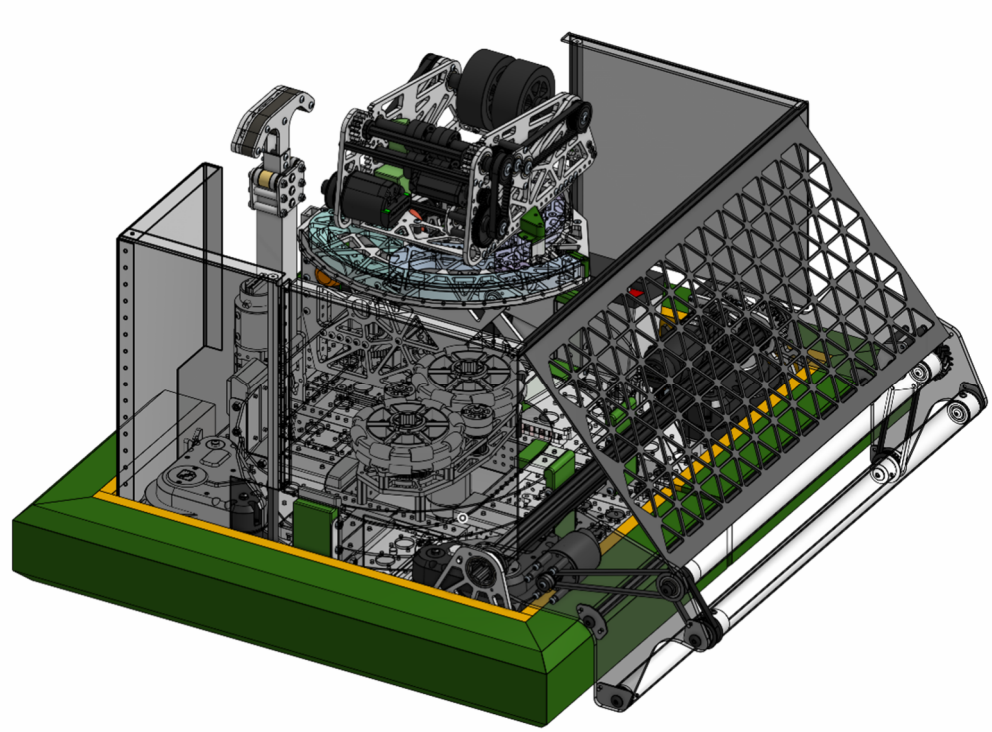
REBUILT

PROUD MEMBER

20 YEARS OF WOLVERINES



URI DISTRICT
EVENT EDITION

ORION

WOLVERINES

1757

2026 TECHNICAL BINDER

YEARS OF

WOLVERINES

WINNER	***WINNER***	***WINNER***	***FIRST IMPACT AWARD***	***FIRST IMPACT AWARD***
2023 WPI DISTRICT	2023 NEDC - WILSON DISTRICT CHAMPIONSHIP DIVISION	2023 NEW ENGLAND DISTRICT CHAMPIONSHIP	2024 WPI DISTRICT	2025 URI DISTRICT

FORWARD

Hello, and welcome to FRC Team 1757's 2025–2026 Season — our twentieth. We want to make sure you read that again, because we almost can't believe it ourselves.

After the hard lessons of 2024, last year's season with Perseus was a deliberate return to who we are: a team that wins by doing a small number of things with exceptional consistency. It worked. We competed with confidence, our software pipeline finally matched the ambitions we'd been writing about for years, and we walked away from the season with our heads held high. That foundation — rediscovered rather than rebuilt — is what we brought into January.

And then FIRST revealed the 2026 game.

REBUILT™ presented by Haas is themed around archaeology — uncovering the past, understanding how the innovations that came before us shape what we build today. Every artifact holds a story. Every tool connects us to the people and ideas that came before.

We could not have written a more fitting theme for our twentieth season if we tried.

20 Years of Wolverines

It started before the robots. In 2005, two eighth graders in a small suburban town had a crazy idea: start a robotics team. Westwood is a town of doctors, lawyers, and accountants — not exactly a hotbed of industrial or technical tradition. There was no shop class legacy to build on, no obvious pathway, and no guarantee that anyone would say yes. But they went looking for the one teacher who might.
They found him. And he was just crazy enough to agree.
In the winter of 2006, that team built their first FIRST Robotics Competition robot. They named him Woody. There was no CNC, no laser cutter, no polycarbonate paneling, and certainly no carbon fiber. There was plywood. There were carriage bolts. There was duct tape — a lot of duct tape — holding together both the robot and everyone's optimism.
Woody probably wouldn't survive a single match by today's standards. But he survived something more important: he launched a team.

> *"Every artifact we uncover holds a story. Each tool, each innovation, each work of art connects us to the people and ideas that came before us."* — FIRST REBUILT, 2026 Season Theme*

Twenty seasons later, we compete with robots built from laser-cut aluminum, precision-machined polycarbonate, and structural carbon fiber. We write thousands of lines of control code. We run extended Kalman filters on our vision pipeline. Our swerve modules hold tolerances that would have seemed like science fiction to the students who drove that first plywood machine across a 2006 competition floor.

The journey from there to here is not a straight line. It is a story of seasons where we flew too close to the sun, seasons where we rediscovered what made us special, and seasons where a small team from a small school did something that larger, better-funded programs couldn't quite manage — not because we outspent them, but because we outthought them. That is the story REBUILT was made for, even if FIRST didn't know they were writing it for us.

In a town that had little interest in STEAM education, we have spent twenty years banging on that drum as loud as possible to anyone who will listen. Generation after generation of Westwood students have left this program with skills, confidence, and a way of thinking about problems that no classroom curriculum could have given them. Those students have gone on to become teachers, engineers, and scientists — working for some of the biggest and best technology companies in the world, including Google, Amazon, GE Aerospace, Universal Robotics, Boston Dynamics, and SpaceX. That is not an accident — it is the whole point. Two eighth graders with a crazy idea turned out to be right about something important.
To every student, mentor, and parent who has been part of Team 1757 across these twenty years: this season is for you. The robot that takes the field in 2026 carries every lesson learned, every late night, and every duct-taped fix that came before it — all the way back to Woody.

Introducing Orion

The 2026 FIRST Robotics Competition game REBUILT challenges alliances to score Fuel into the Hub, cross obstacles, and climb the Tower before time runs out. In a game where every shot counts and where precision separates the good teams from the great ones, our design process converged quickly on a singular goal: build the most consistent, most accurate Fuel scorer we possibly could. Then we needed a name worthy of it.

Meet Orion.

Orion, the great hunter of Greek mythology, was placed among the stars for eternity — his skill was so extraordinary that Zeus himself immortalized him in the night sky. He is a figure defined not by brute force, but by the precision of the hunt. In REBUILT, where the Hub doesn't forgive imprecision and where accuracy compounds into ranking points, that precision is exactly what we went looking for.

The name carries a lineage. In 2022, we built Skadi — named after the Norse goddess of the hunt and archery — a flywheel shooter that marked the beginning of the team's current era of competitive and technical growth. Orion is Skadi's spiritual successor: same philosophy, four years of refinement, and a game that finally puts precision shooting back at the center of the field. Two hunters, two mythologies, one unbroken thread.

Building Orion: By the Numbers

On a cold morning in January, we gathered in our classroom, dusted off the CNC, warmed up the 3D printers, and eagerly awaited the reveal of REBUILT™. What followed was ten weeks of relentless iteration.
The numbers this year tell a different story than just volume. We logged **293 git commits {} lines of code**, **6 CAD models**, **7 shared Google Drives**, **13,986 Discord messages**, **2 major snowstorms**, **30 inches of snow**, **4 snow days**, and many, many cups of coffee later, the robot that rolled out of our shop is the product of that discipline, not just that effort.
We are proud to introduce Orion for the 2026 FRC Season.

Our Fourth Year in the Open Alliance

This is our fourth year participating in the Open Alliance, and we remain as committed to it as we were the first year we joined. The community of teams who share their work openly has shaped our robot in measurable ways — in 2025 we gave design feedback to Team 111 that found its way back into our own robot by the time we reached competition. That feedback loop is what the Open Alliance was built for, and it is real.
For a team in its twentieth season, there is also something fitting about sharing what we know. The students who built that first plywood robot in 2006 did it without the resources that teams today can access: open-source codebases, shared CAD, documented design philosophies from hundreds of teams worldwide. If this binder helps even one team avoid a mistake we've already made — and we have made quite a few — then it has done its job.
If you want to follow our full build season documentation beyond what's contained in this binder, visit our Chief Delphi Build Thread at https://www.chiefdelphi.com/t/frc-1757-wolverines-2026-build-thread-open-alliance/507563. We are always ready to share the knowledge we've gained, or a few hard-earned lessons from twenty years of learning the hard way.

We hope you enjoy this look at the design process and technical details behind Orion. If you have questions, find us in the stands, in the pits, or on the field. We are never hard to locate — we're the ones who have clearly been doing this for a while.

TABLE OF CONTENTS

GAME ANALYSIS

OUR PROCESS

Every season starts the same way: the team gathers for kickoff, watches the game reveal, and spends the weekend in back-to-back sessions breaking the game down before we ever touch a piece of metal. The goal is to walk out on Sunday night with a clear understanding of the game and a direction for the robot.

Our process is based on the framework developed by Team 125, the Neutrons, which involves a systematic breakdown of the game before any design decisions are made. We begin by identifying every individual skill a robot might need — from something as simple as driving forward to something as complex as automatically aligning to a scoring target. This year we identified over 100 potentially valuable robot skills, spanning driving, game piece handling, endgame, sensing, and driver feedback. From there we mapped each skill to the scoring objectives it enables, identifying which skills are strictly required to score and which are beneficial but optional. This list is never fully exhaustive — sensor and indicator skills in particular tend to be broadly useful across the whole robot and don't always show up neatly in the required/beneficial columns — but it gives the team a shared vocabulary for what the robot actually needs to do before anyone starts arguing about mechanisms.

HOW THE GAME IS SCORED

The game revolves around two main tasks: scoring FUEL (balls) into a central HUB, and climbing a TOWER at the end of the match. FUEL scores one point per ball in both auto and teleop. The TOWER awards 10, 20, or 30 points for Level 1, 2, and 3 climbs respectively, with two robots able to reach Level 1 during auto.

There are three ranking points available beyond win/loss: **Energized** (100 FUEL scored), **Supercharged** (360 FUEL scored), and **Traversal** (50 total tower points in the match).We predict a typical winning match score is around 350 points. (obviously that threshold will increase at DCMP and Worlds if we get there)

One structural feature unique to this game is the 25-second **SHIFT** mechanic, which alternates which alliance has access to portions of the field. Getting caught without fuel inventory when the SHIFT flips is a real competitive threat, and the team identified "not being starved during the SHIFT window" as a specific design consideration.

WHAT THE TIME ANALYSIS TOLD US

With scoring objectives mapped, we ran time-based analysis to figure out which tasks are actually worth building around — weighing maximum point value against the time required to complete each cycle.

In auto, the highest-value routine is leaving the line, shooting preloaded balls, and climbing to Level 1 — worth 23 points in under 8 seconds. Going to pick up more balls in auto is viable from the depot, but efficiency drops quickly as travel distance increases. Neutral zone pickups in auto are not worth the time.

In teleop, the depot cycle — travel to depot, pick up 9 balls, return and score — is the most efficient path at roughly 112 match points per minute equivalent. Picking up a larger batch in one trip improves that further. The key lesson from studying prior-year metas is that **burst throughput matters more than continuous throughput**: being able to quickly deliver a large batch of fuel is more valuable than maintaining a slow, steady stream. Hopper capacity and fast intake speed are therefore more important design priorities than trying to sustain a non-stop firing loop.

Endgame is efficient in points-per-second, but the Traversal RP only requires 50 tower points — roughly one-seventh of a typical winning score, which is a historically low threshold. Climbing is worth doing, but not worth over-engineering.

WHAT WE DECIDED TO DESIGN FOR

Based on this analysis, the team agreed that the path to wins and ranking points runs through ball throughput. The Energized and Supercharged RPs are both driven by raw FUEL volume, and together they represent two of the three available ranking points. A robot that can run fast, high-capacity depot cycles and reliably reach at least Level 2 on the TOWER covers the vast majority of what any strong alliance needs.

Our target is to contribute to all three RPs without needing to be the sole contributor to any of them — a well-rounded robot that any alliance wants to pick rather than a narrow specialist. The primary design decision is a **large-hopper, high-throughput shooter robot on a fast swerve chassis targeting Level 2 climb**, with Level 3 remaining a stretch goal if the build schedule allows. If mechanical complexity forces a tradeoff late in the season, the climb is the first place we would simplify.

The drive speed targets that came out of this analysis are **16 ft/s in teleop** and **10 ft/s in auto**.

IDENTIFYING DESIGN CONSTRAINTS

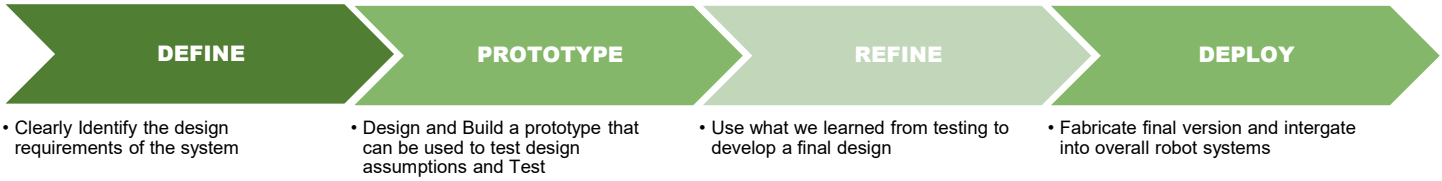
TEAM 1757 AXIOMS OF ROBOT DESIGN

Build Enough Hardware so that Software can continue to improve our Performance.

Be Able to Contribute to all the Ranking Points materially.

There will be lots of Robots; Vision and Partial Automation will Make you Stand out.

THE 1757 RAPID DEVELOPMENT MODEL



ARCHETYPE CONSIDERATIONS

WHAT WE WERE LOOKING FOR

After completing the scoring and time-based analysis, the team had a clear picture of what the robot needed to do: run efficient ball cycles, score reliably into the HUB, and reach at least Level 2 on the TOWER. The archetype review was meant to answer *how* — specifically what mechanisms and configurations from existing robots gave us confidence we could execute those objectives.

WHAT WE REVIEWED

Cranberry Alarm Ri3D was the most directly relevant reference. Their robot featured a large ball hopper, a turreted shooter, and a dedicated "ball kicker" mechanism to feed the shooter — a design concept the team found compelling. They also used mecanum wheels as a centering mechanism, which was an interesting detail worth noting. The downsides were significant though: the hopper was structurally unstable, they had no climb, and the battery was mounted awkwardly on the side of the robot. The team took the ball-kicker and turret concepts seriously but treated the hopper and overall packaging as cautionary examples of what not to replicate.

Penn State Ri3D was studied primarily for their climb prototyping work. The team found it useful as a reference for what a climbing mechanism looks like in practice, but their implementation required a large superstructure that the team wasn't willing to commit to given the weight and packaging constraints already in play. It validated that climbing to Level 3 is mechanically achievable but flagged that it comes at a real structural cost.

6800's 2022 robot was pulled in specifically as a turret reference. The team was drawn to the rotating turret concept as a way to decouple shooting direction from robot heading — which matters for a robot trying to score while also managing field positioning during the SHIFT windows. The noted limitation was the lack of an adjustable hood, which the team decided they did want in order to handle variable shooting distances without repositioning.

WHAT WE DECIDED, AND WHY

None of the reviewed archetypes were adopted wholesale. Instead the team pulled specific lessons from each:
The **turret with adjustable hood** concept came directly from studying 6800 and the Cranberry Alarm shooter. A fixed-rotation or fixed-angle shooter was explicitly ruled out — the team wanted the flexibility to shoot from different field positions without needing precise robot orientation, and the ability to adjust trajectory for distance. The conclusion sheet marked both fixed rotation and fixed angle as hard NOs.

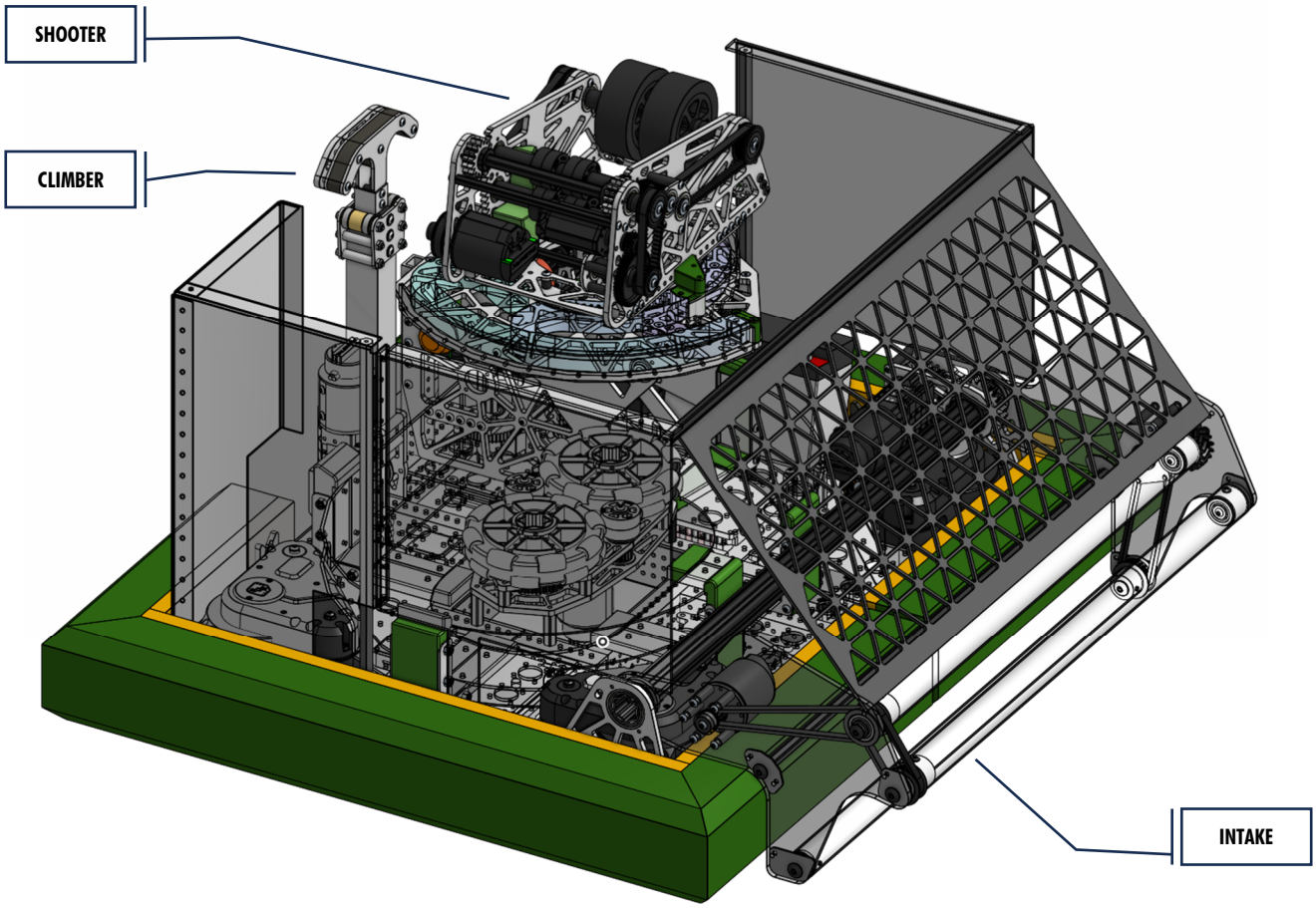
The **kicker-fed spindexer** concept for moving balls from the hopper to the shooter came from Cranberry Alarm's ball-kicker mechanism and Penn State's indexing work. A spindexer feeding a kicker that feeds the shooter became the chosen ball path — it supports the burst-throughput model and keeps the path single and predictable.

The **hopper design** was the most debated question coming out of the archetype review. Cranberry Alarm's unstable hopper made the team cautious about a large fixed hopper, and the design studies ultimately landed on a **retractable hopper** — one that deploys for intake and scoring but tucks back for climbing and collision protection. This added mechanical complexity but solved the climber integration and defense-vulnerability problems that the fixed hoppers in the reference robots suffered from.

On climbing, the team chose to target Level 2 as the reliable goal with Level 3 as aspirational, and to keep the climb mechanism relatively minimal — a single Falcon 500 winch with a MaxPlanetary gearbox stack — rather than building the kind of heavy superstructure Penn State's climb required.

The overall direction that came out of the archetype phase: a compact, retractable-intake swerve robot with a spindexer-fed turret shooter and a straightforward winch climb. Not the most ambitious design on paper, but one the team believed they could execute well within their build timeline.

FINAL ROBOT DESIGN



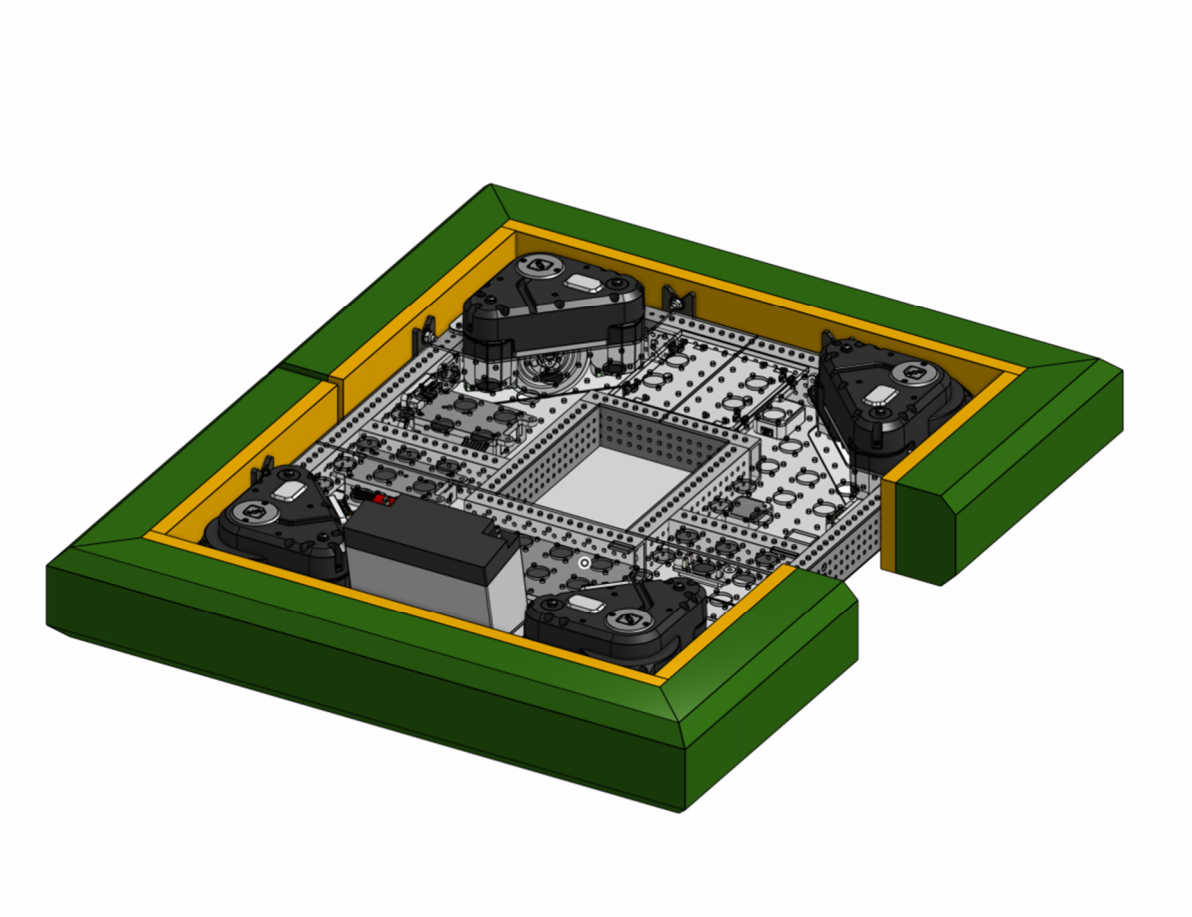
ISOMETRIC VIEW

FRONT VIEW

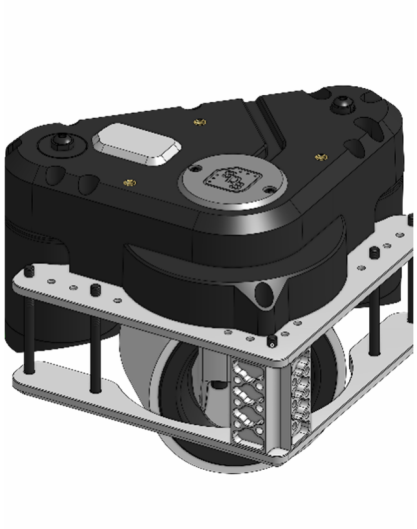
SIDE VIEW

REAR VIEW

MAJOR SYSTEM #1: DRIVE TRAIN



1.1 - SWERVE DRIVE MODULES



Orion SDS MK5iSwerve Module Configuration	
Module	SDS MK5i
Drive motor	Kraken X60 (×4)
Steer motor	Kraken X44 (×4)
Steer encoder	CTRE CANCoder (×4)
Drive gear ratio	L2 — (54:14) × (25:32) × (30:15)
Steer gear ratio	26:1
Wheel diameter	4 inches
Module weight (ready to run)	~6.1 lbs
Drive current limit	60 A (supply)
Steer current limit	40 A (supply)

Orion is Team 1757's fifth competition robot to use a swerve drive drivetrain configuration, continuing a lineage that stretches back to the 2022 season. A swerve drive is a holonomic drivetrain, meaning the robot can translate in any direction on the field — forward, sideways, or any diagonal — without first rotating the chassis to face that direction. Each of the four drive modules contains two independent motors: one that drives the wheel for propulsion, and one that rotates the entire wheel assembly to point it in any direction. Coordinating all four modules simultaneously through software allows the robot to move and rotate independently of each other, enabling a level of field mobility and defensive maneuverability that non-holonomic drivetrains cannot match.

MODULE UPGRADE: MK4I → MK5I

The 2022 robot ran SDS MK3i swerve modules. The 2023, 2024, and 2025 robots all ran SDS MK4i modules — three consecutive seasons on the same module generation. That description, however, understates the reality: it was the exact same four physical modules in every match, every competition, every practice session, every demo, and every offseason event from January 2023 through the fall of 2025. Every time a new robot was built, the MK4i modules were harvested off the previous year's chassis and bolted straight onto the new one. Across that entire period — hundreds of matches and countless hours of drive practice — there was not a single component failure. Not one belt snapped. Not one bearing failed. Not one gear stripped. They were, by any measure, one of the most reliable products the team has ever put on a robot.

That track record made upgrading a genuine decision rather than an obvious one. Walking away from something with a three-year perfect service record carries real risk. For 2026, the team upgraded to the SDS MK5i (Swerve Drive Specialties, swervedrivespecialties.com). The MK5i represents a ground-up redesign of the SDS module platform and addresses several categories of maintenance burden that, while manageable on the MK4i, were real costs across those three seasons. The reasoning is straightforward: if the MK4i proved essentially unkillable in three years of continuous use, a module that is more robustly enclosed, fully gear-driven, and easier to service may prove to be in a different category entirely.

STEERING SYSTEM: BELT TO GEAR

The most mechanically significant change is the steering drive system. The MK4i used a belt-driven steering mechanism. Belts are lightweight and smooth but require periodic tension checks, can stretch or snap under aggressive defense, and are difficult to inspect visually without disassembly. The MK5i replaces the belt entirely with a **fully gear-driven steering system**. Gears do not stretch, do not require tension adjustment, and are significantly more robust under the lateral loading that occurs during hard defense and collision. For a team that runs its drive base across four competition seasons and expects contact, this is a meaningful durability improvement.

Debris Protection

Four years of competition experience on the MK4i produced a consistent maintenance task: cleaning carpet fuzz, field tape fragments, and field debris out of the bevel gears and steering mechanism. The MK5i addresses this directly with several design changes:

- **Enclosed bevel gears** at the wheel prevent debris from entering the bevel gear mesh and contaminating the grease
- **Drive and steering gears enclosed** in a molded glass-reinforced nylon housing that keeps grease in and debris out
- **Sealed main steering bearing** prevents carpet fuzz and fine particulate from entering the bearing race
- **Recessed and enclosed steering encoder** with an integrated cable tie mount to protect wiring from snagging

Collectively these changes are expected to significantly reduce the between-match maintenance time that the MK4i required at competition.

ON-AXIS STEERING ENCODER

The MK5i uses an on-axis steering encoder — the CANCoder sits directly on the steering axis with zero mechanical backlash between the encoder and the wheel's rotational position. This means the encoder reads true wheel angle with no gear lash error. The additional benefit is that the module can be fully disassembled and reassembled without resetting the encoder offset — the magnet is embedded in the main steering gear and stays aligned through any service procedure. On the MK4i, belt tension adjustments and internal service could shift the encoder relationship and require re-zeroing at the robot's next enable.

CONSTRUCTION & ASSEMBLY

The MK5i is built entirely from 7075-T6 aluminum plates and steel drive gears — there are no 3D printed structural parts anywhere in the module. All fasteners are unified to 10-32 threads, meaning a single hex key services the entire module. A quick-change drive gear ratio system allows the drive speed to be changed by swapping only the motor pinion gear, with no internal disassembly of the module required.

GEAR RATIO CONFIGURATION

The L2 gearing ratio was selected as the team's standard from prior seasons and carried forward on the MK5i. It provides a balance of top-end speed and acceleration that suits the team's drive style and the field size of typical FRC games. The gearing can be changed in the field by swapping the drive motor pinion — one of the MK5i's new serviceability features — without any other module disassembly.

WHY THIS IS A SERIOUS UPGRADE

The MK4i never failed the team — but it did require attention. Belt tension checks before matches, debris cleaning between matches, and careful inspection after hard defense were accepted as routine maintenance costs across three seasons. None of it was ever catastrophic; it was just time, and time at competition is always scarce.

The MK5i eliminates the belt entirely, encloses all the gear meshes that previously ingested debris, seals the main bearing, and unifies all hardware to a single fastener size. On paper that reads as incremental improvement. In practice, for a team that ran the same four modules for three straight years without a single failure, it means putting an even more robust foundation under a drivetrain that was already a proven competitive strength.

The team's honest expectation going into 2026: if the MK4i was essentially unkillable, the MK5i may not have a failure mode we can find.

1.2 - ELECTRONICS SUBSYSTEM

PHYSICAL LAYOUT

Continuing a trend from the last three years, Orion uses an underslung electronics system. All electronics are mounted on a pan installed on top of the drive rails and wired upside down beneath it. This arrangement provides wide-open access to the full electronics bay when the robot is tipped carefully onto its side — no robot components need to be removed to reach any electrical connection, fuse, or controller. The electronics are fully enclosed from above by the robot's superstructure and shielded from the sides, protecting them from stray game pieces and contact with other robots.

The one exposure this design creates is downward toward the floor. With the robot's low ground clearance, this is managed with a polycarbonate skid plate fastened to the underside of the drive rails with countersunk 10-32 bolts. The skid plate is removable in under two minutes.

For 2026, the skid plate thickness was reduced from ¼" to ⅛" polycarbonate. Two factors drove this decision: the robot was up against its weight budget, and the team assessed that the 2026 game was unlikely to produce meaningful amounts of field debris that the electronics bay would need to be protected against. In hindsight, we were probably always being over-cautious with ¼" material — for context, 0.25" polycarbonate is rated to stop a .22 caliber bullet, which is considerable overkill for the occasional stray bolt bouncing around an arena floor. That said, ⅛" is a meaningful reduction in margin, and the performance of the thinner skid plate is something we will be watching closely throughout the season. For reference, the robot weighed in at **112.8 lbs** at DCMP#1 — within the 115 lb limit, but with little room to spare, which validates the weight-driven reasoning behind the change.

NOTABLE CHANGES FROM 2025

Vision system: The 2026 vision configuration is the culmination of four years of iteration. In 2022, the robot carried no vision system — positioning relied entirely on wheel odometry dead reckoning and driver skill. The 2022 software stack was award-recognized for other qualities, but vision was not part of it. In 2023, the team introduced their first competition vision system: a single Limelight 2 camera in a fixed chassis mount. It was reliable and consistent, but field coverage was limited

to whatever direction the robot was facing. In 2024, the team attempted a full multi-camera field positioning system using USB 2.0 OV9281 monochromatic global-shutter cameras mounted in the corners of the robot frame, each feeding into OrangePi coprocessors running a custom PhotonVision pipeline. The ambition was right but the honest reason it never fully worked was scope — the system demanded more development and testing hours than the team could dedicate that season. Latency inconsistencies and calibration drift were the visible symptoms; insufficient development time was the root cause. In 2025, the team stepped back to two Limelight 3 cameras in fixed mounts, running the same underlying PhotonVision pipeline architecture but with a far more manageable hardware footprint. The system restored reliability, though the team placed less overall reliance on vision output than in prior seasons while confidence was rebuilt.
The 2026 system is a synthesis of all of those lessons. A single ThriftyCam (Thriftybot TTB-0241) is mounted on the turret rather than fixed to the chassis, running a PhotonVision software pipeline on a dedicated coprocessor. Because the turret actively tracks the scoring hub, the camera maintains consistent line of sight to the hub tags regardless of robot heading — capturing the coverage benefit the 2024 system was trying to achieve, but through mechanical positioning rather than camera count. The trade-off is increased complexity in pose estimation, which the software team addressed with a custom latency-compensated estimator described in the software section.

Motor selection: The 2026 robot uses a mix of Kraken X60, Kraken X44, and Falcon 500 motors across its subsystems. For the specific motor selection and rationale for any individual mechanism, please refer to that subsystem's dedicated writeup elsewhere in this binder.

Electronics System — Major Components

Core Control

Qty	Manufacturer	Component	Notes
1	National Instruments	RoboRIO 2	Team 1757, static IP 10.17.57.2, firmware 6.0.0f1
1	REV Robotics	Power Distribution Hub (PDH)	CAN ID 0
1	Vivid Hosting	VH-109 Radio	Typically assigned 10.17.57.1; actual IP may vary

CAN Infrastructure

Qty	Manufacturer	Component	Notes
1	CTRE	CANivore	Dedicated high-speed CAN bus for all drive components
1	CTRE	Pigeon 2.0 IMU	CAN ID 20, on CANivore bus
1	CTRE	CANdle	CAN ID 21, on CANivore bus; drives external RGBW LED strip
4	CTRE	CANCoder	CAN IDs 40–43, on CANivore bus; absolute steer encoders

Vision & Coprocessors

Qty	Manufacturer	Component	Notes
1	Thriftybot	ThriftyCam (TTB-0241)	Mounted on turret; connected to OrangePi 5 coprocessor #1
2	Radxa	OrangePi 5	Running PhotonVision pipeline; #1 active (turret cam), #2 installed, camera unpopulated
2	Redux Robotics	Zinc-V	One per OrangePi 5; regulated 5V coprocessor power supply

Coprocessor note: Two OrangePi 5 coprocessors are installed on the robot, both powered by dedicated Redux Robotics Zinc-V regulated power supplies (shop.reduxrobotics.com). Only the first is active in the 2026 configuration — it runs the PhotonVision pipeline and handles all onboard vision processing via the turret-mounted ThriftyCam. The second OrangePi 5 was installed to preserve flexibility: if the team determines mid-season that a second field-oriented camera is needed for AprilTag identification independent of the turret, the infrastructure to support it is already in place with no additional wiring work required. As of initial competition deployment, the second coprocessor has no camera connected and remains inactive.

Drive Motors — CANivore Bus

Qty	Motor	Controller	CAN IDs	Function
4	Kraken X60	Integrated TalonFX	30, 31, 32, 33	Swerve drive wheels
4	Kraken X44	Integrated TalonFX	50, 51, 52, 53	Swerve steering

All eight drive TalonFX controllers are on the CANivore bus. Supply current is limited to 60 A on drive and 40 A on steer.

Mechanism Motors — Standard CAN Bus

Qty	Motor	Controller	CAN ID	Mechanism	Current Limit
1	Kraken X60	Integrated TalonFX	1	Intake pivot	40 A
1	Kraken X60	Integrated TalonFX	2	Intake roller	60 A

1	Falcon 500	Integrated TalonFX	3	Spindexer (far)	30 A
1	Falcon 500	Integrated TalonFX	4	Spindexer (close)	30 A
1	Falcon 500	Integrated TalonFX	5	Indexer kicker (lower)	30 A
1	Falcon 500	Integrated TalonFX	6	Indexer kicker (upper)	30 A
1	Kraken X60	Integrated TalonFX	7	Flywheel	80 A
1	Kraken X44	Integrated TalonFX	8	Hood	20 A
1	Kraken X44	Integrated TalonFX	9	Turret	40 A
1	Falcon 500	Integrated TalonFX	10	Climber	40 A

MOTOR SUMMARY

Motor	Count	Applications
Kraken X60	7	Swerve drive (×4), flywheel, intake pivot, intake roller
Kraken X44	6	Swerve steer (×4), turret, hood
Falcon 500	5	Indexer spindexer (×2), indexer kicker (×2), climber
TalonFX controllers total	**18**	All motors use integrated TalonFX

Motor count note: Orion's 18 TalonFX controllers make it the most motor-intensive robot Team 1757 has ever fielded. The previous high was 16 motors on the 2022 robot. The increase to 18 is attributable almost entirely to the 2026 indexer design, which uses four motors (two spindexers, two kickers) compared to the single-motor indexer of prior seasons.

CAN BUS ARCHITECTURE

The robot operates two separate CAN buses. The **CANivore** bus carries all time-critical drive system devices — eight swerve TalonFXs, four CANCoders, the Pigeon 2.0, and the CANdle — providing the higher bandwidth and lower latency that high-frequency swerve odometry and vision-based pose estimation require. The **standard roboRIO CAN bus** carries all mechanism controllers (IDs 1–10) and the PDH. This separation ensures that a fault or high load on the mechanism bus cannot interfere with drive system timing.

The CANivore bus also enables a CTRE feature called **fused sensors (Pro)**. In a standard motor controller configuration, the steering encoder is read as a separate CAN device and the controller uses that reading with some latency and potential for intermittent error. With fused sensors, the TalonFX motor controller fuses the CANCoder's absolute position data directly into its own control loop at extremely low latency — effectively treating the encoder at the end of the steering gear ratio as if it were the motor controller's own internal sensor. This means the controller is always aware of the true wheel angle rather than the motor shaft angle, dramatically reducing the effect of any mechanical backlash in the steering gearbox on the closed-loop response. The result is tighter, more responsive steering that holds angle more accurately at high speed and during aggressive maneuvers.

For 2026, the CANivore bus wiring for the drive system was reworked using the Swyft Robotics CANnect wiring system (swyftrobotics.com). Rather than running individual twisted-pair CAN wires between each swerve module, each module's CAN devices are terminated locally at a WAGO lever-lock terminal block, and the four modules are then daisy-chained together using Cat6 ethernet cable. Cat6 is significantly more durable than conventional FRC CAN wire — it is less prone to insulation damage from repeated flexing, less likely to break at crimps in hard-to-reach locations, and far easier to route cleanly through the chassis. The result is a drive system CAN harness that is more robust to the mechanical stress of a competition season and meaningfully easier to inspect and service than what the team has run in prior years.

1.3 –ROBOT STATE INDICATORS

OVERVIEW

Orion carries a strip of RGBW LEDs mounted around the exterior of the robot frame, driven by a CTRE CANdle controller connected to the robot's main CAN bus. The strip consists of 18 external RGBW elements supplemented by 8 LEDs onboard the CANdle itself, giving 26 total addressable lights visible from all sides of the robot.

The use of RGBW strips — which include a dedicated white channel in addition to red, green, and blue — rather than standard RGB strips is a deliberate hardware choice. RGBW allows the yellow and orange colors used in the system's warning states to be reproduced at full saturation and brightness under arena lighting conditions, where RGB approximations of those colors can appear washed out or ambiguous at distance.

A DESIGN PRINCIPLE SINCE 2022

Team 1757 has used addressable LED strips as a structured driver communication tool on every competition robot since the 2022 season. The core principle has remained the same across all four years: the robot should be able to communicate its current state to the people operating it without requiring them to look away from the field.

A primary driver tracking a fast-moving robot across a full-size competition field cannot safely glance at a laptop screen mid-play. A visible color change on the robot itself closes that information loop without splitting attention. What has evolved year to year is the *audience* and *content* of that communication. In seasons where the game required close coordination between

the robot and a human player at the field perimeter — such as during human player station interactions — the LEDs doubled as a signal to the human player: indicating when the robot was ready to receive a game piece, when it was not, or when a specific action was needed. The specific colors and triggers were redesigned each season to match that year's game, but the strip hardware and the underlying design philosophy carried forward continuously.
The 2026 implementation is the most complete expression of this system to date, adding animation style as a second simultaneous communication channel alongside color — allowing the robot to signal both its readiness state and an upcoming game event at the same time.

WHAT THE LIGHTS SHOW

The LED strip operates in one of eight distinct states at any given moment during a match. The driver reads **color** to understand whether the robot is ready to fire, and reads **animation** — solid versus flashing — to understand whether a hub shift is approaching within the next few seconds.

Robot Situation	Color	Animation	What It Means
Robot e-stopped	Red	Fast strobe	Immediate safety signal — robot is hard-stopped
Disabled, voltage drop detected	Orange	Fast strobe	Power event occurred — alert pit crew before next match
Disabled, normal idle	All colors	Rolling rainbow	Robot at rest, no fault condition
First 5 seconds after autonomous ends	Blue	Slow fade-out	Autonomous activity indicator (see below)
Enabled, shooter aligning or spooling	Yellow	Solid	Robot is working toward a shot but not yet ready
Enabled, locked on and ready to fire	Green	Solid	Cleared to fire — all systems at target
Enabled, not ready — hub shift imminent	Yellow	Fast strobe	Aligning, but the active hub is about to change
Enabled, locked on — hub shift imminent	Green	Fast strobe	Ready to fire, but the target window is closing

The flashing states correspond to the final three seconds before each hub shift boundary in the 2026 game. The strip flashes rather than going solid to give the driver a spatial, peripheral signal that time is running short — the same alert that also triggers a haptic rumble on both controllers. A flashing green strip is the highest-urgency state in the match: the robot is ready to score, but will lose its active target within seconds.

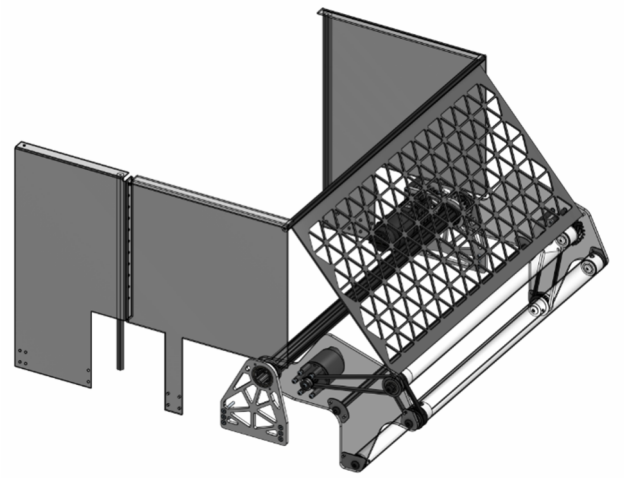
The Post-Autonomous Blue Fade

When the robot transitions from the autonomous period to disabled, the strip displays solid blue and then fades progressively to dark over a five-second window. As the fade progresses, fewer and fewer of the 18 external LEDs remain lit, with the lit portion shrinking from one end of the strip toward the other.
This behavior gives the pit crew and coaching staff an immediate at-a-glance indicator of how the autonomous period went. A strip that is still nearly fully blue when the field goes to teleop-enable indicates the robot was active and moving until the end of autonomous. A strip that has already faded to dark or is mostly dark indicates the autonomous routine completed early or the robot stopped before the period ended. This assessment can be made in the three seconds between autonomous end and teleop enable, before any post-match log review, and without any communication from the drive team.
If the robot detects a voltage drop during the match — indicating the battery was pushed near its limits — the orange strobe replaces the normal rainbow when the robot disables. The pit crew sees this immediately when the robot drives off the field, flagging the battery for replacement or load testing before the next match without waiting for any verbal report from the drive team.

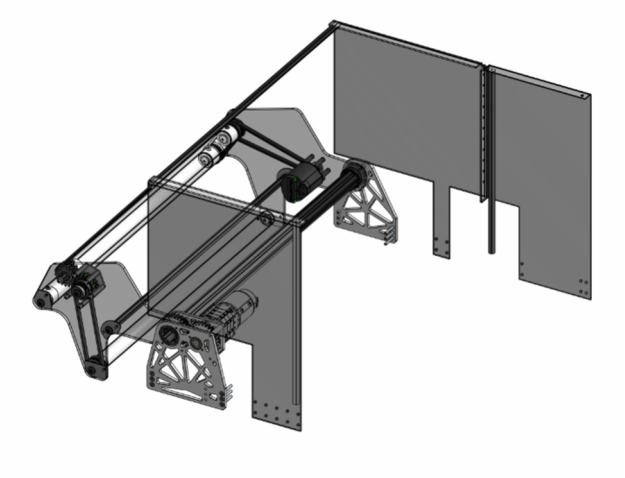
How This Supports the Drive Team

Team 1757 operates with a two-driver structure. The primary driver keeps eyes on the field at all times — robot position, defense, game piece locations, alliance partner coordination. The secondary driver (operator) watches the dashboard, which provides detailed readouts of mechanism states, sensor values, and system health, and communicates relevant information verbally to the primary driver.
The LED system is designed specifically to support that division of labor. The single most important piece of real-time information — *is the robot ready to score, and is the target window about to close* — is always available to the primary driver without requiring any verbal relay from the operator. Everything else flows through the dashboard and the verbal channel. The LEDs do not try to communicate everything; they communicate exactly the one thing the primary driver needs to know while their eyes are on the robot.

MAJOR SYSTEM #2: INTAKE



ISOMETRIC VIEW



ISOMETRIC VIEW

ISOMETRIC VIEW ROLLER ONLY

The intake is responsible for collecting FUEL (game pieces) from the field and delivering them to the spindexer. The team's scoring analysis identified "intake multiple balls at a time from the ground" as a required capability for the FUEL and ENERGIZED RP objectives, and "pick up balls from the depot" as a high-value additional mode for collecting from the human player station. The design selected a full-width deployable roller intake — extending out to the robot's frame perimeter when active and retracting flush with the robot body when not in use.
The retractable design was chosen over a fixed deploy-once intake for several reasons identified in the design studies: it protects the intake during defense and transit, it allows the robot to fit within the starting configuration without using a bumper gap for the intake, and the pivot motor can be used to actively unjam game pieces by varying the deploy angle.

MECHANISM OVERVIEW

The intake consists of two independent mechanisms working together: a **pivot arm** that deploys and retracts the intake assembly, and a **roller** that spins to capture game pieces and draw them into the robot.
The pivot arm swings the intake through a 125.5° arc — from fully deployed at 0° (roller at ground level, contacting game pieces) to fully retracted at 125.5° (roller raised up and flush with the robot). The arm carries the roller assembly on its end, so the roller's position relative to the field is entirely determined by where the pivot is pointing. The arm is approximately 4.843 inches long from the pivot axis to the roller centerline and the moving assembly weighs approximately 7 lbs.
The roller is a powered wheel that spins to sweep game pieces off the floor and into the robot's ball path toward the spindexer. It runs independently of the pivot — the pivot controls where the roller is, and the roller motor controls whether it is collecting, neutral, or reversing.

PIVOT

PHYSICAL DESCRIPTION

The pivot arm is driven through a **64:1 gear reduction**, the highest ratio of any single-stage mechanism on the robot after the hood. This high reduction is necessary because the arm is fighting gravity throughout its range of motion — as the arm deploys downward from its retracted position, gravity acts against the motor, and as it retracts upward, gravity assists. The 64:1 ratio ensures the motor has sufficient torque to move and hold the arm at any position in its range, including holding the roller at ground contact pressure during active intaking without back-driving.

The pivot starts fully retracted (125.5°) and must be actively commanded to deploy. It has four defined positions used during normal operation:

Position	Angle	Use
Retracted	125.5°	Transit, defense, starting configuration
Safe	100°	Intermediate hold while waiting for turret to clear
Oscillate	15°	Slight upward tilt during active shooting to agitate game pieces
Deployed	0°	Full ground contact, active intaking
Depot	5°	Human player station pickup height

The **safe position** (100°) is a direct consequence of the turret/intake collision envelope. When the turret is away from its 90° safe zone, the intake is held at 100° — partially deployed but above the collision boundary — and waits for the turret to return to safe before completing its travel. This position was determined from the CAD model as the highest angle at which the intake structure still clears the turret at all rotation positions.

The **oscillate position** (15°) is a small upward tilt from fully deployed, used during the shooting sequence to gently agitate game pieces in the intake chute and encourage consistent delivery to the spindexer. The pivot cycles between 0° and 15° slowly while the rollers run, preventing game pieces from stacking or jamming at the transition between intake and spindexer.

MOTOR & ELECTRICAL SPECIFICATION

Parameter	Value
Motor	Kraken X60
Motor controller	Integrated TalonFX
CAN ID	1 (standard roboRIO bus)
Gear ratio	64:1
Travel range	0° (deployed) to 125.5° (retracted)
Starting position	125.5° (retracted)
Position tolerance	±2°
Bump increment	5°
Supply current limit	40 A
Neutral mode	Coast

ROLLER

PHYSICAL DESCRIPTION

The roller is a single driven wheel at the end of the pivot arm that contacts game pieces on the field and draws them into the robot. It is driven through a two-stage gear reduction — (36:16) × (24:18) = 3:1 overdrive — spinning the roller faster than the motor shaft to maximize surface speed at the wheel contact patch. At design speed, the roller produces approximately 23,560 in/min of surface speed, aggressively sweeping game pieces off the floor.

The roller wheel diameter is 1.5 inches. The BOM lists a MaxPlanetary gearbox assembly in the intake, which is consistent with the 3:1 ratio staged through two gear pairs.

MOTOR & ELECTRICAL SPECIFICATION

Parameter	Value
Motor	Kraken X60
Motor controller	Integrated TalonFX
CAN ID	2 (standard roboRIO bus)
Gear ratio	(36:16) × (24:18) = 3:1 overdrive
Wheel diameter	1.5 in
Surface speed (design)	~23,560 in/min
Forward voltage	6.0 V
Reverse voltage	−6.0 V
Supply current limit	60 A

The 60 A roller current limit is notably higher than the pivot's 40 A, reflecting that the roller can stall against a game piece pile without stalling the motor — the higher current headroom allows it to push through resistance that a lower limit would cause the controller to pull back from.

TURRET / INTAKE COLLISION INTERLOCK

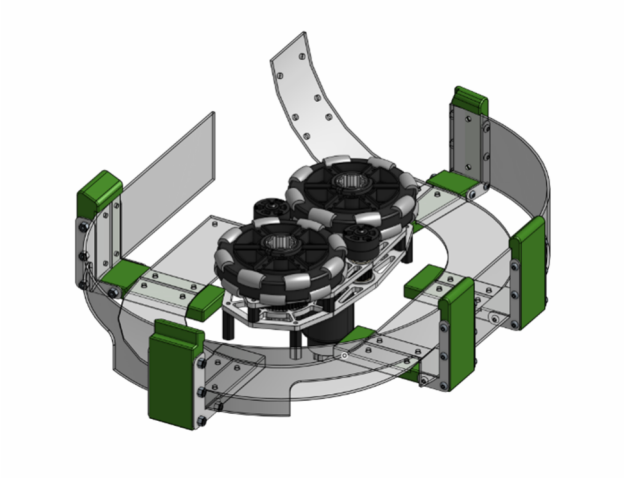
The intake pivot shares a physical collision envelope with the turret. When the intake is deployed beyond **70°** from its retracted position, portions of the intake structure can contact the turret if the turret is not near its 90° forward position. This boundary was established from the CAD model.

Both mechanisms monitor each other's position in software and restrict their own motion when the other is in a dangerous position. The intake's side of this interlock holds it at the 100° safe position until the turret is confirmed to be clear. This protection is entirely software-enforced and operates independently of any command or higher-level system. For the full implementation detail, see Section 2.3 of the Software Section.

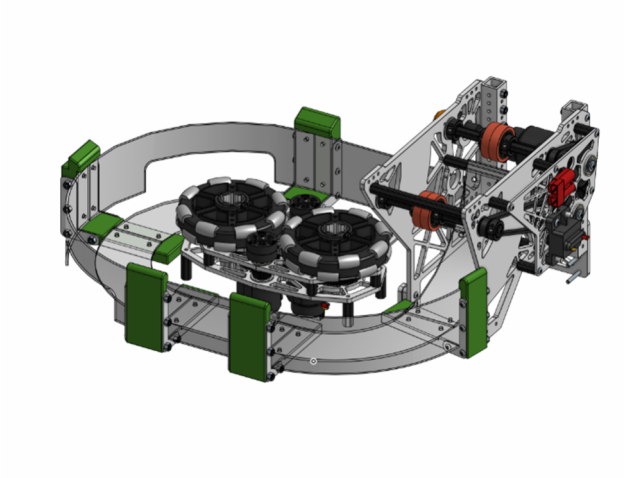
INTAKE SYSTEM SUMMARY

	Pivot	**Roller**
Motor	Kraken X60 ⚠	Kraken X60
CAN ID	1	2
Gear ratio	64:1	3:1
Travel	0°–125.5°	Continuous
Current limit	40 A	60 A

MAJOR SYSTEM # 3: SPINDEXER



ISOMETRIC VIEW – SPINDEXER ONLY



ISOMETRIC VIEW – WITH KICKER SUBSYSTEM

The primary scoring objective in REBUILT is accumulating FUEL (game pieces) in the HUB as rapidly as possible. The Energized RP threshold requires 100 FUEL scored, and the Supercharged RP threshold requires 360. The team's time-based analysis identified continuous, uninterrupted shooting as the highest-value capability in the game — the scoring analysis flagged "continuous stream of balls to the shooter" and "don't get balls stuck in the bot" as critical requirements for the FUEL and ENERGIZED RP objectives.
The spindexer was selected specifically to satisfy those requirements. A reliable, jam-resistant staging mechanism that can hold and advance game pieces without interrupting the shooting sequence was identified as the difference between a robot that scores continuously and one that scores in bursts. The design was evaluated against alternatives including a linear singulator and a hopper-direct path, with the spindexer selected as the intake/possession approach in the design conclusions for its combination of jam resistance, game piece retention, and compatibility with a continuous shooting cadence.

Mechanism Overview

The spindexer is a rotating **oval disc** mechanism that stages game pieces between the intake and the kicker. The oval geometry — rather than a circular disc — maximizes the storage volume available within the robot's frame footprint, allowing the spindexer to hold more game pieces simultaneously for a given structural envelope. Game pieces delivered by the intake fall into pockets on the disc; the disc rotates to advance them toward the kicker feed point on demand. At rest, game pieces are held passively in their pockets — the disc does not need to be actively driven to retain them.
The mechanism uses a two-stage wheel system: a set of large primary drive wheels powered by the two spindexer motors, and a set of smaller secondary wheels driven mechanically from the primary stage rather than by independent motors. This gives the spindexer both a large-diameter outer contact surface for moving game pieces around the disc and a smaller inner surface for guiding them, without requiring additional motors.

Mechanical Specification

Parameter	Primary Wheels	Secondary Wheels
Part	REV 6" MAXSpline Omni Wheel (REV-21-2476)	WCP 1.625" Flex Wheel (WCP-1288, 45A)
Drive source	2× Falcon 500 (direct)	Mechanically driven from primary stage
Wheel diameter	6.0 in	1.625 in
Gear reduction	42:60 (0.70:1 — underdrive)	42:18 (2.33:1 from primary)
Output RPM	~4,800	~4,800
Surface speed	~63,335 in/min	~57,177 in/min

The 42:60 primary reduction (input gear smaller than output — underdrive) provides increased torque at the disc relative to a direct-drive arrangement, appropriate for a mechanism that must reliably advance game pieces under varying load conditions. The secondary wheels are driven at a 42:18 ratio from the primary gear. The gear ratios for both wheel stages were specifically chosen so that their surface speeds match — ensuring the game piece is driven evenly from both contact surfaces simultaneously with no tendency to deflect or spin in place.

Electrical Specification

Parameter	Value
Motor (×2)	Falcon 500

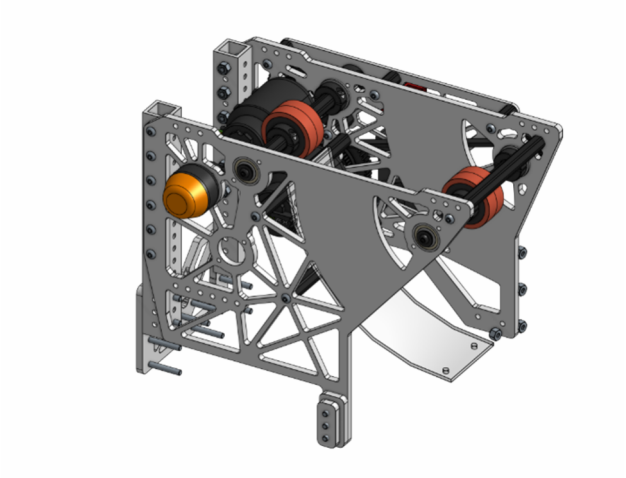
Motor controller (×2)	Integrated TalonFX
CAN IDs	3 (far), 4 (close)
CAN bus	Standard roboRIO bus
Neutral mode	Coast
Supply current limit	30 A
Operating voltage	3.0 V

Both motors drive the disc in the same direction. The low operating voltage (3.0 V against a nominal 12 V bus) reflects a design iteration from testing. Initial runs used a higher voltage, but this caused feed reliability problems at the shooter — the spindexer was advancing game pieces faster than the kicker and flywheel could process them consistently. The voltage was reduced until the feed cadence matched the shooter's throughput. Running well below the motor's capability also keeps current draw low and allows the disc to decelerate smoothly when power is removed without jolting a staged game piece out of its pocket.

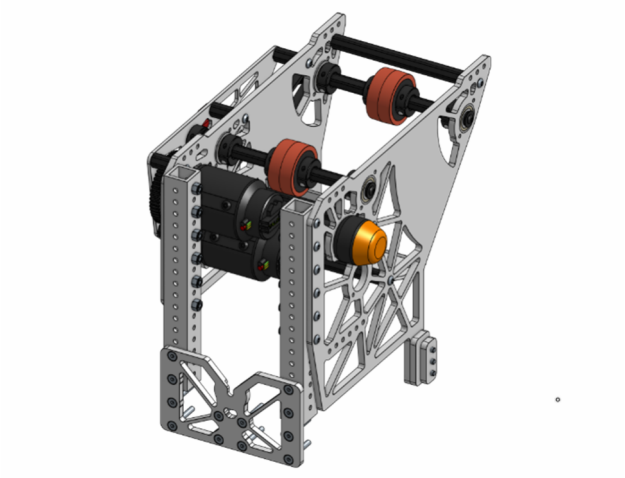
Integration with the Kicker

The spindexer and kicker feed into each other directly — the spindexer's exit point is the kicker's entry point. When both activate simultaneously, the spindexer advances a game piece into the kicker's roller interface at the same moment the kicker rollers drive it toward the flywheel. The mechanism geometry is designed so that simultaneous actuation produces the correct hand-off timing without any sensor feedback between the two stages.

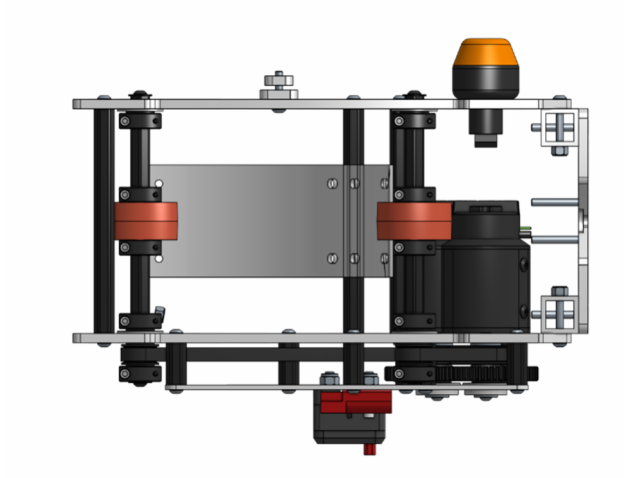
MAJOR SYSTEM # 4: KICKER



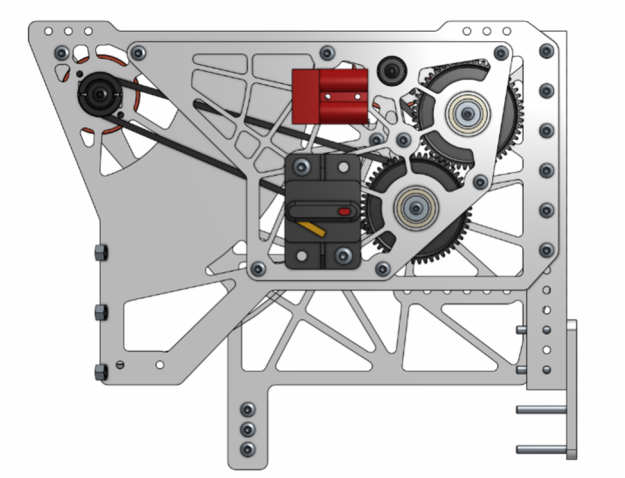
ISOMETRIC VIEW



ISOMETRIC VIEW



TOP VIEW



SIDE VIEW

The kicker is the final stage of the game piece path between the spindexer and the flywheel. The team's scoring analysis identified "continuous stream of balls to the shooter" as a critical requirement — scoring continuously at the hub requires the hand-off from staging to shooter to be fast, repeatable, and reliable match after match. The kicker is the mechanism that makes that hand-off happen.

The two-roller opposed-drive configuration was selected over a single-roller design because it is self-centering: it applies equal force from both sides of the game piece regardless of minor positional variation in how the spindexer delivered it, producing a consistent and repeatable exit velocity into the flywheel. A single-roller design can deflect the piece laterally on delivery; the opposed pair constrains it along the correct path.

Mechanism Overview

The kicker consists of two counter-rotating rollers — one above and one below the game piece path — that pinch the game piece and drive it into the flywheel intake. Both rollers are driven at the same surface speed in opposing directions so that from the game piece's perspective both surfaces are always pushing it forward along the feed path.

Each roller is driven by its own Falcon 500 motor. The two motors are wired so that a single command drives both rollers in a cooperative counter-rotating direction — one clockwise, one counter-clockwise — ensuring both surfaces always push the game piece in the same linear direction through the feed path.

There is a deliberate **deadzone** — a short free-flight gap — between the kicker's exit point and the flywheel's grip point. This gap is intentional: it gives the game piece a moment to align and normalize its trajectory before entering the flywheel, reducing the sensitivity of the shot to minor variations in how the kicker delivered the piece.

Mechanical Specification

Parameter	Value
Motor per roller	1× Falcon 500
Motor controllers	2× Integrated TalonFX

Roller diameter	2.0 in
Gear reduction	50:24 (2.08:1 — overdrive)
Output RPM	~4,800
Surface speed	~62,832 in/min per roller

The 50:24 gear ratio is an overdrive — the rollers spin faster than the motor shafts. This is the deliberate inverse of the spindexer's underdrive: where the spindexer prioritizes torque and controlled positioning, the kicker prioritizes surface speed to ensure the game piece accelerates cleanly into the flywheel intake rather than being dragged or stalled at the transition. At ~62,832 in/min surface speed, both rollers are running matched to each other, preventing any lateral deflection of the game piece as it passes through.

ELECTRICAL SPECIFICATION

Parameter	Value
Motor (×2)	Falcon 500
Motor controllers (×2)	Integrated TalonFX
CAN IDs	6 (upper), 5 (lower)
CAN bus	Standard roboRIO bus
Neutral mode	Coast
Supply current limit	30 A
Operating voltage	3.0 V

Positive voltage is always defined as the forward (feed) direction on both motors. The motor controller inversion configuration is set such that a positive voltage command drives both rollers in the correct cooperative direction — one clockwise, one counter-clockwise — advancing the game piece through the mechanism. Coast neutral mode is used because an abrupt stop while a game piece is mid-transit could jam it against the flywheel entry — coasting allows both rollers to decelerate smoothly.

INTEGRATION WITH THE SPINDEXER AND FLYWHEEL

The kicker sits between the spindexer and the flywheel in the game piece path. It has no sensor to detect whether a game piece is present or has cleared — its job is purely mechanical: when activated, drive the game piece at consistent velocity into the flywheel intake. The decision of when to activate it is managed by the shooting system, described in the Software Section.

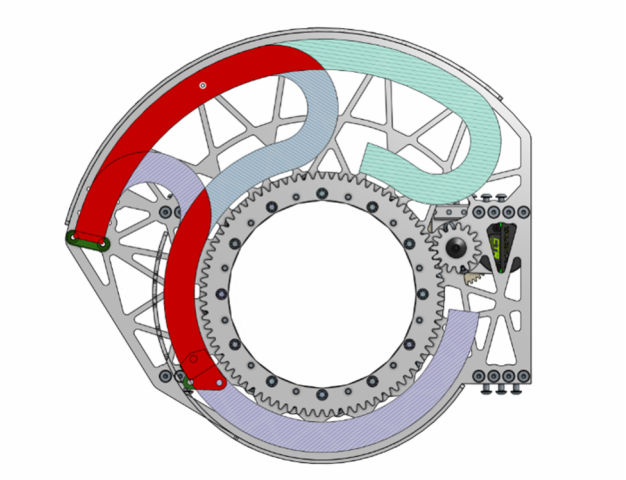
WHY FOUR MOTORS TOTAL FOR THE INDEXER?

The rationale for both motor pairs is the same: robustness and reliability margin. The spindexer's two motors driving the disc in tandem provide more torque than a single motor, reducing the risk of stalling on a poorly delivered game piece. The kicker's two opposed rollers provide self-centering force that a single-roller design cannot — regardless of exactly where the spindexer places the game piece laterally, both rollers grip it and drive it straight.

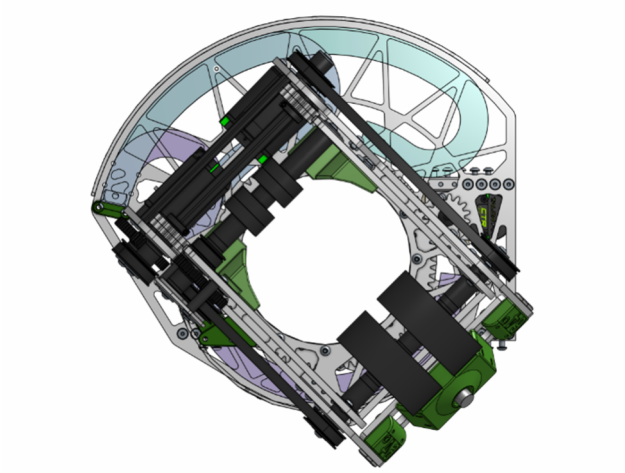
There is also an efficiency argument for two motors rather than one. High-speed spinning mechanisms are more power efficient when the load is distributed across multiple motors operating within their optimal torque band, rather than a single motor being pushed toward its limits. With Orion carrying the most spinning mechanisms in the team's history — flywheel, spindexer, kicker, and intake rollers all running simultaneously — power efficiency translates directly into sustained battery voltage and consistent performance across a full two-minute match. Both mechanisms run at 3.0 V — well below their motor capability — meaning the additional motor count costs relatively little in weight and electrical load while substantially increasing the reliability and efficiency of a feed path that must work every cycle of every match.

MAJOR SYSTEM # 5: TURRET

ISOMETRIC VIEW



CUT AWAY VIEW SHOWING THE WRAPING OF THE IGUS CABLE CHAIN AT VARIOUS TURRET ROTATIONS



Top Down View – Showing Shooter On Top

The turret is one of the most consequential design decisions on Orion and the mechanism that most directly enables the team's core competitive strategy. The scoring analysis identified "align shooter while driving" and "move while spin" as high-value capabilities, and the conclusions sheet reached a clear decision: a turreted shooter with up to 270° of rotation. The team's minimum requirement was 270° of turret travel — enough to cover the complete hub-facing arc during normal play. Continuous rotation (360°) and greater-than-360° designs were evaluated as nice-to-haves but were not required, and the additional mechanical complexity of slip rings or continuous-rotation gearing was not justified by the marginal coverage gain over a bounded 270° design.

The rationale goes deeper than mechanism simplicity. With the camera mounted on the turret, the robot gains consistent hub-tag visibility throughout the match regardless of chassis heading. This is what enables shoot-on-the-move: the turret tracks the hub continuously, the camera confirms pose continuously, and the robot never needs to stop and orient its chassis toward the target to take a shot.

MECHANISM OVERVIEW

The turret is a rotating upper platform that carries both the shooter assembly (flywheel, hood, and upper rollers) and the vision camera. It sits atop the robot's main structure and rotates about a vertical axis, driven by a Kraken X44 motor through a two-stage gear reduction. Software limits prevent the turret from rotating beyond its mechanical travel range, and a collision interlock with the intake pivot ensures neither mechanism can damage the other during concurrent motion.

MECHANICAL SPECIFICATION

Parameter	Value
Motor	Kraken X44
Motor controller	Integrated TalonFX
CAN ID	9 (standard roboRIO bus)
Gear ratio	(50:12) × (80:15) = ~22.2:1

Travel range	0° to 270°
Starting / safe position	90°
Max angular velocity	360°/s
Max angular acceleration	1,440°/s²
Position tolerance	±2°
Safety tolerance	±5°
Current limit	40 A (supply)
Neutral mode	Brake

The two-stage gear reduction produces a 22.2:1 overall ratio, providing significant torque at the turret plate and allowing the Kraken X44's compact form factor to be used where a larger motor would not fit within the mechanism geometry. The Kraken X44 was selected over the X60 specifically because the reduced physical size of the X44 suited the turret's packaging constraints while still providing sufficient torque at this reduction.
The 270° travel range covers the full arc from the robot's left side, across the front, to the robot's right side — the complete hub-facing hemisphere during normal play. The turret starts at 90° (pointing forward) and is initialized to this position on boot. A separate safety tolerance of ±5° is used for interlock checks, providing a buffer beyond the 2° tracking tolerance before the collision protection engages.

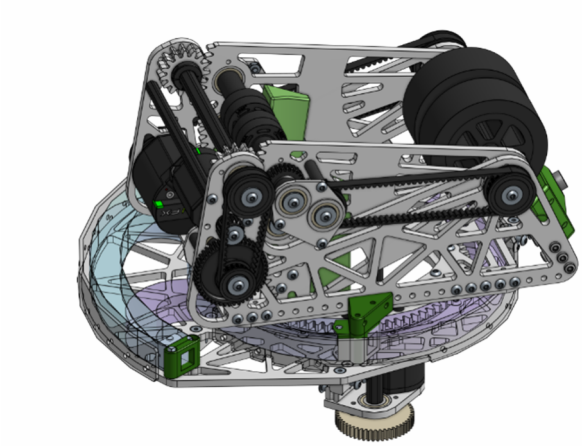
COLLISION INTERLOCK WITH INTAKE

The turret and intake pivot share a physical collision envelope during intake deployment. If both mechanisms move simultaneously without coordination, they can contact each other and cause damage to both.
The safe operating boundary is defined by the intake pivot angle. When the intake is deployed beyond **70°** from its retracted position, the turret must be at or near its **90° forward-pointing position** to clear the intake structure. When the intake is fully retracted, the turret can rotate freely through its full 270° range. These boundaries were determined from the CAD model and confirmed during mechanical integration.
For a description of how the collision avoidance is enforced in software, see Section 2.3 of the Software Section.

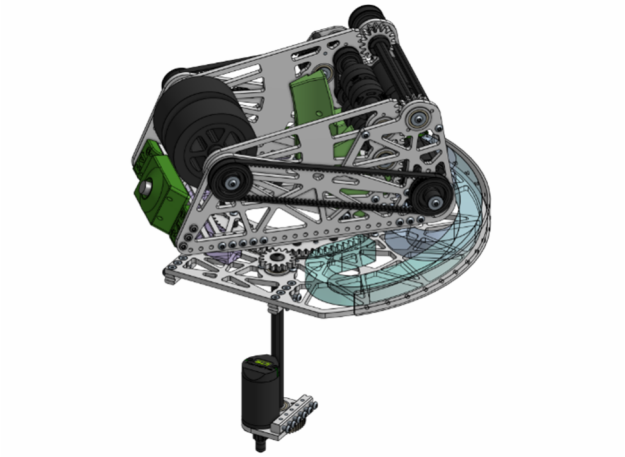
The avoidance of this collision is enforced entirely in software — each mechanism monitors the other's position every control cycle and restricts its own motion when the other is in a dangerous position. This approach means the protection is active regardless of what command is running, and neither mechanism depends on the other to stay safe. For the full description of how this is implemented, see Section 2.3 of the Software Section.

For information on how the turret tracks the hub, the MotionMagic control configuration, and velocity feedforward during movement, see Section 2.4 of the Software Section.

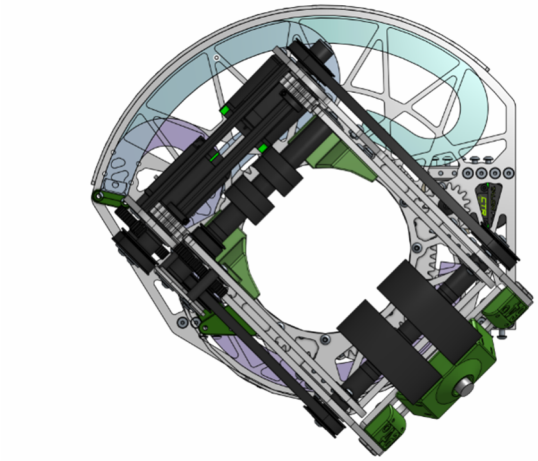
MAJOR SYSTEM # 6: SHOOTER



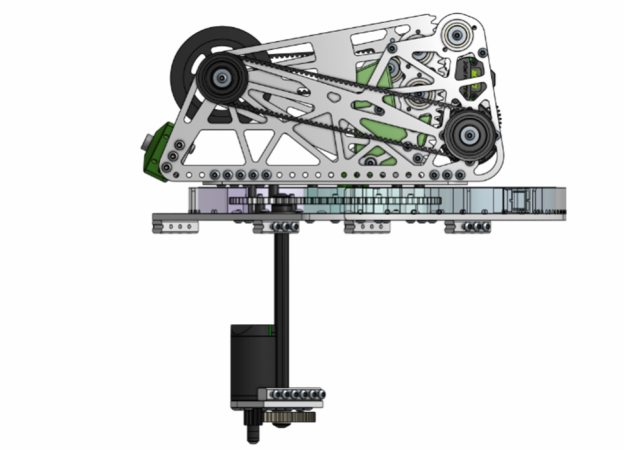
ISOMETRIC VIEW



ISOMETRIC VIEW



TOP VIEW



SIDE VIEW

The shooter system is the mechanism responsible for launching game pieces (FUEL) into the scoring HUB. It consists of two closely integrated components — the **flywheel** and the **hood** — both mounted on the turret. The flywheel imparts velocity to the game piece; the hood controls the launch angle. Together they determine where the shot lands. Neither is useful without the other.

A game piece delivered by the kicker enters the shooter from below, is gripped between the flywheel wheel and a set of upper counter-rotating rollers, and exits through the hood opening. Adjusting the hood angle changes the trajectory — lower angles produce flatter, longer-range shots; higher angles produce steeper arcs for closer distances. This variable-angle capability was a core design requirement identified in strategy analysis, allowing the robot to score accurately across the full range of distances it encounters during a match.

Turret / Intake collision interlock: Because the shooter system is mounted on the turret, it is subject to the same physical collision boundary with the intake pivot. When the intake is deployed beyond its safe range, the turret — and everything mounted on it including the shooter — is restricted to its forward-pointing position until the intake clears. This is enforced in software; see Section 2.3 of the Software Section for details.

FLYWHEEL

PHYSICAL DESCRIPTION

The flywheel is a single 4" solid urethane wheel (Thriftybot TTB-0106, 45A durometer, 1.5" wide) mounted on a horizontal shaft inside the shooter housing. It is belt-driven from the Kraken X60 motor via a 30T-to-30T HTD5 belt and pulley pair, giving a 1:1 drive ratio. A set of upper counter-rotating rollers is driven from the same flywheel shaft via a secondary belt stage, ensuring the game piece is gripped symmetrically from both above and below as it exits.

The 45A urethane durometer was selected for its balance of grip and durability. It is firm enough to transfer energy to the game piece efficiently and compliant enough to grip reliably without slippage. Unlike foam wheels, solid urethane does not develop flat spots or uneven wear patterns over extended use.

MOTOR & ELECTRICAL SPECIFICATION

Parameter	Value
Motor	Kraken X60
Motor controller	Integrated TalonFX
CAN ID	7
CAN bus	Standard roboRIO bus
Belt ratio	1:1 (30T : 30T)
Supply current limit	80 A
Neutral mode	Coast

The 80 A supply current limit is the highest on the robot, reflecting the flywheel's demand during spin-up from rest to operating speed. Coast neutral mode is used because braking a high-inertia flywheel at full speed would produce significant back-EMF stress on the motor controller.

HOOD

PHYSICAL DESCRIPTION

The hood is a pivoting deflector plate that forms the exit aperture of the shooter. It rotates about a fixed pivot point on the shooter housing, changing the angle of the exit opening across a 37° range of motion. At its minimum position (0°) the hood is nearly flat, producing the shallowest possible launch angle. At maximum (37°) it is steeply raised, producing the highest arc. The idle position is minimum — the hood rests flat when not actively shooting.

The hood pivot is driven through a three-stage gear reduction totaling 45.3:1 — (48:16) × (24:18) × (170:15). This high ratio was chosen to give the mechanism excellent holding torque and fine position resolution at the exit angle, allowing it to resist the vibration of the flywheel at speed without wandering. The operator can trim the hood angle in ±1° increments from the operator controller during a match to compensate for any real-world variation from the calibration conditions.

MOTOR & ELECTRICAL SPECIFICATION

Parameter	Value
Motor	Kraken X44
Motor controller	Integrated TalonFX
CAN ID	8
CAN bus	Standard roboRIO bus
Gear ratio	(48:16) × (24:18) × (170:15) = 45.3:1
Travel range	0° to 37°
Idle position	0° (minimum)
Position tolerance	±1°
Supply current limit	20 A

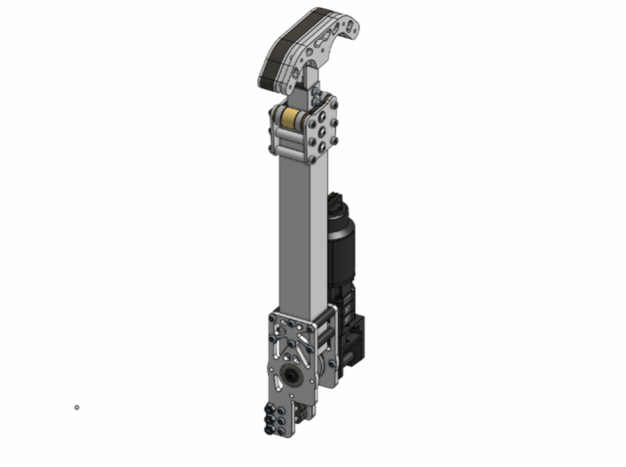
The Kraken X44 was selected over the X60 because the mechanism geometry on the turret did not have room for the larger motor. The 20 A current limit is the lowest on the robot — the hood is a low-energy positioning mechanism, not a high-speed actuator, and 20 A is more than sufficient to drive a 45.3:1 reduction through a 37° arc.

SHOOTER SYSTEM SUMMARY

	Flywheel	Hood
Motor	Kraken X60	Kraken X44
CAN ID	7	8
Ratio	1:1 (belt)	45.3:1 (gear)
Travel	Continuous rotation	0° – 37°
Current limit	80 A	20 A

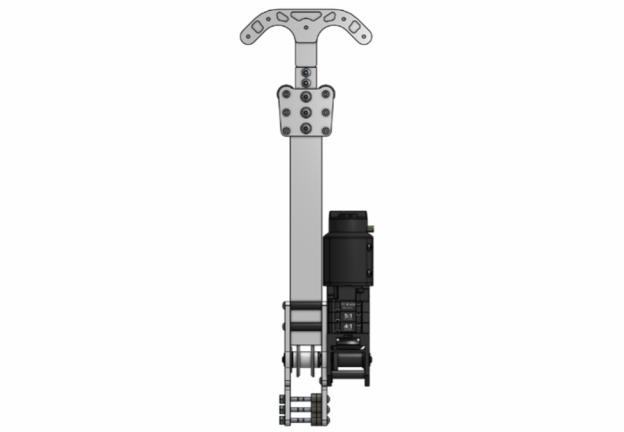
Both mechanisms are mounted on the turret and rotate with it. For information on how flywheel speed and hood angle are selected as a function of shot distance, how the readiness gate coordinates the shooter with the turret and kicker, and how the MotionMagic controllers are configured, see the Software Section.

MAJOR SYSTEM #7 CLIMBER



ISOMETRIC VIEW – CLIMBER RETRACTED

SIDE VIEW - EXTENDED



SIDE VIEW - RETRACTED

The tower climb is the endgame scoring mechanism in REBUILT. The scoring analysis identified Level 1 climb as a **required** capability — it is worth 10 points in both autonomous and teleop and contributes to the Traversal RP, which requires the alliance to accumulate 50 total tower points in a match. Level 2 (20 pts) and Level 3 (30 pts) were identified as high-value additions that, combined with alliance partner climbs, can reliably secure the Traversal RP.
The time-based analysis budgeted 5.6 seconds for a Level 1 climb from travel to locked. Given the 30-second endgame window, efficient and reliable engagement is more important than maximum climb height — a robot that consistently hits Level 1 every match contributes more to the alliance than one that attempts Level 3 unreliably.

MECHANISM OVERVIEW

The climber is a **winch-and-spool** mechanism. A spool driven by the motor winds a rope or cable, which in turn draws a telescoping arm or hook upward onto the tower rung. The position of the climber is tracked in linear units (meters) derived from the spool rotation — the software converts motor shaft rotations to meters of rope travel using the spool circumference. The climber has two defined positions: fully retracted (0 m) for transit and fully deployed (0.222 m / 8.75 in) for engagement with the tower rung. A bump command allows the operator to fine-tune position in 0.1-inch increments during the climb, and the fudge offset can intentionally exceed the normal clamped limits to ensure full engagement with the rung even if the robot's initial alignment is slightly low.
Important operational note: The motor controller's position register is zeroed to 0 at every robot startup, which means the climber **must be fully retracted** at power-on. If the robot is started with the climber partially deployed, the software will have an incorrect position reference for the entire match.

MECHANICAL SPECIFICATION

Parameter	Value

Motor	Falcon 500
Motor controller	Integrated TalonFX
CAN ID	10 (standard roboRIO bus)
Gear ratio	(5:1) × (4:1) = 20:1
Spool diameter	0.75 in
Spool circumference	~2.356 in (59.8 mm)
Max travel	8.75 in (222 mm)
Retracted position	0.0 m
Deployed position	0.222 m (8.74 in)
Position tolerance	±0.05 m
Bump increment	0.1 in (2.54 mm)
Supply current limit	40 A

GEAR RATIO & FORCE CAPABILITY

The 20:1 two-stage gear reduction was sized to lift the robot's full competition weight — including battery and bumpers — with a meaningful safety margin. At 40 A supply limit and approximately 80% mechanical efficiency, the motor produces sufficient torque at the spool to generate approximately 1,290 N (290 lbf) of pulling force. The robot's total competition weight including battery and bumpers is approximately 622 N (140 lbs), giving a **factor of safety of approximately 2.1×**. This matches the design target from the climb analysis worksheet (target factor of safety: 2.0×). The margin ensures the climber can engage and hold reliably even with minor mechanical losses, cable angle variation, or field surface variation under the tower.

SPOOL & DRIVE TRAIN

The spool diameter of 0.75 inches was selected to balance two competing factors: a larger spool winds faster but produces less force for a given motor torque; a smaller spool produces more force but winds slower. At 0.75 inches with a 20:1 reduction, the mechanism reaches full deployment in a time consistent with the strategy budget while maintaining the required force margin. The AndyMark CIB (Continuous Improvement Block) kit components are used in the climber drive train.

SOFTWARE

1 - SOFTWARE PHILOSOPHY & DESIGN PRINCIPLES

Team 1757 enters the 2026 season with a software stack built around three convictions that have guided the programming team for several years: that reliable, well-structured code is a competitive advantage that compounds over time, that every robot system must be testable without touching hardware, and that rigorous logging is not optional instrumentation but the foundation of every tuning and debugging decision made at competition. The 2026 codebase — named "Rebuilt" in both the repository and the season theme — is the result of two significant offseason projects that put those convictions into practice at a depth the team has not previously achieved.

1.1 OVERARCHING SEASON GOAL: SIMULATE AND VERIFY EVERYTHING

The programming team entered the 2026 season with a declared goal: simulate and verify as much as possible before any code reaches the physical robot. This influenced every architectural decision in the stack. The IO abstraction layer exists specifically so that simulation, real-hardware, and log-replay implementations can be swapped behind identical interfaces. PhotonVision's simulation classes are used to verify the vision system's accuracy against a modeled field before it is deployed to hardware. SysId characterization routines are run in simulation before being run on real mechanisms. The goal is not a perfect simulation — it is to ensure that code is functional and obviously correct before consuming any of the team's limited and precious allocated bot time.

This philosophy is operationally significant for a team of Westwood's size. Programming time on the physical robot is rationed against mechanical build, driver practice, and electrical work. Every hour of debugging that can be done in simulation is an hour that does not have to be done with the robot on a cart. By the time code was first deployed to Orion, every subsystem had already been exercised through hundreds of simulated loop cycles with realistic sensor feedback.

1.2 VISION SYSTEM AS A TECHNICAL PRIORITY

The 2026 season features the most ambitious vision configuration Team 1757 has built. Rather than a camera fixed to the robot chassis, the primary camera is mounted on the turret. This keeps the camera consistently pointed toward the scoring hub regardless of robot heading, but it also means the camera moves in field space as the turret rotates. Correctly estimating robot pose from a moving camera requires accounting for both the robot's pose and the turret's angle at the exact moment each image was captured.

To handle this, the team wrote a custom extended Kalman filter pose estimator — TurretedRobotPoseEstimator — that maintains timestamped buffers of both robot odometry history and turret rotation history. When a vision observation arrives with a capture timestamp, the estimator looks back in both buffers to reconstruct the field geometry at the moment of capture before applying the Kalman update. This is described in detail in Section 5.

1.3 SHOOT-ON-THE-MOVE AS A SCORING STRATEGY

A robot that must stop to shoot is a robot that can be defended. The 2026 software implements a shoot-on-the-move (SOTM) pipeline that computes a lead-angle correction for the robot's current velocity, allowing the driver to continue moving while the shooting sequence executes. This requires the turret's tracking command, the flywheel speed lookup, and the hood angle lookup to all consume an *effective* target location that accounts for the robot's velocity and the shot's flight time — rather than the raw hub position. The algorithm runs an iterative convergence loop each control cycle and is described fully in Section 4.

1.4 HUB SHIFT AWARENESS

The 2026 game introduces a hub shift mechanic: scoring targets alternate between two hubs on a fixed schedule during the teleop period. The software tracks which hub is currently active based on match time and the autonomous period result, automatically routing the turret tracking, flywheel speed, and hood angle to the correct hub target throughout the match. Three seconds before each shift boundary, both controllers produce a haptic warning sequence so the drive team can prepare. This awareness is embedded in RobotState and requires no manual input from the driver or operator during a shift.

1.5 IO ABSTRACTION PATTERN

Every hardware subsystem in the 2026 codebase follows an IO abstraction pattern that is central to the season's simulation goal. Each subsystem is implemented as three components:

- An **abstract IO base class** defining the contract between the subsystem and its hardware (e.g., TurretSubsystemIO). This specifies what inputs the subsystem can read and what outputs it can write, with no hardware dependencies.
- A **real hardware implementation** that reads from and writes to physical CTRE TalonFX motor controllers via the Phoenix 6 API (e.g., TurretSubsystemIOTalon). The TalonFX is the motor controller; the Kraken X60 or Falcon 500 is the actual motor it drives.
- A **simulation implementation** that replaces the motor controller with a WPILib physics simulation of the mechanical system downstream of the controller — modeling the rotor inertia, mechanism load, and sensor response that the real motor and gearbox would produce (e.g., TurretSubsystemIOSim).

The subsystem class itself communicates only with the abstract IO layer and is never coupled to hardware directly. The key insight is that the simulated implementation models what happens *between* the controller command and the physical outcome: the motor's acceleration, the encoder's response, and the mechanism's behavior under load. This makes function calls identical between simulation and reality — the subsystem does not know or care which implementation it is running against, and there is no isSimulation() branching inside any periodic logic.

When a motor controller is replaced or a sensor changes, only the IO implementation layer needs to be updated. The subsystem logic, command logic, and constants are completely insulated from the hardware change.

Why IO abstraction matters:
The IO abstraction is not just a software pattern — it is an operational strategy. When a TalonFX motor controller failed at DCMP in a prior season, the fix required touching exactly one file and one class. The rest of the stack was unaffected and untouched. In 2026, the same boundary protection extends to simulation fidelity: the real and sim implementations have been verified to produce matching outputs under identical inputs.

2 - CODE ARCHITECTURE & PROJECT ORGANIZATION

The repository is organized into five primary directories under src/, each with a clear and exclusive responsibility. This separation is enforced through code review and pylint, which flags cross-boundary imports that violate the intended structure.

Directory / File	Responsibility	Key Contents
robot.py	Entry point; logging bootstrap	MentorBot class, mode routing (REAL / SIM / REPLAY), command scheduler callbacks
robotcontainer.py	Subsystem wiring & button bindings	Subsystem instantiation by mode, auto chooser, OI config, override bindings
robotstate.py	Global game-state singleton	Dual pose estimators, SOTM logic, hub shift schedule, objective routing
subsystems/	One folder per subsystem	IO base + Talon + Sim implementations; periodic logic and SysId routines
commands/	All user-visible robot behaviors	Shooting, intake, climb, drive, override, and autonomous commands
constants/	All tunable parameters & mappings	Drive geometry, shooting lookup tables, field geometry, vision thresholds
util/	Shared infrastructure	LogTracer, LoggedTunableNumber, RobotPoseEstimator, flip utilities, joystick wrappers

2.1 SUBSYSTEMS ON THE 2026 ROBOT

The 2026 robot (Orion) is the most mechanically complex robot Team 1757 has fielded. The software manages eight concurrent subsystems, each with independent IO implementations and command sets:

- **Drive** — Four-module SDS MK5i swerve drive, Pigeon 2.0 gyro, PathPlanner-integrated autonomous
- **Turret** — Rotating upper structure carrying the camera and shooter assembly; includes collision interlock with intake
- **Flywheel** — One TalonFX motor controller driving one Kraken X60 motor, with speed-ramping and idle profiles
- **Hood** — Adjustable angle mechanism for shot trajectory control
- **Indexer** — Ball staging and feeding mechanism between intake and flywheel
- **Intake** — Deployable ground intake with pivot and roller subsystem; includes collision interlock with turret
- **Climber** — Endgame climb mechanism with deploy and retract stages

- **Vision** — PhotonVision camera pipeline running on a dedicated coprocessor, turreted configuration
- **LEDs** — Addressable LED feedback strip for driver and operator state visibility

The subsystems are instantiated in RobotContainer using a Python match/case statement on the robot mode enum (RobotModes.REAL, SIMULATION, or REPLAY). This means the entire hardware stack is swapped in a single conditional block, with zero runtime branching inside any individual subsystem. Log replay is a full first-class operating mode where all subsystems run against inputs sourced from a previous match's log file rather than live hardware.

2.2 COMMAND-BASED CONTROL FLOW

The robot follows WPILib's command-based paradigm throughout. Subsystems declare their default commands in RobotContainer: the turret tracks the target by default, the indexer holds position, and the hood moves to its minimum angle when idle. Commands are composed declaratively using the cmd module — parallel(), sequence(), waitUntil(), and andThen() — rather than imperatively managing state inside loops. This keeps individual command files short, readable, and independently testable.

Operator intent is expressed through two controllers: an Xbox controller for the driver and a custom farm box controller for the operator. The driver manages all drive commands plus the shooting trigger (right trigger) and intake deployment (left bumper). The operator box handles all climb operations, intake modes, indexer overrides, and the shooting objective toggle (SHOOT vs. FEED mode). Override commands for flywheel, hood, and turret are exposed via NetworkTable buttons, allowing dashboard triggering at competition without consuming controller inputs.

2.3 TURRET / INTAKE COLLISION INTERLOCK

One of the most safety-critical software problems on Orion is the three-dimensional collision envelope between the rotating turret and the deploying intake pivot. Both mechanisms can occupy the same physical space during transition, and a collision between them would damage both subsystems. The team solved this at the software layer with a bidirectional interlock that operates entirely within each subsystem's own periodic loop — independent of any higher-level robot state or command.

The turret subsystem checks the current intake pivot angle on every periodic cycle. When the intake is beyond kPivotDangerZoneStart, the turret clamps its goal angle to kTurretStartingAngle — a known safe position clear of the intake's sweep arc — regardless of whatever goal was commanded by any shooting or auto routine. The intake subsystem performs the complementary check: if the turret is detected to be outside its safe zone, the intake clamps its pivot movement and waits. Once the turret has moved to a safe position, the intake executes its full desired rotation in a single motion.

Safety design: These interlocks are not mediated by RobotState, command scheduling, or any external coordinator. Each mechanism is solely and directly responsible for keeping itself safe. This ensures that if a higher-level system produces an incorrect goal — due to a software bug, a communication error, or a failed command — the mechanism-level check remains active and effective. It is a defense-in-depth approach to collision avoidance.

To support verification and tuning of the interlock during development, AdvantageScope's custom robot model feature was used with a 3D model of Orion exported directly from CAD. This made it possible to visualize the turret rotation and intake pivot position simultaneously in simulated space, catching geometric assumptions that were wrong before they could cause hardware damage on the physical robot.

Lesson Learned
Implementing the collision interlock as a mechanism-level property rather than a command-level property meant it survived a midseason refactor of the shooting command structure without any changes. The protection is invisible to the commands layer — it just works, regardless of what the command asks for.

2.4 TURRET CONTROL ARCHITECTURE

The turret uses **MotionMagic position control** via the Phoenix 6 API — a motion-profiled position controller that follows an exponential velocity profile to reach the target angle smoothly. The profile is parameterized by maximum velocity (360°/s) and maximum acceleration (1,440°/s²). Feedforward gains (kS, kV, kA) derived from SysId characterization provide accurate open-loop voltage prediction, with a PD controller correcting residual error.

Software limit switches are configured in the TalonFX firmware at both ends of the 270° travel range. These are enforced at the controller level — not just in the subsystem code — so the turret cannot be driven past its physical limits even if a software error produces an out-of-range goal.

Velocity feedforward during tracking — the default turret tracking command (trackedTurretMoving) computes not only the target angle but also the instantaneous rate of change of that angle based on the robot's current velocity and the turret's

kinematic offset from the robot center. This angular velocity feedforward is passed to the MotionMagic controller each cycle, reducing tracking lag during high-speed robot motion and improving shot geometry during shoot-on-the-move sequences. The turret's physical location offset from the robot center (kTurretLocation: -0.102 m, 0.178 m, 0.368 m) is used both in this calculation and throughout the pose estimation pipeline to correctly model the camera's field position as the turret rotates.

2.5 INDEXER COMMAND STRUCTURE

The spindexer and kicker are commanded as a single IndexerSubsystemGoal with three states: HOLD (both mechanisms at rest, coast), KICK (both active simultaneously), and EJECT (both reversed). The spindexer's two motors are configured in an **aligned leader/follower** arrangement — motor 3 is the leader, motor 4 mirrors it exactly in the same direction — so the subsystem sends a single voltage command and both motors respond. The kicker's two motors are configured in an **opposed leader/follower** arrangement — motor 6 is the leader, motor 5 applies the inverse voltage — automatically producing counter-rotation at the rollers from a single command.

Both mechanisms run in open-loop voltage control at ±3.0 V. There is no position or velocity feedback in the indexer — it is a purely voltage-driven system, which keeps the command logic simple and eliminates the risk of closed-loop instability in a mechanism that only needs to spin forward, stop, or reverse.

The indexer is held at HOLD by its default command and only transitions to KICK when the shooting command's readyToShoot() gate opens — requiring the flywheel, hood, and turret to all simultaneously reach their target states. This gate is the only condition under which the indexer feeds a game piece.

2.6 INTAKE CONTROL ARCHITECTURE

The pivot uses **MotionMagic position control with ARM_COSINE gravity compensation** — a gravity feedforward term that varies with the cosine of the arm's current angle, correctly modeling the varying torque demand as the arm moves through its arc. Without gravity compensation, the controller would need a large P gain to overcome gravity at the worst-case angle, which would cause overshoot at other positions. The ARM_COSINE model allows a smaller P gain to be used throughout the range, producing smoother and more accurate motion. Feedforward gains (kS, kV, kA, kG) were derived from SysId characterization of the arm under load.

The pivot supports a live **fudge offset** of ±5° incremented from the operator controller (buttons 15 and 16), allowing the drive team to trim the deployed ground contact angle during a match without a code change.
The roller runs in simple open-loop voltage control at ±6.0 V. No position or velocity feedback is needed — the roller either runs forward (intaking), runs reverse (ejecting), or is neutral.

Oscillate mode is a special pivot behavior used during the shooting sequence. When active, the pivot cycles between 0° (fully deployed) and 15° (slight upward tilt) while the rollers run, gently agitating game pieces at the intake/spindexer interface to promote consistent delivery. The oscillation reverses direction each time the arm reaches its target, producing a slow continuous rocking motion.

The intake's side of the turret/intake collision interlock checks RobotState.turretRotation every periodic cycle. When the turret is outside its safe zone (beyond ±5° of 90°) and the commanded pivot goal would take the arm below 100°, the pivot is held at the 100° safe position regardless of the commanded goal. Once the turret returns to safe, the intake resumes its full motion to the original target in a single movement.

3 - DRIVE SYSTEM & POSE ESTIMATION

3.1 SWERVE DRIVE IMPLEMENTATION

The drive system uses four SDS MK5i swerve modules. Each module is driven by a Kraken X60 motor through a TalonFX motor controller for the drive wheel, and a Falcon 500 through a TalonFX motor controller for the steering mechanism. Absolute steering position is read from a CANCoder on each module. All four modules communicate over a dedicated CANivore CAN bus, providing higher bandwidth and more deterministic timing than the standard roboRIO CAN loop.
The DriveSubsystem implements field-relative and robot-relative drive modes through a CoordinateMode enum, and applies ChassisSpeeds.discretize() to correct for translational drift during rotation — a standard correction for high-speed holonomic drivetrains that many teams omit. Wheel speed desaturation ensures commanded speeds are always achievable across all four modules simultaneously. A defense state locks all four modules in an X-pattern using the diagonal geometry of each wheel's position as its locked angle.

If the gyro disconnects, the subsystem falls back to a twist-based heading estimate derived from swerve module position deltas — graceful degradation that allows the robot to continue functioning at reduced accuracy rather than failing completely.

3.2 DUAL POSE ESTIMATOR ARCHITECTURE

The 2026 codebase maintains two independent pose estimators simultaneously: a **field estimator** and a **hub estimator**. Both run the same underlying TurretedRobotPoseEstimator algorithm but are fed different subsets of vision observations, filtered by which AprilTags are visible in each reading.

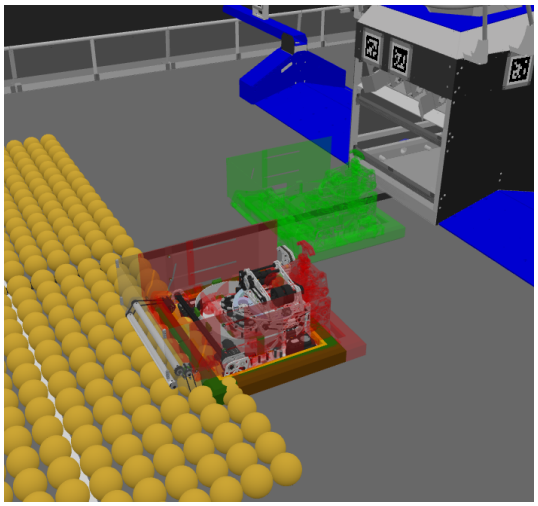
Estimator	Tag Filter	Primary Use	Update Sources
fieldEstimator	All valid field tags	Path following, feed objective, field display	Odometry + all accepted vision
hubEstimator	Alliance hub tags only	Shot distance, turret angle, hood angle	Odometry + hub-tag vision only
odometry	None (no vision)	Baseline reference, replay analysis	Wheel encoders + Pigeon 2.0

The separation means that drift in the field estimate — due to playing-field variation, floor slippage, or off-hub tag detections — does not corrupt the shooting pipeline. All shot distance calculations, turret angle targets, and hood angle targets are derived exclusively from the hub estimator.

3.3 TURRETEDROBOTPOSEESTIMATOR & LATENCY COMPENSATION

The team implemented a custom pose estimator class, TurretedRobotPoseEstimator, that applies an extended Kalman filter to fuse wheel odometry with vision observations from a camera mounted on a moving turret. This presents a challenge that a standard pose estimator does not handle: the camera's position in field space changes as the turret rotates, so the transform from camera observation to robot pose must account for the turret angle at the exact moment the image was captured, not the turret angle when the observation arrives at the estimator.

The estimator solves this with two timestamped interpolation buffers running in parallel. A TimeInterpolatablePose2dBuffer holds a 2-second rolling history of robot odometry poses. A TimeInterpolatableRotation2dBuffer holds a 2-second rolling history of turret angles. When a turreted vision observation arrives with a capture timestamp, the estimator samples both buffers at that timestamp, reconstructing the exact robot pose and turret rotation at the moment of capture. It then inverts the turret transform, derives the robot-body pose implied by the observation, and feeds that into the Kalman update with standard deviation weights that scale with observation distance and tag count. This correctly compensates for the 50–100ms latency between image capture and processing without accumulating systematic heading error.



Architecture Note

The base RobotPoseEstimator — the core Kalman filter structure fusing wheel odometry with vision observations — is a Python reimplementation of Team 6328's pose estimator, acknowledged in the code with the comment *"shamelessly reimplemented from 6328."* The TurretedRobotPoseEstimator extension — the timestamped turret rotation buffer, the latency-compensated turret transform inversion, and the turreted vision update math — is original team work, written specifically for this robot's nonstandard camera configuration.

3.4 AUTONOMOUS PATH FOLLOWING

Autonomous routines are built using PathPlanner and loaded dynamically from the deploy directory. The AutoBuilder is configured with a PPHolonomicDriveController using tuned translation and rotation PID gains specific to the autonomous period. Named commands (shoot, deployIntake, Nothing) are registered with NamedCommands so they can be embedded in PathPlanner auto choreographies without code changes. The auto chooser uses a LoggedDashboardChooser, so every selection made before a match is recorded in the log file for post-match review.

4 - SHOOTING PIPELINE & GAME-STATE MACHINE

4.1 SUBSYSTEM STATE COORDINATION

Scoring in the 2026 REBUILT game requires three mechanisms to be simultaneously ready: the flywheel must be at target speed, the hood must be at target angle, and the turret must be pointed at the correct field position. The shooting commands coordinate all three via WPILib's parallel composition and a readyToShoot() predicate on RobotState that checks all three flags simultaneously. The indexer command is blocked behind a waitUntil(RobotState.readyToShoot) gate and only fires once all conditions are met. This is a deliberate design choice for 2026 — coordinating all three systems before feeding ensures every shot that releases a game piece is a shot with correct geometry.

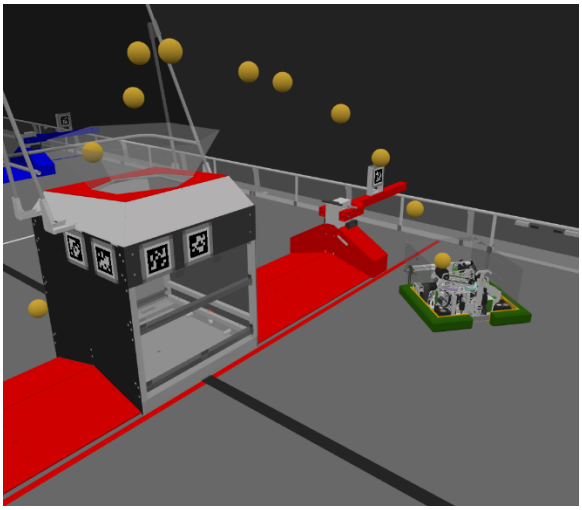
4.2 DISTANCE LOOKUP TABLES

Rather than computing flywheel speed and hood angle analytically from a physics model, the 2026 software uses piecewise-linear interpolation tables sampled at ten distance breakpoints from 1.24 meters to 6.0 meters. The values in these tables are a hybrid of physical testing and mathematical modeling. The team measured real shot data across a range of distances, built a physics-informed model in Desmos (shot map: desmos.com/calculator/jg3zj1xo8p, feed map: desmos.com/calculator/wf2vba0h0n) and tuned the model parameters until they matched the measured data. The outer distance breakpoints — 4 m, 5 m, and 6 m — were not physically tested but were instead computed from the validated model. This approach is intentionally pragmatic: it extends the tested range to distances the team could not easily measure while grounding the extrapolation in a model that has been matched to reality at closer distances.

Distance (m)	Flywheel (rad/s)	Hood Angle (°)	Shot Time (s)	Feed Flywheel (rad/s)
1.24	200	12.0	0.780	110
1.71	230	15.0	0.787	135
2.11	230	20.0	0.792	150
2.97	250	23.0	0.911	185
4.00	280	25.0	1.020	220
6.00	320	35.0	1.050	274

Two separate interpolation tables exist for SHOOT and FEED objectives. The FEED table maps to lower flywheel speeds and a fully open hood, targeting a drop zone rather than the hub. The active table is selected at runtime by the RobotState objective enum, toggled by dedicated operator buttons.

4.3 SHOOT-ON-THE-MOVE (SOTM) COMPENSATION



SCREEN SHOT OF SHOOTING ON THE MOVE WORKING IN ROBOT SIMULATION

Stationary shooting is straightforward — aim at the hub and fire. Shooting while the robot is moving requires predicting where the hub will be relative to the robot when the game piece actually arrives. The 2026 pipeline implements this using an iterative convergence algorithm with five iterations per control loop cycle.
In each iteration, the algorithm computes the shot time for the current effective distance using the shot-time interpolation table, then back-calculates the target's apparent position at the moment of arrival by subtracting the turret's velocity vector — including both robot translation and the contribution from robot rotation acting on the turret's offset from the robot center — scaled by shot time. The effective objective location is updated for the next iteration. After five iterations, the result converges to a stable effective objective position without requiring any direct feedforward to the turret motor controller.

Design note: Five iterations were chosen as a practical balance between computational cost and accuracy. The algorithm runs entirely within the 20 ms loop budget. The number of iterations was not derived analytically — it was set conservatively to ensure the estimate has stabilized before being used to compute turret angle and flywheel speed targets.

4.4 HUB SHIFT GAME STATE MACHINE

REBUILT introduces a hub shift mechanic: scoring targets alternate activity on a fixed schedule during the teleop period. RobotState implements full awareness of this mechanic, tracking which hub is active at any given match time based on the autonomous period winner, read from the DriverStation game-specific message ("R" or "B").

Three seconds before each shift boundary, a hubAboutToChangeTrigger fires a haptic rumble sequence on both controllers: three quick pulses followed by one long pulse. The robot will not attempt to score on an inactive hub, and the effective objective location automatically switches to the correct hub address when a shift occurs — no driver input required.

Match Time Remaining	Hub State	Active Hub Determined By
111–140s (teleop start)	All hubs active (transition)	N/A
86–110s (Shift 1)	One hub active	Opposite of autonomous winner
61–85s (Shift 2)	One hub active	Autonomous winner
36–60s (Shift 3)	One hub active	Opposite of autonomous winner
21–35s (Shift 4)	One hub active	Autonomous winner
0–20s (Endgame)	All hubs active	N/A — endgame scoring window

5 - VISION SYSTEM

The 2026 vision system is the most significant offseason engineering project on the software side. The entire vision subsystem was rewritten to support a nonstandard configuration: a single camera mounted on the turret rather than rigidly attached to the robot chassis. Understanding why the team arrived at this configuration requires a brief look at what came before it.

5.1 FOUR SEASONS OF VISION ITERATION

2022 — No vision; dead reckoning and driver control: The 2022 robot carried no computer vision pipeline. Robot positioning was handled entirely through wheel odometry dead reckoning and driver skill. The software stack that season was award-recognized for other qualities, but vision was not part of it.

2023 — Limelight 2, fixed mount: The team's first competition vision system was a single Limelight 2 camera mounted to the robot chassis, running the Limelight firmware pipeline. It was stable and consistent, but its field of view was fixed to whatever direction the robot was facing — the driver had to orient the chassis toward the target before vision could contribute useful pose information.

2024 — Multi-camera OrangePi system: Recognizing that a single fixed camera was the limiting factor, the team made an ambitious jump: multiple USB 2.0 OV9281 monochromatic global-shutter cameras mounted in the corners of the robot frame, each feeding into OrangePi coprocessors running custom PhotonVision pipelines. The intent was to provide 360-degree field coverage and a much richer pose estimate from simultaneous multi-angle tag observations. The concept was sound, but the honest reason it never reached competition reliability was scope: the system was simply too large and complex for the effective programming hours the team could dedicate to it that season. Latency inconsistencies across the OrangePis and calibration drift between events surfaced as the visible symptoms, but the root cause was that a fully multi-camera fused pipeline demanded more development and testing time than was available. The team competed that season with reduced confidence in vision output.

2025 — Dual Limelight 3s, fixed mounts: After the 2024 experience, the team stepped back to two high-quality fixed cameras — a pair of Limelight 3s fused with swerve odometry through an extended Kalman filter — using the same underlying PhotonVision pipeline architecture from 2024 but with a far more manageable hardware footprint. The system worked reliably, though the team placed less overall reliance on vision output than in prior seasons while confidence was rebuilt. It restored trust in the pipeline approach, but returned the coverage limitation: two cameras fixed relative to the chassis, still dependent on robot heading for consistent hub-tag visibility.

2026 — ThriftyCam on a turret: The 2026 solution synthesizes the lessons from all three prior approaches. Rather than adding cameras to cover more angles, the team moved the camera — a Thriftybot ThriftyCam (TTB-0241) — onto the turret, which actively tracks the scoring hub throughout the match. PhotonVision is the software pipeline running on the dedicated coprocessor, selected in part because its simulation API integrates directly with the turreted pose estimation pipeline and was essential to the 2026 simulation-first development goal. The camera now maintains consistent line-of-sight to the hub tags regardless of robot heading — achieving the coverage goal of the 2024 system with the mechanical reliability of the 2022–2023 and 2025 approach.

5.2 TURRETED CAMERA ARCHITECTURE

The camera — designated TURRET_CAM in the codebase — is mounted on the turret and moves in field space as the turret rotates. Unlike a chassis-mounted camera whose field-to-camera transform is constant, the TURRET_CAM's transform from the field frame depends on both the robot's pose and the turret's current angle. The turret-to-camera transform (kTurretToCameraTransform) is a fixed, calibrated constant describing the rigid geometric relationship between the turret rotation axis and the camera's optical center.

The advantage of a turreted camera is consistent AprilTag visibility regardless of robot heading. Because the turret's default command actively tracks the scoring hub, the camera faces the hub tags throughout the match — providing high-quality, low-latency hub-tag observations even when the robot is driving laterally or away from the hub. This directly improves hub estimator accuracy during the high-speed movements of SOTM shooting.

5.3 EXTENDED KALMAN FILTER FUSION

Vision observations are fed into the TurretedRobotPoseEstimator via an extended Kalman filter update derived from the continuous-time Kalman formulation with A = 0 and C = I. Standard deviations for each observation are computed as a function of the squared average tag distance divided by the number of tags used — observations from farther away or using fewer tags are trusted less by the filter.

The Kalman gain calculation explicitly solves the closed-form update from the odometry standard deviations and the vision observation standard deviations. When the odometry covariance is zero, the gain for that axis is also zero, meaning a vision observation has no effect on that axis. This allows individual axes to be selectively trusted — for instance, a highly reliable gyro can hold heading authority while vision still contributes translational corrections.

5.4 OBSERVATION FILTERING & ACCEPTANCE CRITERIA

Not every AprilTag detection is trustworthy enough to feed into the pose estimator. The VisionSubsystem applies a multi-criteria rejection filter to each observation before passing it to the Kalman update:

- Tag count of zero is always rejected

- Single-tag observations with ambiguity above kMaxVisionAmbiguity are rejected — high ambiguity indicates the solver cannot distinguish between two geometrically symmetric pose solutions
- Observations with Z-axis error exceeding kMaxVisionZError are rejected — a robot on the floor should have a near-zero Z component in its pose estimate
- Observations with X or Y coordinates outside the field boundary are rejected as physically impossible

All accepted and rejected observations are logged to separate AdvantageScope topics (RobotPosesAccepted, RobotPosesRejected, TurretedTransformsAccepted, TurretedTransformsRejected), enabling post-match analysis of vision health. Observation routing to the hub estimator is further filtered: an observation is sent to the hub estimator only if every tag used in the observation is a member of the alliance-specific hub tag set. Detections of perimeter tags, trench tags, or other non-hub field elements contribute to the field estimator but are completely excluded from the hub estimator.

5.5 SIMULATION VERIFICATION WITH PHOTONVISION SIM

A core motivation for the vision rewrite was the ability to verify the system's accuracy before any real-robot deployment. The VisionSubsystemIOPhotonSim class uses PhotonVision's built-in simulation API to model the camera's field of view and tag detection behavior given the simulated robot pose and turret angle. In simulation mode, RobotState provides the simulated turret pose (getSimTurretPose) which includes the simulated robot position plus the turret kinematic chain, and this pose is fed to the PhotonVision simulator to generate synthetic observations.

These synthetic observations pass through the same VisionSubsystem filtering and Kalman update pipeline as real observations would in a match. This makes it possible to verify that the pose estimator converges correctly, that the hub estimator tracks through simulated driving maneuvers, and that the SOTM lead-angle calculation is geometrically correct — all without requiring a physical field or any hardware.

Lesson learned: Simulation testing of the vision pipeline before hardware deployment revealed that the initial turret-to-camera transform calibration had a sign error in the Y component of the translation. The error would have produced a consistent 8 cm lateral offset in hub-relative pose estimates. Catching it in simulation cost 10 minutes. Catching it at a competition would have cost a match.

6 - DEVELOPER TOOLING & ENGINEERING PROCESS

6.1 PYKIT: DETERMINISTIC LOG REPLAY FOR PYTHON

PyKit is the team's second major offseason engineering project and the foundation of the entire 2026 logging and testing workflow. It brings the features of the AdvantageKit deterministic replay system — developed by FRC Team 6328 — into a Python environment compatible with RobotPy and the team's existing codebase.

The core purpose of PyKit is straightforward: every input to the robot system — motor encoder positions, sensor readings, controller inputs, camera observations, game-specific messages — is logged through a single unified Logger singleton to a WPILOG file on a USB drive at 50 Hz. When a bug is observed on the field, the log file can be brought back to a development machine and the code can be re-run against those exact inputs in REPLAY mode, producing new expected outputs from the modified code. Because every input is deterministic, every output is reproducible. The programmer sees exactly what the robot saw, and can verify exactly what the new code would have done.

This is the difference between *"we think we fixed it"* and *"we can prove we fixed it."* PyKit was built with two specific motivations in mind: first, to be able to verify whether changes to the vision pipeline will have their intended effect by replaying real match logs through the updated code — without needing field time to test; and second, to identify where the Python interpreter is consuming excessive loop time and determine where to optimize. Both goals address the practical constraints of a small school team where field time is limited and Python's performance characteristics are a genuine engineering consideration.

The three operating modes implemented by PyKit map directly to the three IO implementation types in every subsystem:

Mode	Input Source	Output Destination	Primary Use
REAL	Physical hardware via Talon IO	USB drive + NetworkTables (live)	Competition and practice matches
SIMULATION	Simulated hardware via Sim IO	WPILOG file + NetworkTables	Pre-deployment verification, tuning
REPLAY	Previous .wpilog file	New *_sim.wpilog file	Post-match debugging, fix verification

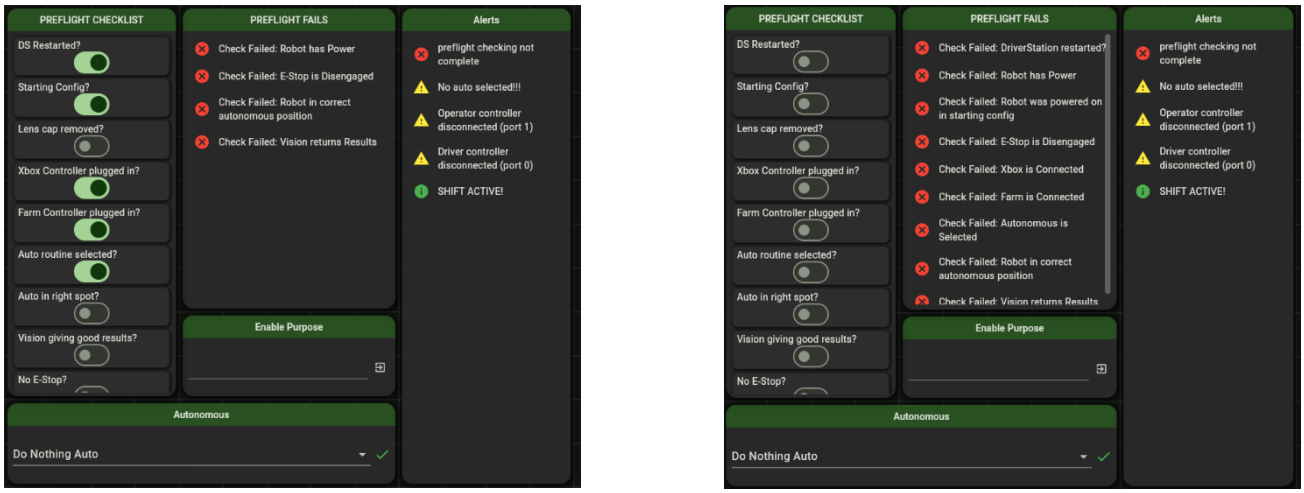
6.2 LOGTRACER: LOOP-LEVEL PERFORMANCE MONITORING

The LogTracer utility provides microsecond-level timing breakdowns of every periodic loop. It instruments every major operation within a subsystem periodic cycle and logs the elapsed time to AdvantageScope under a LogTracer/<Prefix>/<Action>MS topic. This makes it possible to identify which subsystem or operation is consuming excessive loop time — a critical diagnostic for Python-based FRC code, where the interpreter overhead makes loop overruns more likely than in Java or C++. Every release build is profiled against LogTracer data before competition to verify that the full periodic cycle completes within 20 ms under worst-case conditions.

6.3 LOGGEDTUNABLENUMBER & LIVE GAIN EDITING

PID and feedforward gains are managed through the LoggedTunableNumber infrastructure, which links each constant to a NetworkTables entry under the /Tuning/ namespace. When the robot is running in tuning mode (kTuningMode = True), dashboard changes to any tunable number propagate immediately to the running code via AutoUpdateGroup callbacks. When tuning mode is disabled for competition, all tunable numbers return their hardcoded default values with zero NetworkTables overhead.
Status note: The LoggedTunableNumber system was merged late in the 2026 build season and is not yet used for any active gain in the current codebase. It is infrastructure in place for future seasons — the team anticipates it becoming a meaningful tuning tool in 2027 once it can be integrated from the start of the build cycle.

6.4 PREFLIGHT CHECKLIST SYSTEM



SCREENSHOTS OF THE PREFLIGHT CHECKLIST FROM THE DS

The PreflightChecklist class implements a structured pre-match verification protocol. Nine checks are defined, each backed by a LoggedNetworkBoolean that the drive team manually confirms on a dashboard widget before each match. The checks cover driver station restart, robot power state, starting configuration, emergency stop status, controller connectivity, autonomous selection, starting position accuracy, and vision health. The entire checklist state is logged, so post-match analysis can confirm whether all checks were complete before a match that produced unexpected behavior.
If any check is not confirmed, a persistent Alert fires on the dashboard and a preflightAlert remains active throughout the match. The robot enables normally — the checklist is advisory, not a hard interlock — but the active alert is impossible to miss. In previous seasons, vision configuration mistakes and controller disconnects were discovered only after autonomous had already failed. The preflight system surfaces these conditions with enough time to correct them.

6.5 SYSTEM IDENTIFICATION (SYSID) ROUTINES

Every actuated subsystem — turret, climber, intake, flywheel, and hood — exposes a sysIdRoutine() method that returns a WPILib SysIdRoutine command. These routines run quasistatic and dynamic characterization sweeps against the mechanism's travel limits, automatically logging all data in the format expected by WPILib's SysId analysis tool. SysId routines are exposed in the autonomous chooser rather than as button bindings, so they can be run in a controlled environment without risk of accidental triggering during a match. Running these routines against a freshly-assembled mechanism provides the feedforward constants (kS, kV, kA) needed to initialize PID controllers with physically grounded starting gains rather than tuning from zero.

6.6 CODE QUALITY & COLLABORATION INFRASTRUCTURE

The repository enforces code quality through a three-gate commit check: Black for formatting, pylint for static analysis, and mypy for type checking. GitHub Actions runs all three gates on every pull request and code cannot be merged until it passes. pyfrc simulation tests verify that the robot initializes and runs through all mode transitions without exception. The Renovate bot monitors all Python dependencies and opens automated pull requests when new versions of robotpy, pathplannerlib, or phoenix6 are released. The uv package manager with a committed uv.lock file ensures that every developer's environment is byte-for-byte identical, eliminating the class of bugs caused by dependency version mismatches between contributors' machines.

Open Alliance: The full source code is published publicly on GitHub under an MIT license, including the committed AdvantageScope layout, CAN device configuration table, and all PathPlanner auto files. Consistent with the team's Open Alliance participation across all prior seasons, teams building similar architectures are encouraged to fork and adapt.

7 - LED SYSTEM & DRIVER COMMUNICATION

7.1 A DESIGN PHILOSOPHY SINCE 2022

Team 1757 has used addressable LED strips as a structured driver communication tool continuously since the 2022 season, and the system has grown meaningfully more sophisticated each year. The core philosophy has remained constant: the robot should be able to communicate its internal state to the people operating it without requiring them to look at a laptop screen. What has evolved is the depth of that communication and the audience it targets.

In the early seasons of the program, LEDs served a single purpose — telling the primary driver whether the robot was ready to act on a command. A driver watching a fast-moving robot across a field cannot check a dashboard, and every glance away from the robot is a risk. A color change visible from across the arena closes that loop without splitting attention. By 2024 and 2025, the team expanded this thinking to include the human player station: game rules in those seasons required precise coordination between the robot and the human player at the field perimeter, and the LEDs became a channel for signaling intake readiness and pickup requests across the noise of a competition venue floor. The specific colors and triggers changed year to year to match each game's communication needs, but the underlying design axiom — that a robot should actively telegraph its state to its operators — has been present in every season's codebase since the framework was first built.

The 2026 implementation is the most complete expression of this design yet. It integrates LED behavior directly with game-specific state (RobotState.hubActive(), RobotState.hubAboutToChange(), and RobotState.readyToShoot()), producing a system that communicates not just mechanism readiness but the robot's real-time understanding of the game clock and field conditions.

7.2 HARDWARE

The 2026 LED system is built around a CTRE CANdle controller (CAN ID 21, on the CANivore bus), which drives both its eight onboard LEDs and an external strip of 18 RGBW LEDs mounted around the robot frame — 26 total addressable elements. The use of RGBW rather than standard RGB strips is deliberate: the dedicated white channel produces a true, neutral white that standard RGB approximations cannot match, and allows colors like yellow and orange to be rendered with greater saturation and brightness visibility under arena lighting. The CANdle is addressed through the Phoenix 6 API using the same SolidColor, StrobeAnimation, and RainbowAnimation control classes used across the rest of the CTRE motor controller stack, keeping the LED system integrated into the same logging and control infrastructure as every other actuator on the robot.

7.3 STATE MACHINE & COLOR LOGIC

The LED subsystem's periodic() loop implements a priority-ordered state machine that evaluates DriverStation and RobotState conditions on every 20 ms cycle and commands the appropriate animation. States are evaluated in strict priority order: e-stop overrides all other conditions, disabled conditions are checked next, and enabled conditions are evaluated last. Within each enabled condition, hub-shift awareness takes priority over readiness state, producing the combined behavior of color (ready/not ready) and animation style (solid/flashing) simultaneously.

Robot State	LED Behavior	Color	Animation
E-stopped	Strobe	Red	Fast strobe — immediate safety signal
Disabled, brownout detected	Strobe	Orange	Fast strobe — alerts pit crew to voltage drop
Disabled, normal	Rainbow	All	Continuous sweep — "at rest" / idle indicator

Disabled, first 5s after auto	Fade-out	Blue	Solid blue, fading — auto duration indicator
Enabled, not ready, hub stable	Solid	Yellow	Steady — shooter aligning or spooling
Enabled, ready to shoot, hub stable	Solid	Green	Steady — cleared to fire
Enabled, not ready, hub shift imminent	Strobe	Yellow	Fast strobe — aiming, shift warning active
Enabled, ready to shoot, shift imminent	Strobe	Green	Fast strobe — locked on, shift warning active

The combination of color and animation carries two independent channels of information simultaneously. The driver reads **color** to know whether the robot is ready to fire, and reads **animation style** to know whether a hub shift is approaching. A flashing green strip means the robot is locked on *and* the active hub is about to change — the highest-urgency state in the match, where the drive team has roughly three seconds to fire before the target becomes inactive. This mirrors the haptic rumble sequence described in Section 4.4 and gives the driver a second, spatial channel for the same alert.

7.4 THE POST-AUTONOMOUS BLUE FADE

One of the less immediately obvious behaviors in the LED state machine is the post-autonomous blue fade. When the robot transitions from autonomous to disabled — which occurs at the end of autonomous period before the teleop enable — the LEDs display solid blue and then fade to off over a five-second window. The brightness of the remaining strip at any moment during the fade is proportional to the remaining fade time. This is implemented by calculating a percent value from elapsed time and using a SolidColor command to light only int(percent × 18) of the external LEDs, leaving the remainder dark.

The practical effect is that the pit crew and coaches watching from the alliance station can see at a glance how long the robot was moving during autonomous — a fully-lit blue strip indicates the robot was active until the very end of auto, while a partially-faded or already-dark strip suggests the autonomous routine completed early or the robot stopped prematurely. No dashboard readout is needed to make this assessment during the 3-second gap between autonomous end and teleop enable, when decisions about whether to report a hardware issue or proceed normally are being made in real time.

7.5 INTEGRATION WITH THE DRIVER INTERFACE

The LED system is the outward-facing half of a two-layer driver feedback design. The primary driver watches the robot and reads the LEDs. The secondary (operator) driver watches the Elastic dashboard, which provides detailed sensor telemetry — flywheel speed, hood angle, turret position, vision confidence, hub estimator state, and the preflight checklist — and communicates relevant information verbally to the primary driver.

This division of labor is intentional. The primary driver's attention is on the field: robot position, defense, game piece locations, and alliance partner coordination. The operator's attention is on the dashboard: mechanism health, shot readiness, and any alerts that surface from the AlertLogger. The LED system ensures that the single most important piece of real-time information — *can the robot shoot right now, and is the target about to change* — is always available to the primary driver without requiring any communication from the operator. Everything else flows through the verbal channel.

The orange brownout strobe is a specific addition in 2026 that addresses an edge case from prior seasons. When the robot detects a voltage drop below the brownout threshold (approximately 7.0 V, with buffer above the hardware brownout at 6.5 V), it sets a persistent brownoutFlag in RobotState. If the robot later disables — whether at the end of a match or due to a field fault — the LEDs display the orange strobe rather than the normal rainbow, immediately signaling to the drive team and pit crew that a power event occurred during the match. This information is also available in the log file, but the LED signal ensures it is visible on the field cart the moment the match ends, before any post-match analysis has begun.

ENGINEERING TEAM

Team Captain

Anya Jiang

Vice-Captain

Luke Szigety

Design Lead	*Open Alliance Lead(s)*	*Outreach Lead(s)*	*Business Lead*
Constantina Flevarakis	**Henry Marsland & Yuhan Chen**	**Hamsini SIddi & Duanmu Jiang**	**Isabel Wu**

STUDENT MEMBERS

Sarah Lu	**Aarthi Movva**	**Liyann Alrayashi**	**Cat Pham**
Ben Nguyen	**Carson Koukkos**	**Marshall Atkins**	**Melissa Yang**
Erin Hwang	**Lily Xu**		**An Pham**

WHS Faculty Advisor

James Looney

Senior Mentors

Dwight Meglan **Chris Aloisio°** **Steve Harrington°**

Mentors

Anthony Gelsomini **Manny Barros°** **Mark Holthouse** **Charley Marsland °** **Luke Maxwell°**

Mentor Emeritus

Eric Yamaguchi°

Westwood Robotics, Inc Board Of Directors

Steve Harrington° *President*	**Catherine Tseng** *Treasurer*	**Manny Barros°**	**Cheten Gopal** *Clerk*	**Jack Tseng**

**Denotes High School Seniors* *°Denotes FRC Alumni*

SPONSORS

Cardathea	Westwood High School	Mathworks	Global Partners
TE Connectivity	Group 1 Automotive	Needham Bank	Dedham Savings
Poirer	Google	Roche Bros	Polymaker
Comella's	Jake N Joes		Print Master

VENDORS

Andymark	West Coast Products	Vex Pro	Swerve Drive Specialties
Discord	Google Work	Onshape	Robo Promos
McMaster-Carr	The Home Depot	Cross the Road Electronics	MS Visual Studio
Python	GitHub	RobotPy	Java

1757

Special thanks to everyone who makes this team possible Including

Our Students
Our Mentors
Our Teachers
Our Parents

Our Amazing Sponsors

Westwood Public Schools
Timothy Piwowar - Superintendent
Caitlin Ahern - Asst. Superintendent
Cindy Cox - Building Use Coordinator
Tom Carey - Director of Facilities
Ron Smith - Asst Director of Facilites
Amy Davenport - Principal
Aishleen Marcus - Assistant Principal
Tom Millett - Assistant Principal
Andrew Miller - Science Dept. Chair
WHS Janitorial/Facilities Staff
WHS Library Staff

#FRC1757
@WWRobotics1757