURSUING THE PERFECT AUTON

In the past, we would only ever use Roadrunner to code our autonomous paths. But after last year, when the only limiting factor in our auton cycles was speed, I knew that something needed to change. After brainstorming with the team, making simulations in Desmos, and spending late nights for testing and tuning, we finally created our own custom PurePursuit algorithm. I can safely say that this autonomous is the best one we've ever coded by far.



EXTENDED KALMAN FILTER

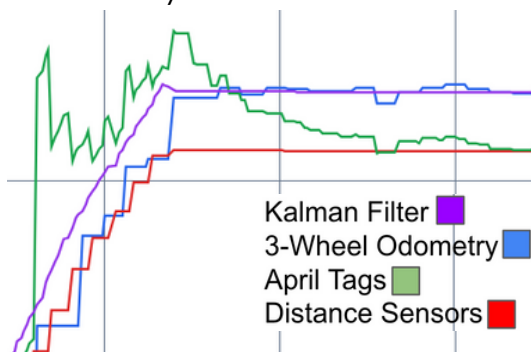
Problem: Need to know **exact position** of robot to execute autonomous paths

Initial Solution: Used Odometry and IMU, but experienced severe drift in robot's position estimate

Final Solution: Combined multiple sensors (Odometry, IMU, Distance Sensors, AprilTag Vision) in an **extended kalman filter**

- Used non-linear regression and a moving window to **fuse all localization methods** while ensuring that outliers were eliminated

Impact: Reduces localization error to **less than 2 cm** when driving across the field, allows robot to reliably drive to stacks and cycle in autonomous



KALMAN FILTER RELOCALIZATION

EXTENDED KALMAN FILTER SENSOR OVERVIEW

FRONT OF ROBOT

BOTTOM OF ROBOT

LINEAR SLIDE PID

Problem: Using the built-in controller was **too slow**; Extending with full power was inaccurate

Solution: Implement **custom trapezoidal motion profiling with PID** to optimize speed and control

Impact: Extension time improved by 22%, with accuracy within less than half a centimeter of target position

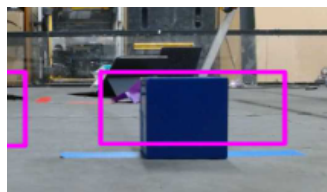
- Slide extension time is the limiting factor when cycling, this 22% improvement leads to a **full cycle improvement** of 12.5%



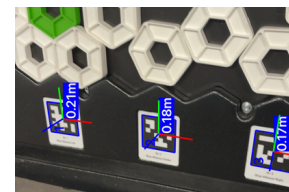
CUSTOM PID PROFILE

EASYOPENCV AND APRIL TAGS

- Tested for memory leaks and optimized build times in **EasyOpenCV Simulator**
- Needed April Tag and Team Prop pipelines for localization and autonomous objectives



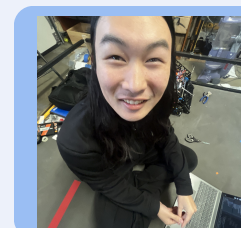
CUSTOM PIPELINE SIMULATION



APRIL TAG SIMULATION

AIDAN'S STORY - KALMAN FILTER CONFUSION

Two heads are always better than one. That's usually how the saying goes, but combining four sensor inputs with our Kalman Filter was a lot harder than expected. Since each sensor has its own noise and poll time, things become complicated. After noticing that the Kalman Filter was lagging behind, we realized our moving window was a whopping 300 ms-- too much old information. The Kalman Filter can be confusing, but every day, we learn more about why it works the way it does.



TELEOP

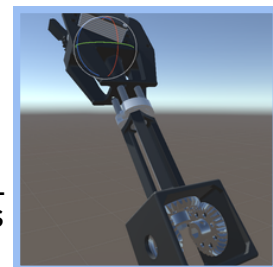
OBJECTIVE: Accomodate for adaptable gameplay with simple controls

DIFFERENTIAL INVERSE KINEMATICS SIMULATION

Problem: Original "box" outtake couldn't consistently create mosaics because of imprecise pixel placement

Solution: Accurate control of differential claw

- Derived servo power using **Inverse Kinematics Simulation**
- Found rotational positions of gears based on desired claw position

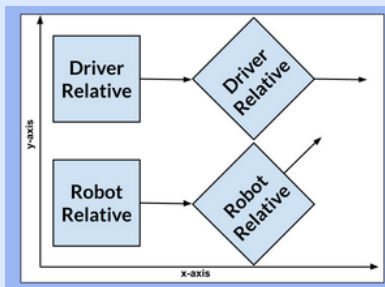


DIFFERENTIAL SIMULATIONS IN UNITY

DRIVER-RELATIVE CONTROLS

Problem: Basic mecanum drive is hard to control (confusing without clear view of robot)

Solution: Driver relative robot controls provides streamlined driving experience. Vertical joystick input translates to movement along an axis, rather than the forward direction of the robot

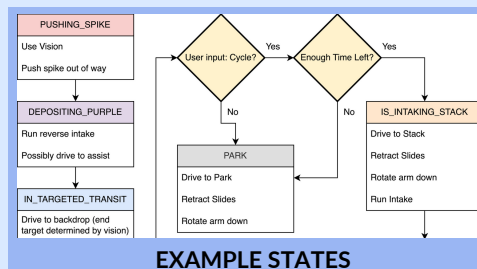


IMPACT Easier drivetrain control reduces cycle times by 1 second.

STATE MACHINES

Problem: Actions across multiple systems are completed linearly, or one at a time

Solution: State machine contains goals that each subsystem **pursues simultaneously**, enabling control of multiple subsystems at the same time



EXAMPLE STATES

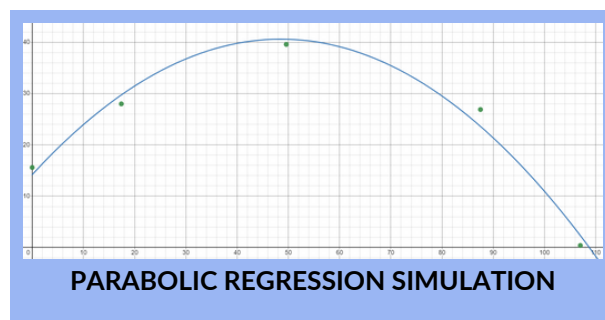
IMPACT Minimized dependencies on other subsystems, decreasing yellow pixel deposit time by 2 seconds

DRONE TRAJECTORY SIMULATION

Problem: Drone launcher only had one preset angle, requiring a specific launch position

Solution: Automatic, machine-learning based drone launcher angling based on robot position

- **Trained** parabolic regression loss model on dataset collected via drone launcher videos
- **Validate** and **adjusted** model output based on real-life testing
- **Tested** final model angle outputs based on distance to zone (input)



PARABOLIC REGRESSION SIMULATION

IMPACT Reduced the complexity of endgame controls, resulting in ~8% faster endgame objective completion

VIHAAN'S STORY - SCORING AUTON POINTS IN ENDGAME

Fold, load, and launch. Rinse and repeat. After countless drone launches, I finally accumulated enough data to train our machine learning model. We could now represent the drone flight path with a graph and return the optimal launcher angle based on the robot's position on the field. The drone might be counted as endgame points, but I now consider it to be 100% autonomous.

