# mumble

## OnyxTronix #2231


FIRST ISRAEL ישראל

2231

תיכון שוהם Shoham High School

מועצה מקומית שוהם

# mumble

## OnyxTronix 2026 Robot

2231

REBUILT

PRESENTED BY HAAS

***Science can amuse and fascinate us all, but it is engineering that changes the world***

***- Isaac Asimov***



2231

REBUILT™

PRESENTED BY HAAS

***Science can amuse and fascinate us all, but it is engineering that changes the world***

***- Isaac Asimov***



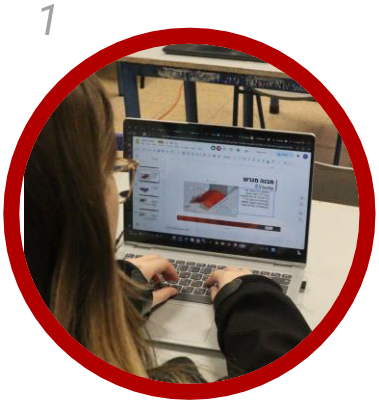
2231

REBUILT™

PRESENTED BY HAAS

# Engineering Process

1



**Game Analysis**
QFD - defining and prioritizing the robot's actions

Beginning of week 1

2

**Prototyping**
validating ideas and finding optimized setups

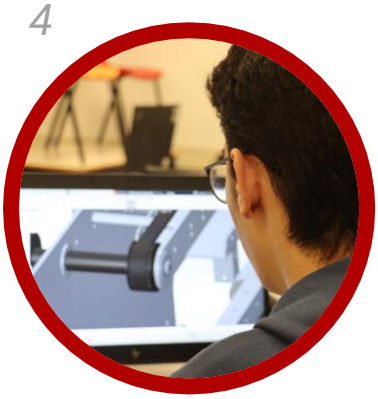
Week 1 – 5 Days

3

**PDR**
evaluation of systems' prototypes and integration into a robot

Summary of week 1

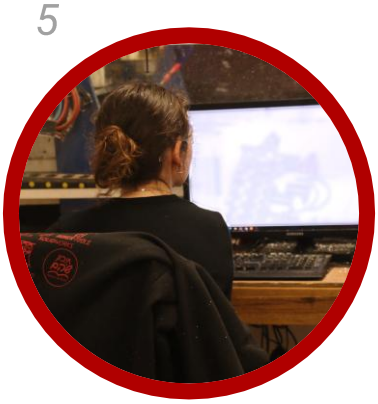
4



**CAD**
Developing robot 3D design model and integrating the different systems

Summary of week 2

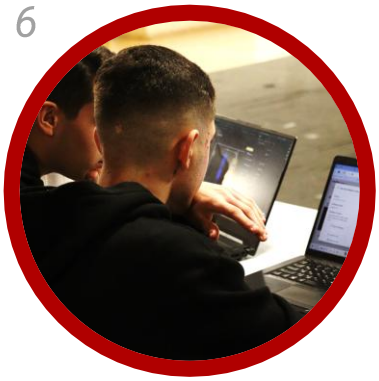
5



**Manufacturing & Assembly**
manufacturing robot β in-house, while assembling it

Complete production and assembly in 5 days, after 19 days robot β is assembled and wired.

6



**Testing & Modeling**
mechanical tests on **robot-β** and modifying model to improve **robot-$\alpha$ design.**

Finished modeling robot $\alpha$ in the end of week 4

7

**Manufacturing & Assembly**
Outsource and in-house manufacturing with assembly of robot $\alpha$

Throughout week 6

8

**Testing, improving**
fine tuning the different systems

Until the competition – changes\upgrades\additions in robot $\alpha$

# Game Analysis

## *QFD – Defining, prioritizing and brainstorming*

## 1 Defining and prioritizing robot capabilities

| Fundamentals<br>Basic Abilities | Must Have<br>High Priority | Advantage<br>Mid Priority | Nice to Have<br>Low Priority |
|---|---|---|---|
| Shooting on the move, while facing defense | Intaking and shooting simultaneously | Efficient Loading Station intake | Multi-side intake capability |
| Collects FUEL behind the Climb | Expandable hopper | Depot Intake capability | Buddy Climb |
| Full-width intake system | Full alliance zone | Reliable endgame climb | |

## 2 Prototypes' Brainstorming

After reviewing past seasons, we realized reaching a world-class level required a definitive Week 1 strategy. To replace subjective intuition with a data-driven approach, we spent the off-season developing a cohesive workflow utilizing new, specialized analysis tools. Leveraging this objective framework for 2026, we engineered a robot optimized for fast cycles under heavy defense, specifically integrating a 360° turret to maximize our firing window and scoring uptime.

# Prototyping

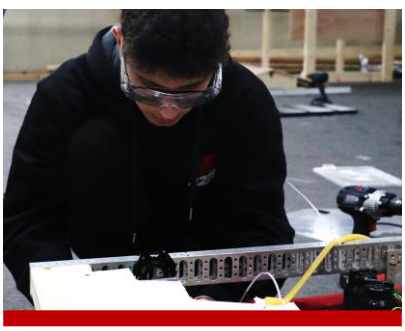
We built wood and aluminum-made prototypes, aimed at validating ideas that meet the requirements for the robot, according to the game analysis. We checked the applicability, desired pressure, types of materials, sizes, and more.

## Proof of Concept

We built several concepts for each action that the robot needs to perform. That way, we could compare different concepts and find the one that led to the optimal result.

## Optimizing Values

After proving the concept, we tested different pressures and sizes to find the best results for our priorities.



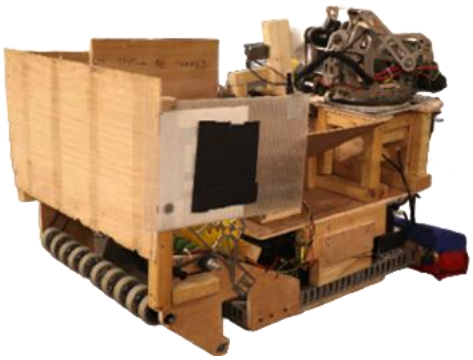
We conduct a **P**reliminary **D**esign **R**eview (**PDR**) meeting, selecting the designs for our robot.

After selecting each prototype individually, we integrated them into a "prototype robot“ and ran integration and logic tests on it.

# From Concept to Product

This year we developed our robot in three stages**: prototype bot, robot β, robot $\alpha$**, in order to make the best robot possible. We made a fully detailed work plan and agile schedule to support this way of work and R&D

## Prototype robot



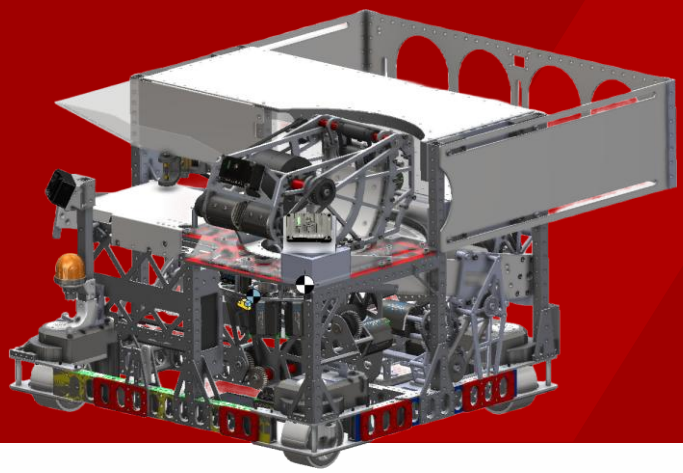
Designed to ensure integration between all systems selected from the prototypes. After selecting the prototypes, we assembled them into a complete robot, and began integration and software algorithm tests

## Robot β

Robot β is the first robot to be designed and assembled. After volume testing and integration of the prototype robot, on which we will perform tests of all mechanisms and their integration more precisely.

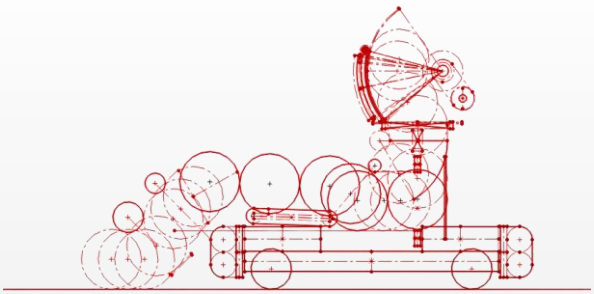
## Robot $\alpha$


After completing the design modification from the tests on Robot β, we manufacture, assemble, and wire Robot $\alpha$. This robot will be used for the competitions. All while updating Robot β accordingly in order to enable **simultaneous work on both β and $\alpha$.**



# CAD

We 3D modeled each and every part of our robot in SOLIDWORKS to match our reliability and precision expectations. Throughout the years, we have developed several creative and unique features we are very proud of.

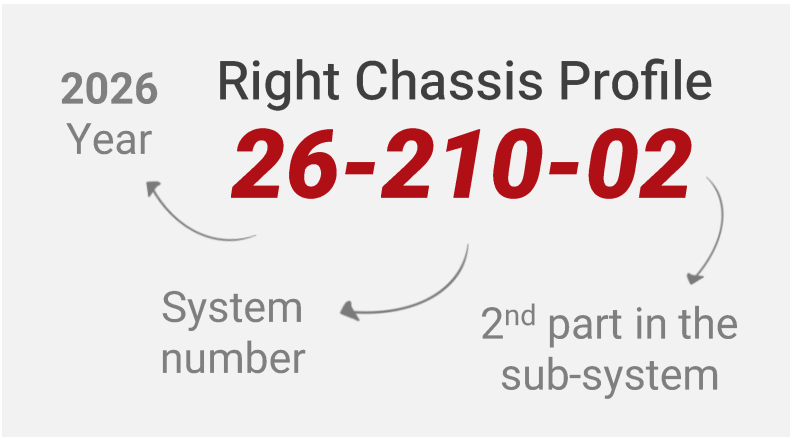


## 2D Modeling

We begin the CAD process by modeling a 2D sketch of each subsystem in order to determine the key dimensions of the system's geometry and simulate motion.

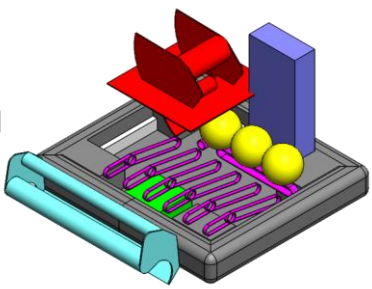
## Unique SKU System

**We use serial numbers for all our parts, for example:**




Using the SKU system simplifies the manufacturing, eliminating duplication of parts and streamlining the manufacturing process to be organized and efficient.

In order to avoid integration issues, we have created a simplified model of the robot that assists in understing the **volumes** of each system, ease of executing to robot model, and perform **geometric testing.**



# Manufacturing the Robot

Before we begin manufacturing the robots, we hold a CDR - **C**ritical **D**esign **R**eview meeting, to summarize the 3D model, and receive remarks about the robot design and its function, to send it to manufacturing without any mistake.

**19%**
In-house 3D printers

**36%**
In-house lathe machine

**27%**
In-house CNC machine

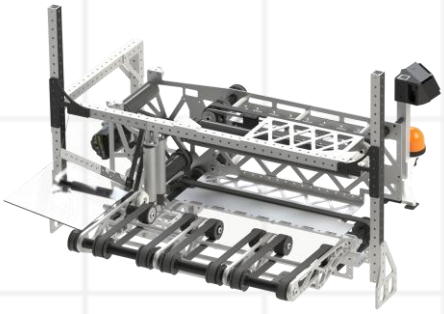
**18%**
Commercial off-the-shelf

We have created a **Bill Of Materials (BOM)** table in monday.com to manage our process in the most professional way possible. We wrote for each part SKU, description, quantity, material, status, and much more.

The order of the manufacturing process is according to mechanisms. Firstly, we manufacture all parts of the first mechanism that have been prioritized, and only then do we move on to the next mechanism allowing us to assemble mechanisms at the time of manufacturing the other, optimizing our assembling time.

Following the production of two robots, in robot-β, our prioritization is different - speed will be more important than quality, but to a certain extent, we gave up on the material weight removal in the various parts, thus saving valuable time. In contrast to robot-$\alpha$, where quality is important, as well as speed, therefore the parts are manufactured in house more carefully and sometimes outsourced.

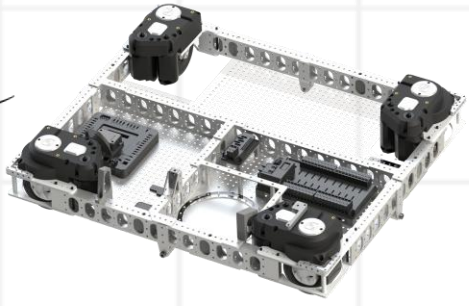
# mumble



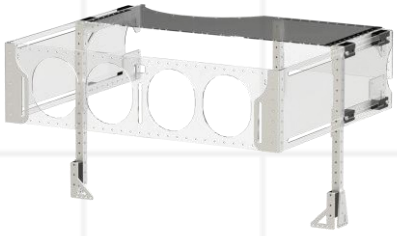
## 25-400-00
Conveyance

## 25-200-00
Fuel Ground Intake


## 25-300-00
Shooter and Turret



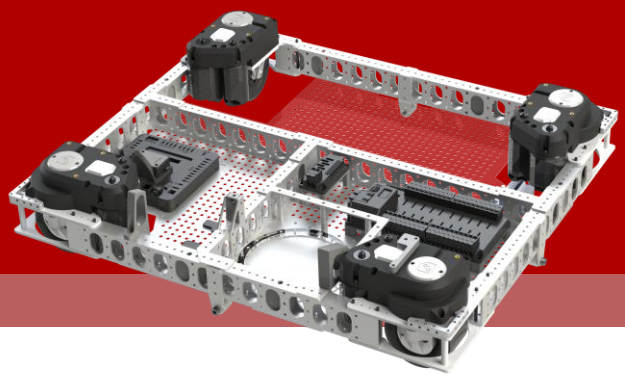
## 25-100-00
Chassis



## 25-600-00
Cartridge

# Chassis

## 26-100-00

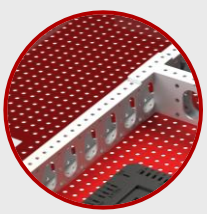


A closed chassis consisting of 4 aluminum profiles and 2 additional inner profiles for reinforcement and systems integration.

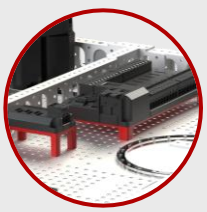
Consisting of 4 SDS MK5n Swerve modules, each powered by a Kraken X44 motor for steering and a Kraken X60 for driving. We chose to use the R1 gear ratio for driving which is 7.03:1 and for steering the gear ratio is approx. 26.09:1.

**15.4kg**

The roboRIO is assembled with a bottom plate for convenience in need of disassembling



The Belly pen is half polycarbonate for weight reduction



The electronic components are elevated for easier cable management

**4.5** ***m/s max velocity***

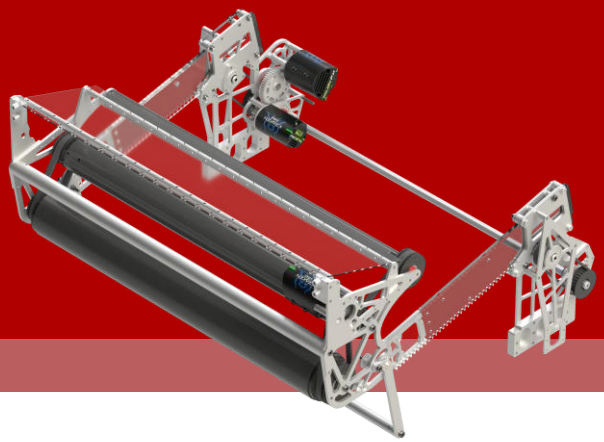
Dynamic speed, current, and acceleration limits based on the robot's state and position on the field

The software filters input from the driver to achieve a smooth and stable ride

We utilize CTRE's FRC utilities for our swerve codebase

# Intake

## 26-200-00



The system is designed to intake fuel from the floor and transfer them to the conveyance subsystem in 0.5 seconds. It was made to be robust and withstand collisions. The system is 660mm wide.

The system consists of 1 Kraken X60 and 1 Kraken X44:

- The Kraken X60 motor is designed to power the intake rollers with a gear ratio of 9.6:1
- The Kraken X44 motor is there to open and close the system linearly with a gear ratio of 4.5:1

**0.4 sec to open**

**9.3kg**

The system opens and closes linearly for simplicity and sustainability

The system consists of 3 active rollers for consistent intake

The system includes a kicker that expands with the system

**5** ***fuel wide***

We programmed the ability to extract fuel from the robot through the intake in a high performant and efficient way

Recognizes physical collisions and limits torque accordingly to prevent physical damage to the system

Dynamic current limits according to the robot's state and position on the game field

# Shooter

## 26-300-00

The system was designed to score and deliver fuel from the conveyance system into the hub. It consists of a 4in aluminum wheel for 1.5in wide thread, additionally, 4 1in sushi rollers that are designed for spin control of the fuel. In addition, the system includes a dynamic arc, to control the angle of the shooting to match the position on the field. All wheels are powered by 2 Kraken X60 motors with a gear ratio of 1.25:1. The dynamic arc is powered by a Kraken X44 motor with a gear ratio of 58:1.

**0.1 sec to warm up**

**8.2kg**

2 Sushi rollers in order to control the fuel's spin

The Hood controls the shooting angle of the fuel with 2 serrated plates on a gear

2 small bearings on every side of the hood's side plates that improve the stability of the hood

**12** ***fuel/sec***

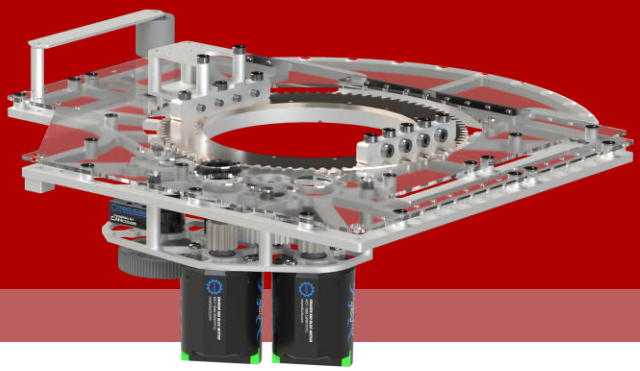
Autonomous shoot sequence based on the robot's position and the alliance shifts' state

Upon entering the trench, the hood retracks to protect itself from impact

Calculates the best arc angle to shoot from every position on the field using vision systems

# Turret

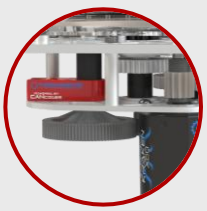
## 26-303-00



The system is designed to provide versatility for shooting fuel without the restraint of the chassis' angle.
It is powered by 2 Kraken X60 motors with a gear ratio of 4.347:1.
The turning of the turret is possible using a 7in bearing inside a 10DP 88T gear that connects to the gearbox which assists us to result in the gear ratio mentioned above.

**3.6kg**

The shooting system is assembled on top of a 10DP gear we manufactured internally



We use an encoder in the Turret's gearbox that grants us a 1:1 conversion rate for maximum accuracy

The Turret system is moved by 2 Kraken X60 motors for quick and fast movement

**495** ***Turning degrees***

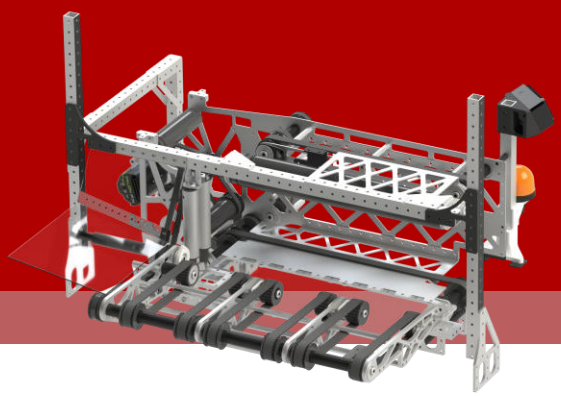
A Limelight 4 is placed on the shooter to ensure accurate localization and minimal dead zones

Dynamic turret tolerance according to the distance of the robot from the target

Autonomously tracks the target and rotates accordingly using vision systems

# Conveyor

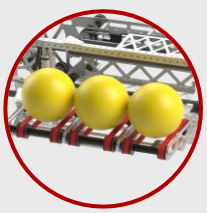
## 26-400-00



The conveyance system is designed to transfer fuel from the intake to the shooter. It is divided into 2 parts, conveyer and trigger. The conveyer transfers the fuel to the trigger using pulleys and timing belts as the converyors, in addition to a horizontal roller, the timing belts are powered by a Kraken X60 with a gear ratio of 3.125:1, and the roller is powered by the same motor, with a gear ratio of 2.14:1. The trigger transfers the fuel from the conveyer to the shooter. It is powered by Kraken X60 motor, with a gear ratio of 5:1.

**0.4 sec from intake to trigger**

**9kg**



There are 3 Conveyor belts that ensure 3 available fuel for shooting at any time

The system includes 2 vertically placed rollers (1 active) to fix geometric errors

The trigger is constructed by 3 rollers moving in synchronization using belts to save space for the conveyor system

**12** ***fuel / sec***

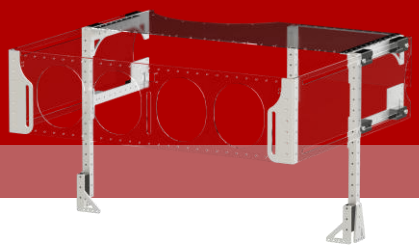
We programmed a button for releasing fuel that gets stuck in the conveyor system, while not harming robot's performance in game.

Current limits to prevent burn out of the battery during shooting, and to preserve its lifetime

We use a PID control system in order to prevent overuse of power and to prevent getting fuel stuck in the robot's systems

# Cartridge

## 26-600-00



The cartridge subsystem is an all-passive subsystem which moves with the intake. It's designed to hold the fuel in their way to the shooter, while waiting for our opportunity to shoot.

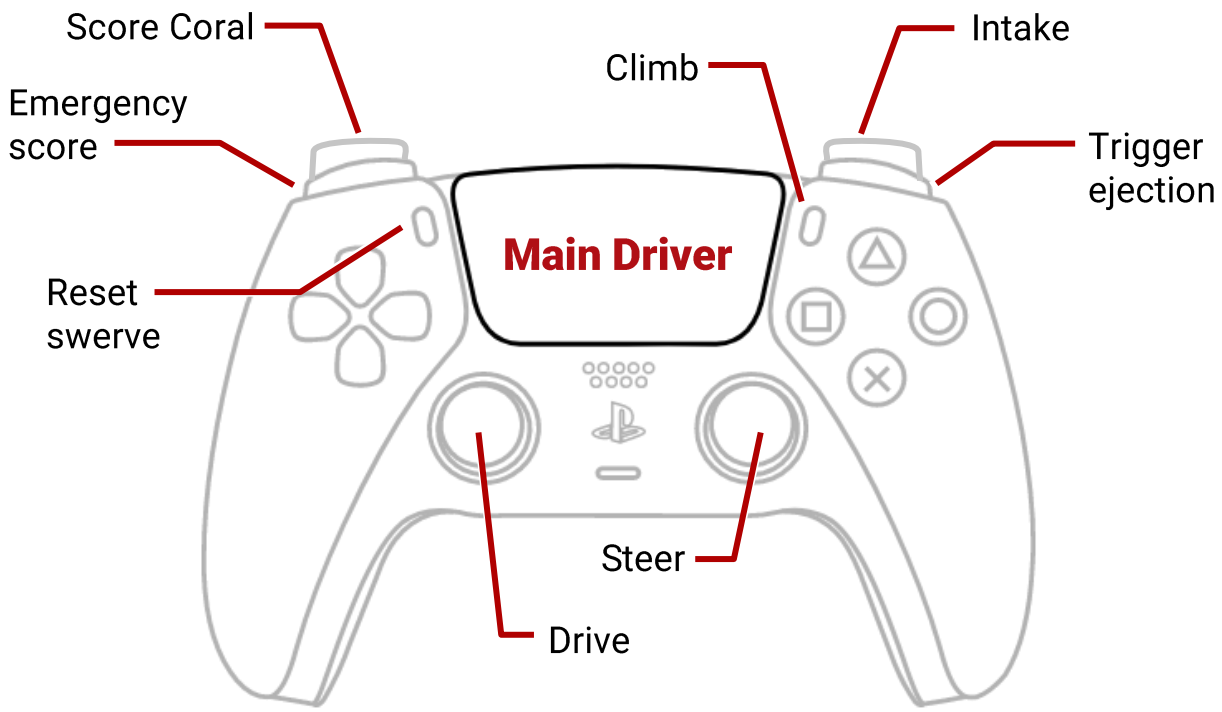
**~450 fuel/game**

**2.7kg**

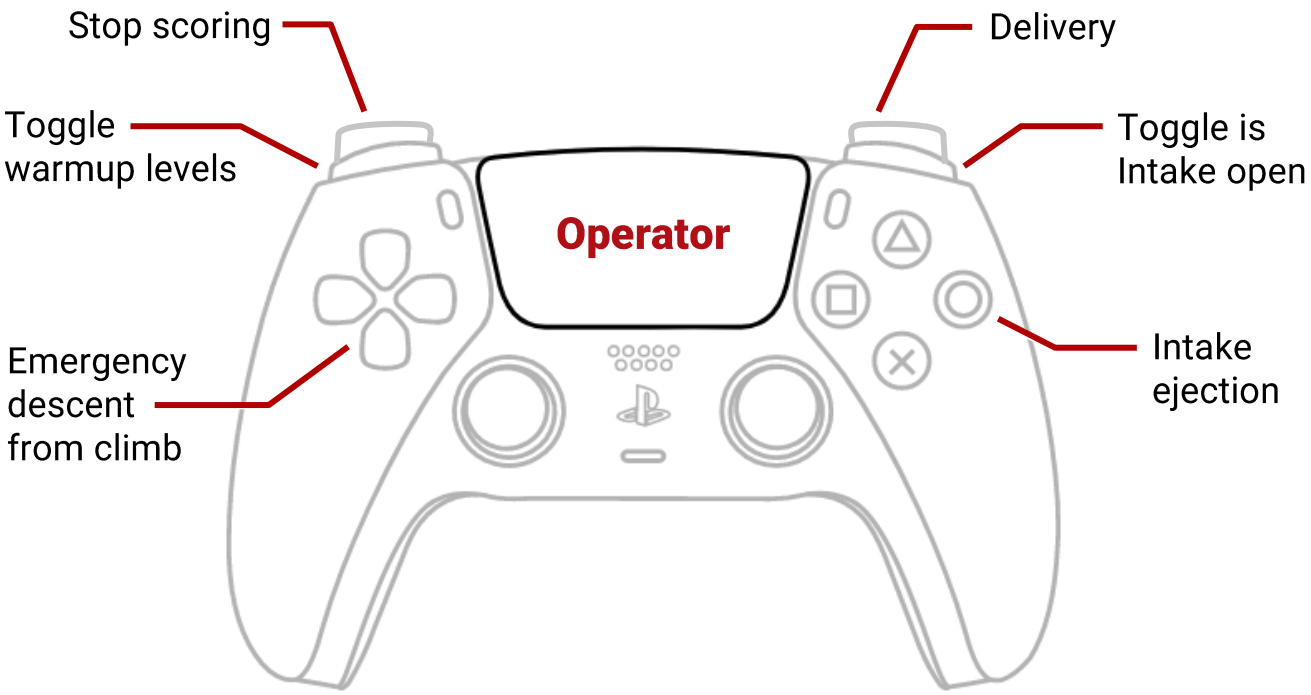
A vertical slot on each side, to enable the linear-diagonal movement of the ground intake

With the ability of the cartridge to expand, the robot can hold up to 50 fuel at a time

# Controlling the Robot



# Controlling the Robot



# Our Innovative

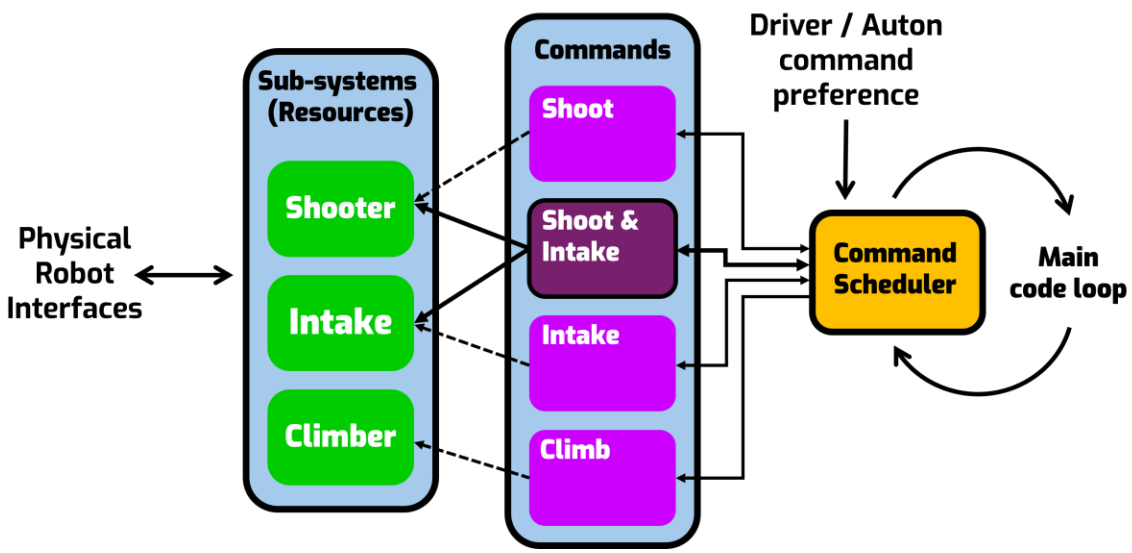
One of the most significant programming challenges is maintaining a holistic code structure. Rather than micromanaging individual motors, the goal is to conceptualize a mechanism - such as the turret - as a unified, single entity engineered to achieve a specific outcome, like Reaching a target angle precisely.

To achieve this level of software abstraction, **two primary paradigms are typically employed**:

- **Command-Based Programming**: The default FRC framework. It is provided by *FIRST* as a fully prepared platform accompanied by comprehensive documentation.
- **State-Based Programming**: A custom architecture that is usually built from the ground up. This approach is currently utilized by a select few elite teams who have largely kept their architectures proprietary.

**Up until the current season, our team operated exclusively within the Command-Based paradigm.**

In this architecture, every action is defined as a distinct "Command", and each mechanism is a “subsystem” that can be used by a single command at a time. To construct intricate operations, developers must sequentially or logically chain these commands together.

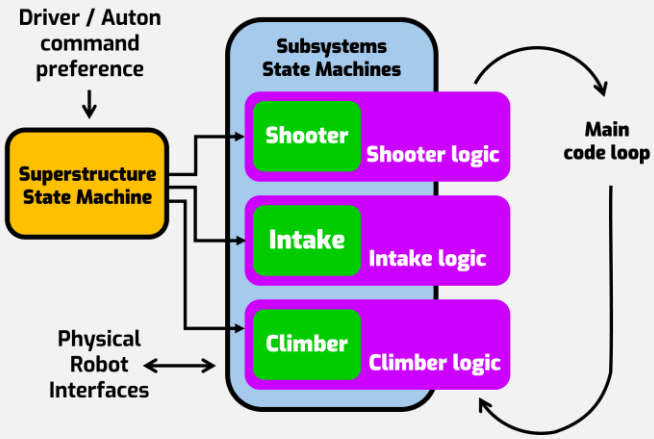


# Framework

We paid a heavy price for this generality in the form of **elevated execution times and severely restricted optimization capabilities.** Specifically, it hindered our ability to perform command pruning or dynamically inject multi-system automations sequences mid-execution. Although we managed to retrofit these capabilities into the Command-Based structure, our workarounds inadvertently degraded the robot's main loop cycle time.
We concluded that we could no longer compromise on performance and advanced capabilities for the sake of an out-of-the-box generic solution. **During the Off-Season, we designated the development of a proprietary State-Based Framework as a mission-critical project for the upcoming year.**

Within this architecture, the robot is conceptualized as a global Finite State Machine (FSM), with each individual subsystem operating its own discrete FSM to execute sophisticated operations. Every state is meticulously defined and **tailored strictly to the specific demands of the game challenge** and the required hardware.

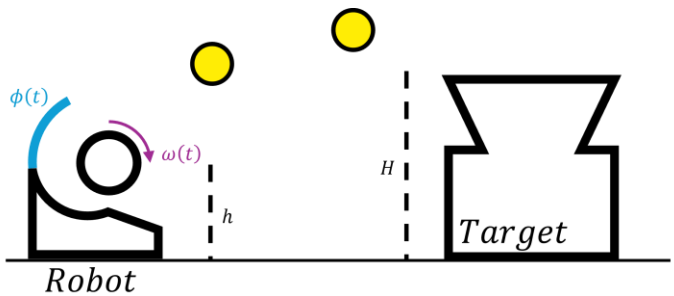
We are proud to say that **this custom-built framework has successfully elevated our codebase's level of abstraction. It has profoundly streamlined our ability to integrate arbitrary commands into any action on demand, entirely eliminating the performance overhead previously imposed by the generic command structure**

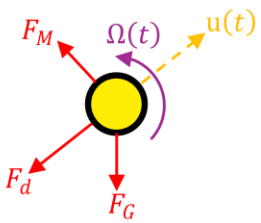


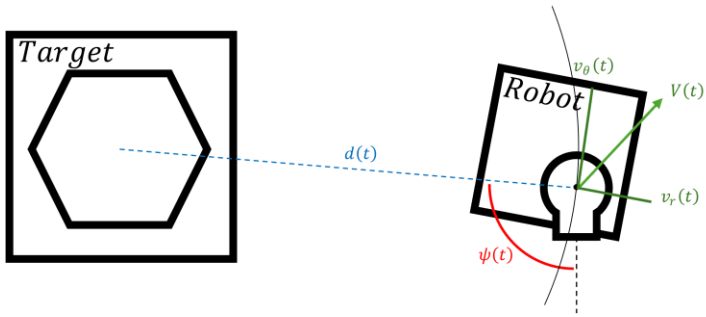
# Shoot On The Move

We initiated the development of our dynamic targeting system by constructing a comprehensive kinematic simulation. This model accounts for gravitational acceleration, the robot's frame-relative velocities, aerodynamic drag, and the Magnus effect.

By the conclusion of Week 1, we achieved a functional environment whereby inputting the robot's pose, hood exit angle, and flywheel surface velocity the model numerically integrates the equations of motion. This produces a continuous projectile trajectory, establishing the mathematical baseline required to solve for high-probability scoring solutions.




$$\|\vec{u}(0)\| = \frac{1}{2}\|\vec{\omega}\| \cdot r_{wheel} \cdot \gamma$$

$$\|\vec{\Omega}(0)\| = \frac{1}{2}\|\vec{\omega}\| \cdot \frac{r_{wheel}}{r_{ball}} \cdot \gamma$$

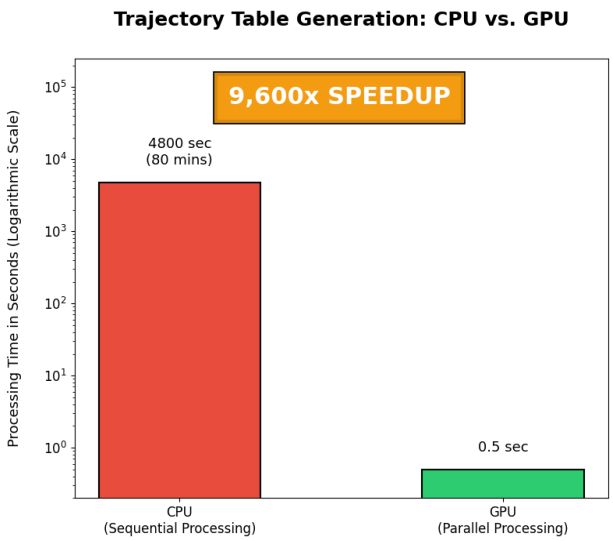
$$\vec{F}_d = -\frac{1}{2} C_d \cdot \rho A \cdot \|\vec{u}(t)\|^2 \cdot \hat{u}(t)$$

$$\vec{F}_M = \frac{1}{2} C_l \cdot \rho A \cdot \|\vec{u}(t)\|^2 \cdot \hat{\Omega} \times \hat{u}$$

Moving beyond trajectory simulation, we dynamically calculated the required hood angle and shooter velocity based on the robot's real-time kinematics. We used the Newton-Raphson method and an iteration through wheel velocities, to determine the optimal firing parameters.

# Shoot On The Move

After achieving convergence for a single state, we ran the algorithm across a continuous domain (distances of 1.5 to 5.5 meters, radial velocities of ±5 m/s) to generate a lookup table. This enabled 2D interpolation to find **optimal shooting parameters for any feasible robot state**. By offloading these calculations to the GPU, we reduced the computation time from several minutes to **under a second for over 4,000 values**, drastically streamlining the debugging process.



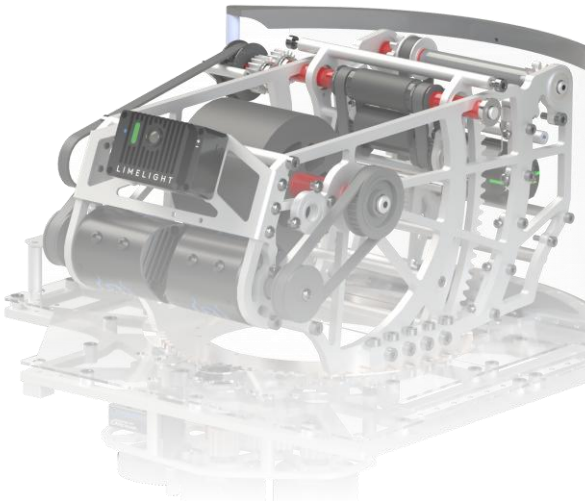

Every robot's shooting mechanism will exhibit distinct physical behaviors based on the volume of balls fed in rapid succession (varying "inertia retention rate"). To account for this, we introduced a tuning parameter, (gamma), representing inertia retention as a value between 0 and 1. is utilized within the simulation to appropriately attenuate the mechanical energy transferred to the ball**. The value is the sole parameter requiring manual calibration.** Furthermore, it scales collectively across all data points: **calibrate for one point, and it will be accurate for all.**

$$\|\vec{u}(0)\| = \frac{1}{2}\|\vec{\omega}\| \cdot r_{wheel} \cdot \gamma$$

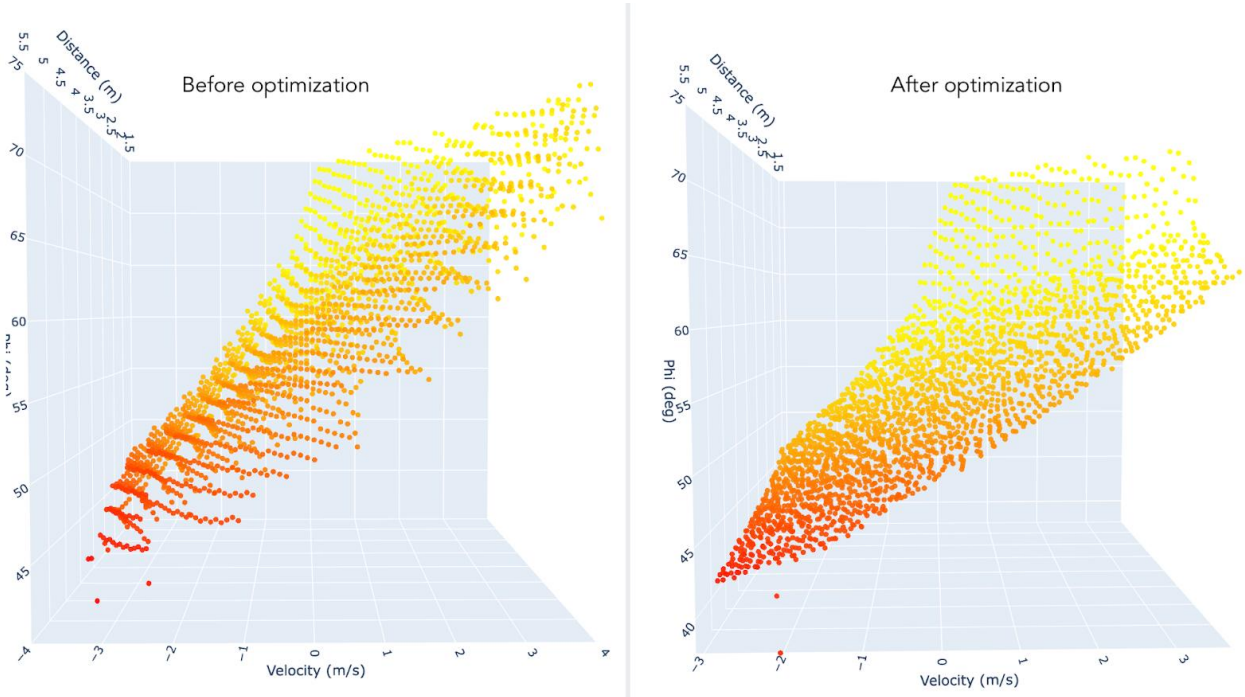
$$\|\vec{\Omega}(0)\| = \frac{1}{2}\|\vec{\omega}\| \cdot \frac{r_{wheel}}{r_{ball}} \cdot \gamma$$

Calculating the initial shooting velocity, while compensating for energy loss using the Gamma factor.

# Shoot On The Move

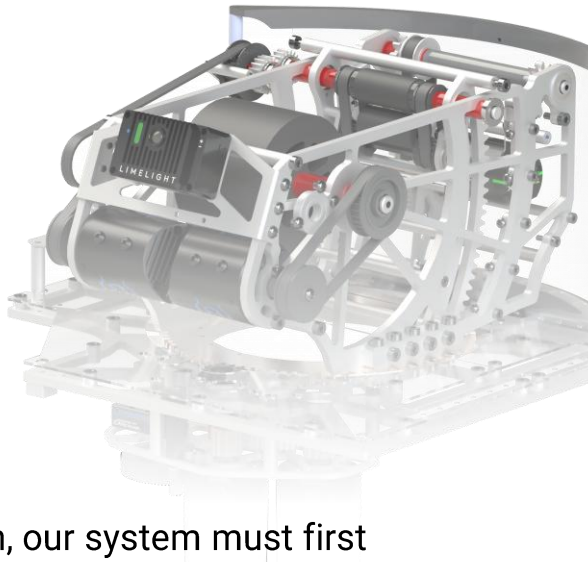



We focused next on the constraints of our lookup table. We wanted to reach a seamless transition between hood angles - to eliminate abrupt jumps. To visualize and fix this, we generated a 4D graph: inputs (distance to target and velocity magnitude) formed base axes; outputs were Z-axis height (shooter wheel velocity) and color gradient (hood angle). Initial observations showed highly noisy topological maps. To fix this, we **enforced a smoothness constraint**: the difference between adjacent points must not exceed a calibrated (epsilon) error threshold, ensuring a smooth, continuous output graph.




Phi is the angle of the Hood, in degrees
The color indicates the shooting speed: The lighter the color the faster the shooting

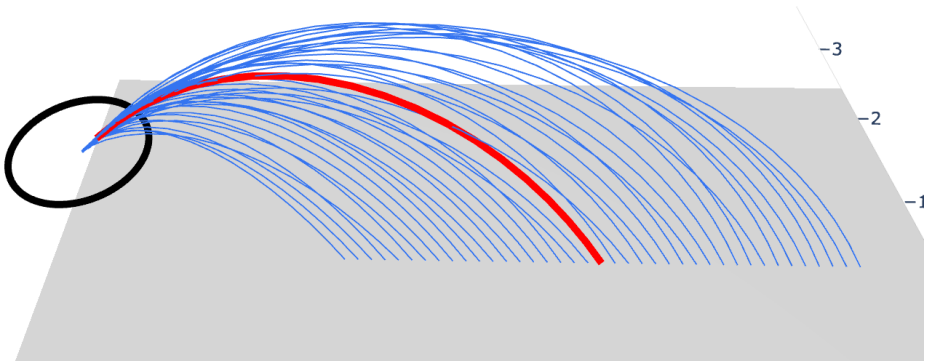
# Shoot On The Move

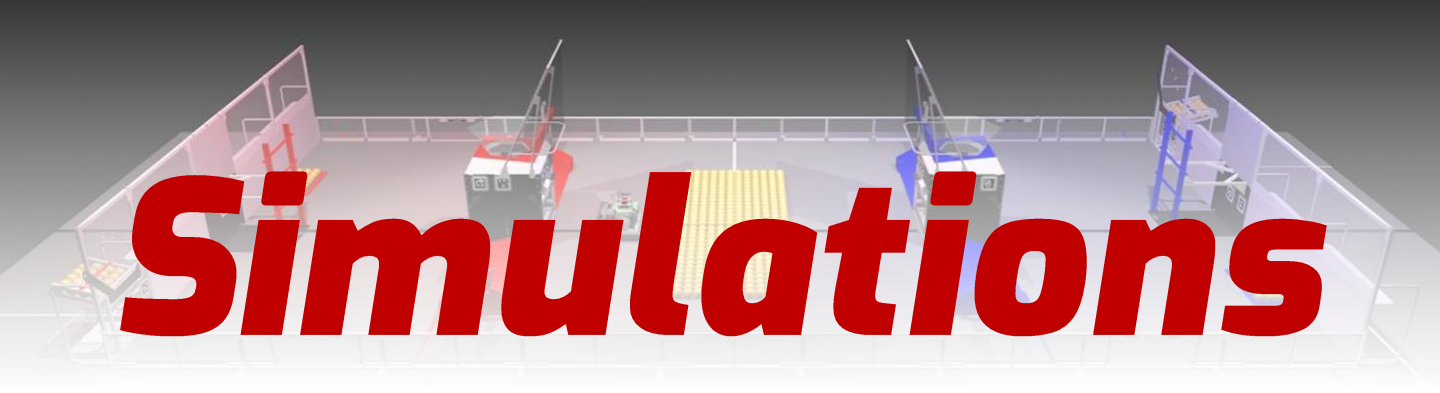



To calculate precise firing parameters in motion, our system must first isolate the turret's polar velocities (radial, azimuthal, and angular) relative to the HUB. We derive these by taking the vector sum of the robot's linear and rotational velocity, resolving the resultant vector into two distinct axes.

Once decoupled, these velocities are applied to targeting logic. Radial velocity (with distance) is fed into a simulation-generated lookup table, which prescribes the **optimal base firing vector**. Since this table assumes tangential velocity is zero, we correct for azimuthal motion by a straightforward vector subtraction: deducting the tangential component from the theoretical firing velocity. This operation dynamically recalibrates the turret angle (as well as hood angle and flywheel velocity) so the resultant vector, when summed with robot movement, precisely negates the tangential velocity and lands the projectile.

To execute these dynamic recalibrations, our advanced turret control architecture dictates both position and velocity. Target position is derived from the calculated angle, and the target velocity is determined by the turret's angular velocity around the HUB. This **dual - state control scheme ensures smooth, highly accurate, and continuous turret tracking.**

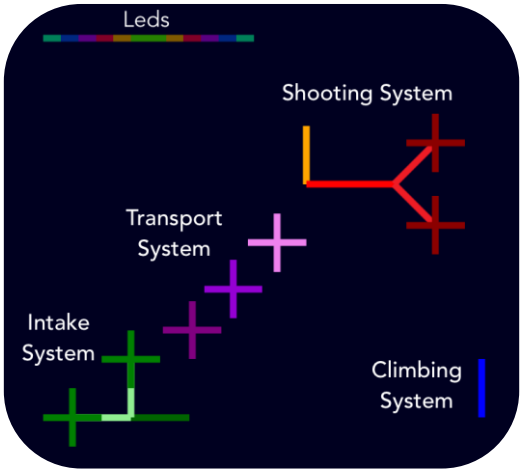




# Simulations

Throughout our season, simulations were an integral part of our software workflow, evolving in complexity as our robot's code matured. To drive these simulations and render our environments, we relied heavily on AdvantageScope, utilizing it from the earliest stages of development through to our final field tests.

**Week 1: Foundational Logic & 2D Simulation:** From the very first week of the build season, we began drafting the robot's architecture. We required a robust method to verify our underlying logic before any physical parts were assembled. Consequently, we implemented a simulation for each subsystem that modeled motor behavior and visualized the mechanism within a 2D environment.




The simulations allowed us to verify multi-subsystem operations - such as our physics solvers and complex shooting mechanisms - and resolve integration conflicts. By testing the Superstructure virtually, **we drastically mitigated the risks associated with deploying untested code onto the physical hardware.**

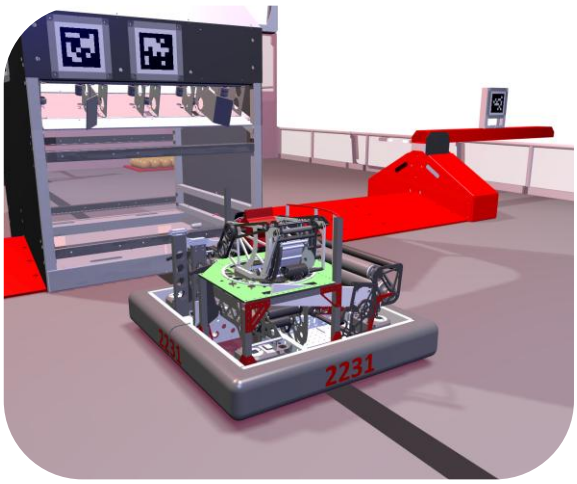
These initial 2D simulations empowered us to validate our logic long before a physical robot existed. Through this process, we identified and rectified foundational code flaws prior to receiving the physical mechanisms from the mechanical subteam.

**Weeks 2 & 3: Superstructure & Systems Integration:** Moving into the second and third weeks, our focus shifted to architecting the robot's Superstructure - the overarching state machine that efficiently orchestrates all subsystems. We needed a way to rigorously test and optimize it.

# Simulations

**Advanced Development: 3D Odometry & Field Simulation:** As our autonomous routines and advanced features took shape, we developed a comprehensive 3D simulation that actively incorporated the robot's field pose into our testing protocols.

This enabled us to validate critical, position-dependent actions - such as automated shooting sequences triggered from within the Alliance Zone - ensuring the robot reacts intelligently regardless of its coordinates on the field. As a subset of this simulation, we also engineered a mock FMS (Field Management System) to replicate field behavior and transition states during a simulated match.



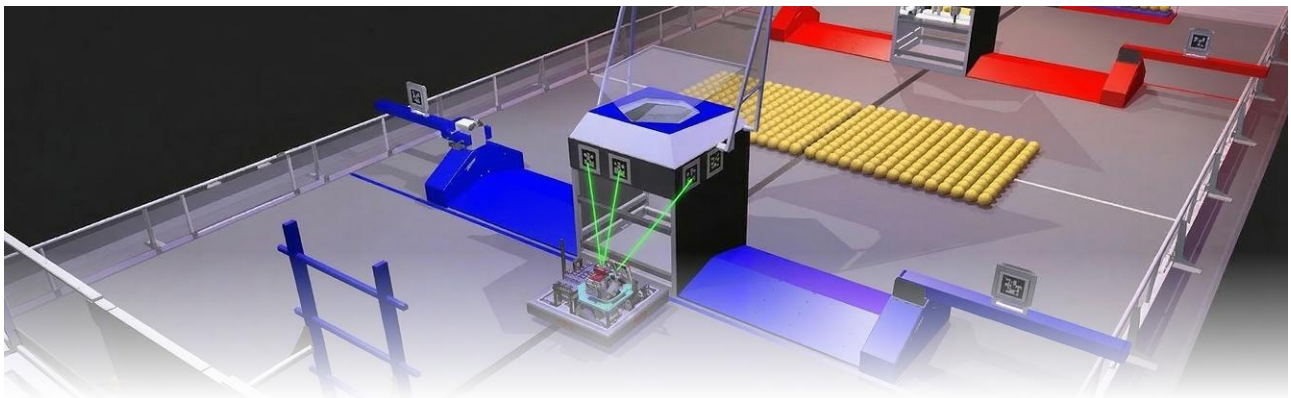

Ongoing Workflow & Retroactive Debugging Even after the physical robot was completed, our simulation and telemetry workflow remained crucial.

Using AdvantageScope, we were able to scrub through historical logs, giving us deep visibility into every state, action, and active subsystem on the robot for retroactive debugging. Ultimately, we relied on these simulations entirely throughout the season to guarantee that new code would not cause mechanical failures before it was ever deployed to the actual robot.

# Localization

While AprilTags provide the absolute 3D pose estimation necessary for aligning with field elements, raw vision data is inherently susceptible to noise introduced by distance, motion blur, and perspective ambiguity. To mitigate this, we implemented a **custom sensor fusion algorithm using a Kalman Filter** to integrate the vision data with our drivetrain odometry.  Our system employs **dynamic covariance scaling**; the filter dynamically shifts its 'trust' weight, prioritizing proximate, low-ambiguity AprilTag detections while relying more heavily on odometry during high-speed maneuvers or when tags are distant. This ensures a continuously accurate, highly resilient global field position - with around 2 cm accuracy variance.

To achieve unwavering field localization, we engineered a dynamic, turret-mounted Limelight system that continuously tracks the most reliable and proximate AprilTags throughout the robot's traversal, effectively **eliminating vision dead zones** and ensuring pinpoint pose estimation. While standard drive odometry is notoriously vulnerable to compounding drift caused by wheel slip, our automated turret system directly mitigates physical inaccuracies by maintaining a semi-unbroken visual lock on AprilTags (apart from times while shooting on the move). By articulating the camera independently of the chassis and continuously feeding high-fidelity visual telemetry to our pose estimator, the system seamlessly overrides noisy odometry data during field disruptions, maintaining a robust and uncompromisingly accurate robot pose at all times.



# Automations

## Maximizing Performance Through Software


One of our primary design philosophies is to minimize the number of things the driver needs to pay attention to at the same time. We do so by shifting the focus from robot operation to strategic gameplay. We achieve this by integrating **advanced automations across all match stages** - Autonomous, Teleop, and Endgame. This allows the driver to concentrate fully on field awareness and efficient intake, the system automates nearly all complex functions.

**The driver's manual control is streamlined specifically to FUEL collection, emergency overrides, and shot cancellation.** By handling the technical execution automatically, the robot optimizes efficiency, ensures self-protection, and allows the driver to focus on high-speed performance and scoring.

### Autonomous Firing & Intelligent Field Navigation

The robot's software architecture centers on a **fully autonomous firing sequence**, removing the decision-making burden. The robot independently initiates the shot only when within the Alliance Zone and the HUB is active, ensuring no FUEL is wasted. During this sequence, the system calculates clear line-of-sight, suppressing shots if obstacles like the climbing structure or the trench are in the trajectory. When positioned outside the Alliance Zone, the turret automatically indexes toward predefined "Delivery" zones based on the current field side, allowing for the accurate staging of FUEL for future cycles.

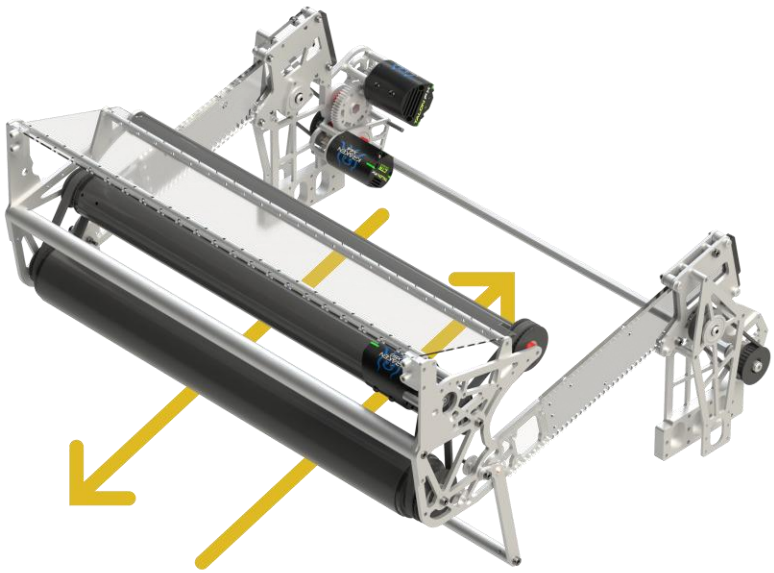
This offensive autonomy is complemented by proactive self-inflicted damage prevention. When the robot enters the trench, the hood automatically retracts for a smooth, collision-free passage. This allows the driver to focus on high-speed maneuvers and positioning without worrying about mechanical interference.

# Automations

## Maximizing Performance Through Software

**Active Intake Protection:** As soon as we begin cycling FUEL, our first line of defense is the **Active Torque Limiting** logic programmed into our intake. Deployed via a linear rack and pinion, the intake motor utilizes a torque position based PID control loop with a strictly defined **output ceiling**. Ordinarily, this loop applies sufficient torque to hold the intake rigidly at its full extension outside the frame perimeter.

However, because the intake is exposed to high-velocity collisions with other robots and the field perimeter, we introduced a compliant control element. Rather than fighting an impact with maximum stall torque - which could shear the rack and pinion teeth or bend the structure - the software imposes a **torque threshold**. When an external force exceeds this limit, the motor allows for a forced mechanical retraction. This approach allows the intake to absorb shocks and back-drive safely during heavy defense, immediately returning to the setpoint once the external pressure is removed.

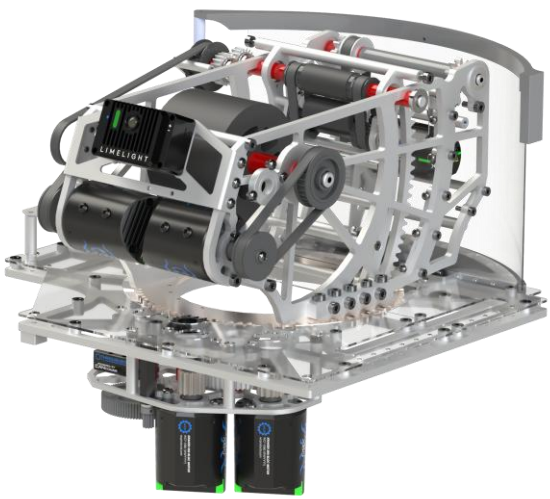


# Automations

## Maximizing Performance Through Software

**Dynamic Turret Tolerance & Target Tracking** As we navigate the field, our turret is in a constant state of target acquisition. To drastically reduce lock - on time without sacrificing accuracy, we programmed a dynamic error tolerance for the turret''s angle. The allowable angular error of the turret is calculated in real-time based on the robot's distance from the HUB and the width of the HUB.

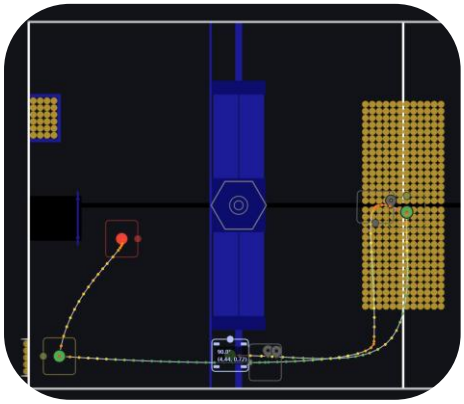
Geometrically, the physical width of the HUB remains constant, but its apparent angular width changes based on our distance. When the robot is close to the HUB, the acceptable angular tolerance is significantly larger. As the distance increases, the software tightens the tolerance. By mathematically exploiting the width of the target, our control loops can validate a "ready to fire" state much faster at close range, optimizing our cycle times without risking missed FUEL.



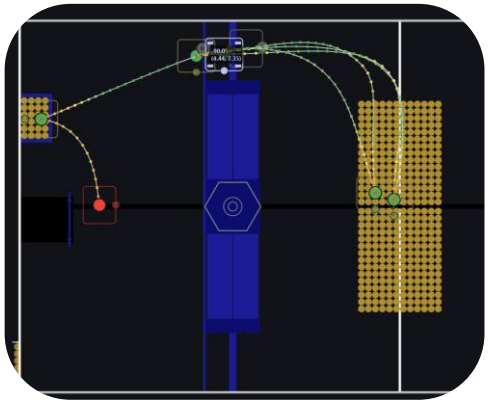

# Autonomous Period

**Precision Routing with PathPlanner:** PathPlanner application grants us absolute control over the robot's kinematics and allows for highly precise, position-based event scheduling. By plotting our exact routes, we can seamlessly trigger subsystem actuations at the optimal moments along the trajectory, maximizing our scoring efficiency in the opening seconds of the match.

**The "AutoPilot" Architecture & Code Encapsulation:** Our new code structure enabled us to achieve a flawless handoff between PathPlanner's trajectory tracking and our proprietary "AutoPilot" navigation mechanism. By aggressively encapsulating our code, AutoPilot serves as a universal, reusable autonomous drive module. The exact same autonomous navigation behaviors we use during Teleop can be injected directly into our Autonomous routines. This code reuse guarantees consistency and drastically reduces debugging time across all game periods.



**Left Double Cycle Depot:** Executes a strategic delivery to the field center to prime the transition stage. This is followed by an autonomous intake and "shoot-on-the-move" sequence from the feeder. The autonomous concludes with a fully automated climb



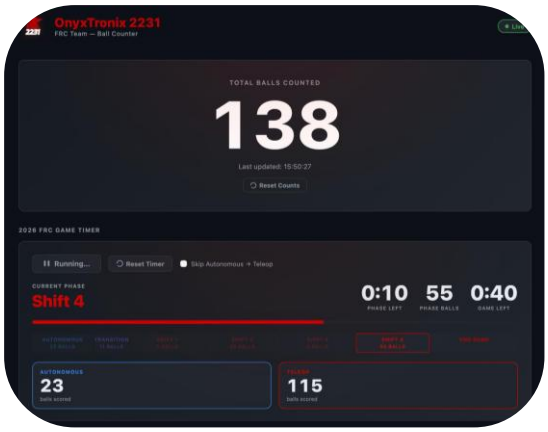
**Left Double Cycle Depot:** Executes two rapid cycles to the field center to establish early dominance over the middle. Then scores efficiently in our Alliance zone, before transitioning to the Depot for a move-and-shoot sequence. This routine concludes with an automated climb.

# Strategic Software

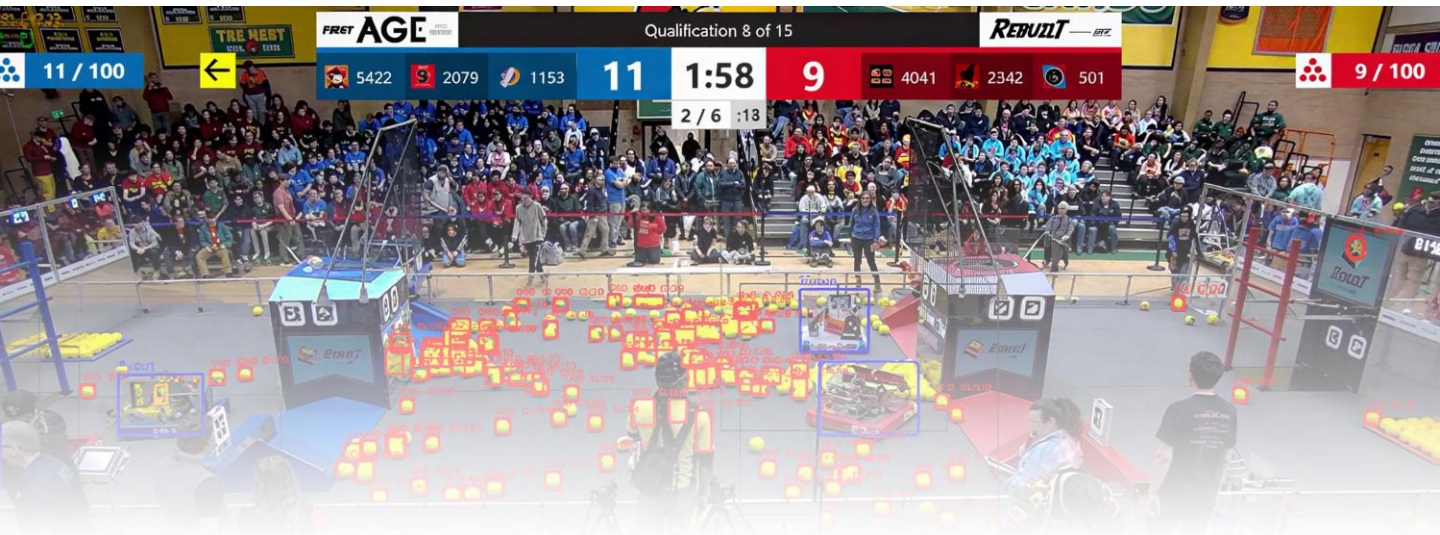
## Beyond the Robot

Our software team’s impact extends beyond the robot's RoboRIO, developing custom technological tools to elevate the entire team's strategic edge.

**Real-Time HUB Analytics** We engineered a custom vision system powered by a Raspberry Pi and a camera mounted directly to our practice field's HUB. This system actively identifies and counts FUEL as it enters the target, providing our drive team with real-time scoring metrics and immediate feedback on their cycle efficiency during practice.




**AI-Driven Scouting Platform** Because the high-speed nature of the game makes manual scouting highly error-prone, we developed a Scouting AI based on real-time video processing. The software autonomously tracks robots and FUEL across the field, automatically calculating the specific scoring output of every individual robot. This provides our strategy and drive teams with flawlessly accurate, unbiased data that is impossible to capture purely by human observation.



# 20 YEARS of OnyxTronix

CTR ELECTRONICS

aws

monday.com

Qualcomm

SHERMAN
TAILORING INTEGRATED SOLUTIONS

Icon Systems

Elsar since 1985

LEDICO

HAAS
Gene Haas Foundation

extal
We know aluminium

KENESTO

ptc

FIRST ISRAEL ישראל

תיכון שוהם
Shoham High School

חמש
יש לנו פנאי בטבילן

מועצה מקומית שוהם