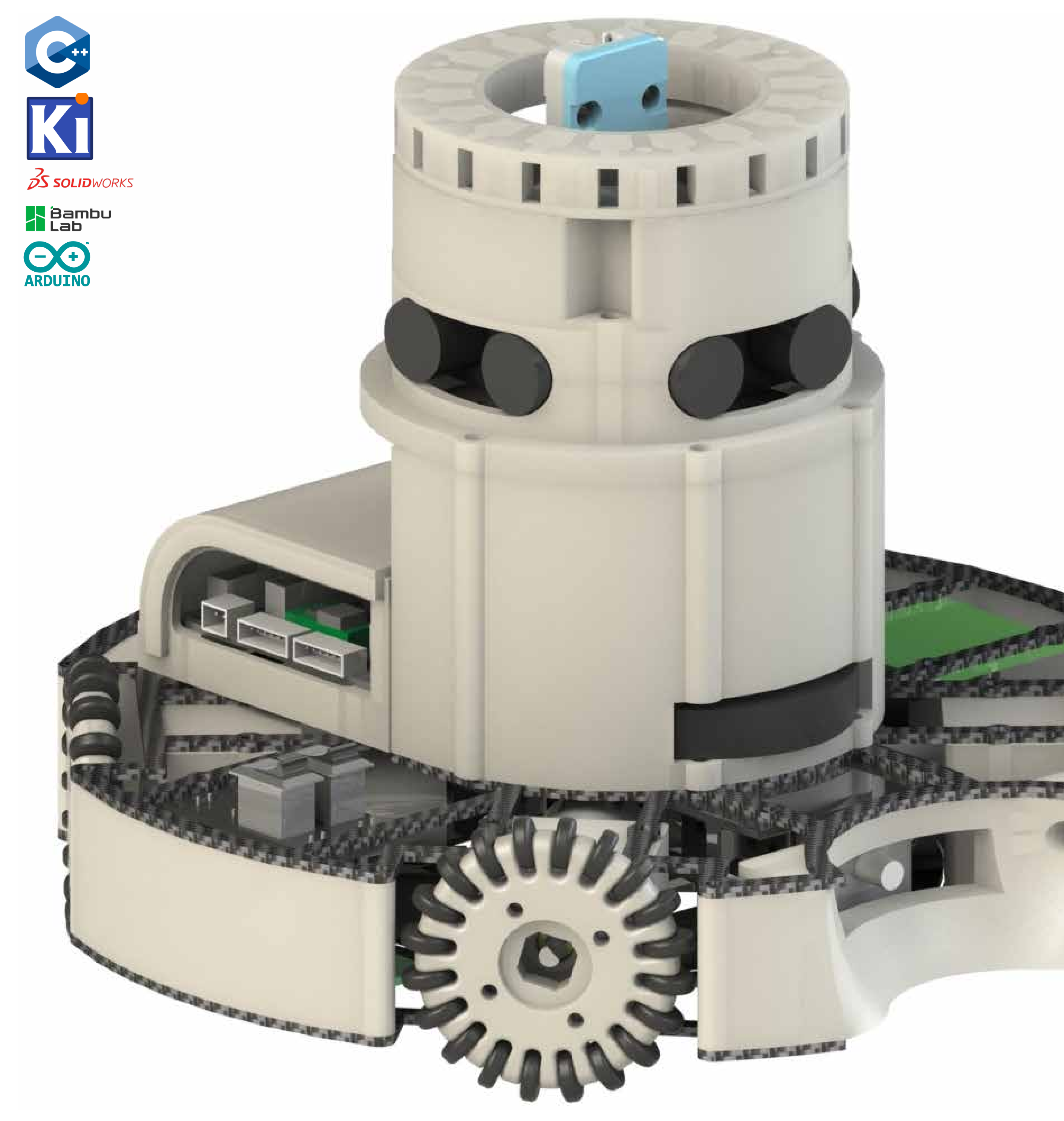
RoboCup Junior Soccer Lightweight

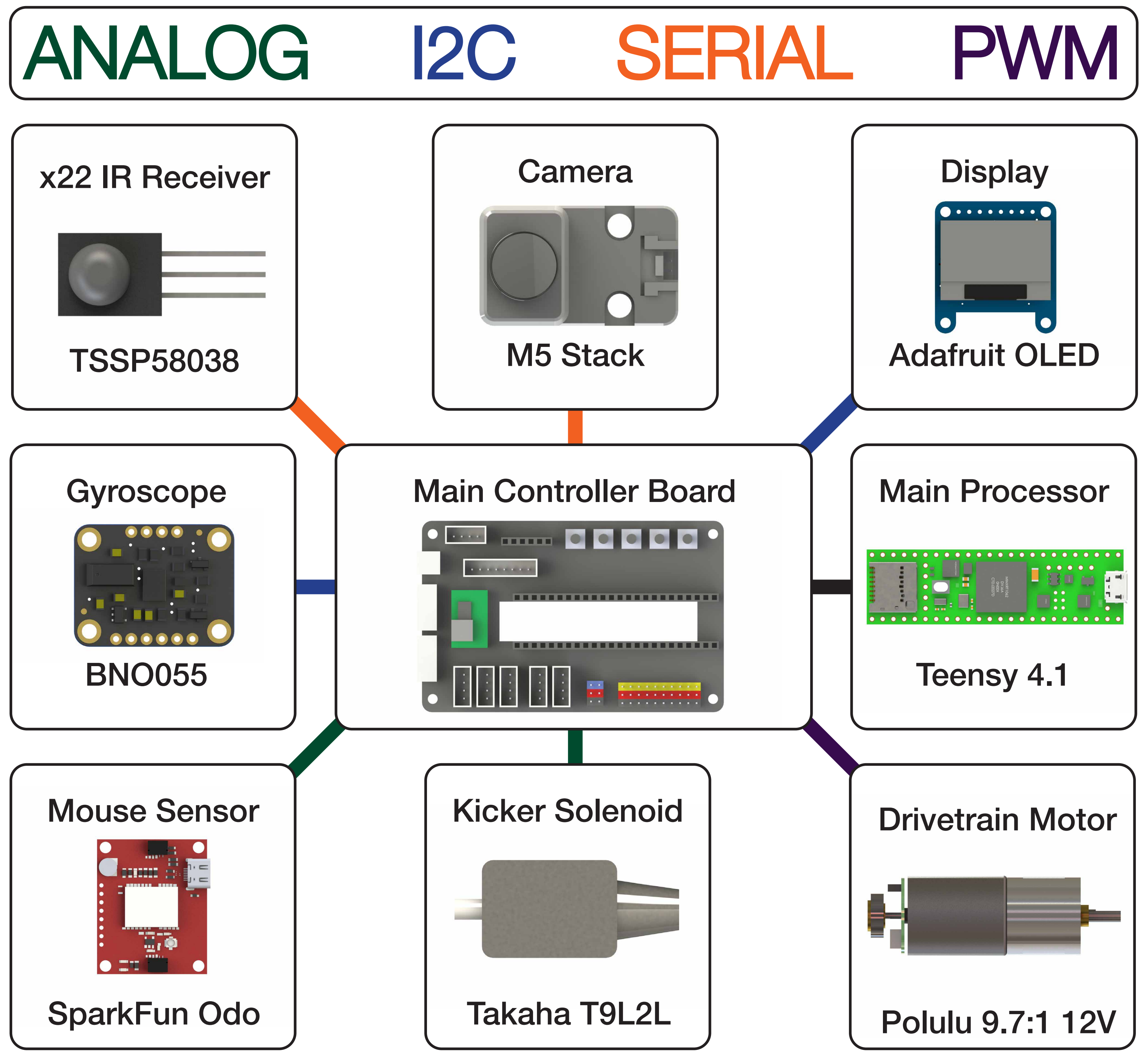
Lovbot Legends

Youtube Channel



Cost of Development
~$1400 USD
Time
Mechanical: 200+ Hours Electrical: 50+ Hours Software: 500+ Hours Strategy: 20+ Hours

Bill of Materials

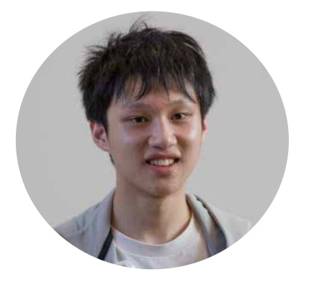


Abstract

Lovbot Legends is a Canadian RoboCup Junior Soccer Lightweight team of two students returning to the world championships for the third year. Key innovations this year include a custom “invisible barrier” algorithm for field boundary enforcement, a vector-based line-following system for dynamic defense, and enhanced localization through an M5 camera and optical mouse sensor.

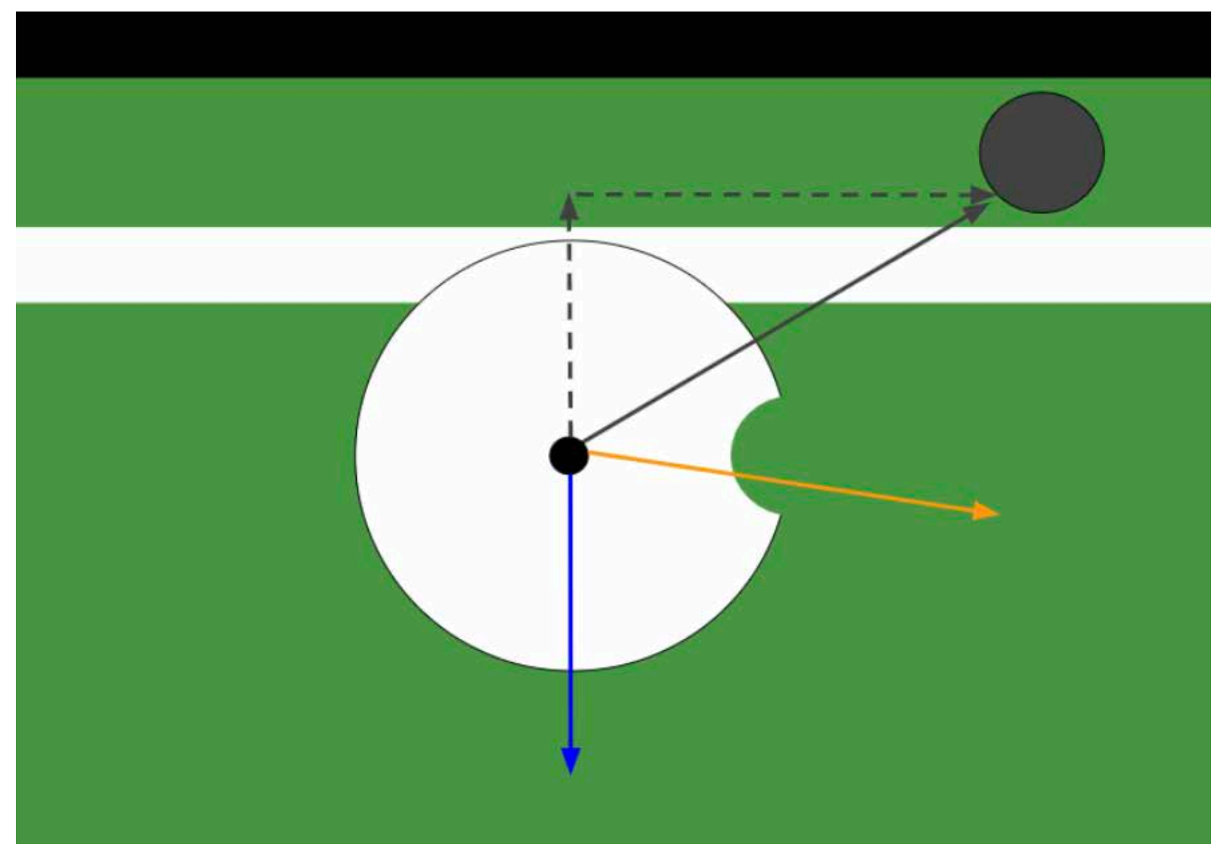
Informed by past hardware failures, this year’s design emphasizes durability with simplified hardware, improved motor consistency, custom lightweight omniwheels, and a front-mounted IR sensor that increases ball control under pressure.

Will Wu
Responsibilites: Hardware; Electrical; Offense Strategy
Past Teams: Lovbot Quantum; MapleTech Nebula



Kyle Gu
Responsibilites: Software; Camera; Defense Strategy
Past Teams: Lovbot Titans; MapleTech Elite

Invisble Barrier Detection



To prevent field boundary violations without relying on optical line detection, our robot implements a virtual “invisible barrier” using repulsion-based control logic. This system estimates proximity to the field edge via ultrasonic distance sensors positioned on all four sides of the robot. Instead of discrete stop conditions or hard-coded boundaries, we apply a continuous repelling force vector calculated from distance readings. The repelling force is computed independently in both the X and Y axes, using the piecewise function:

Live Demo

$$F(d) = \begin{cases} 1.0, & \text{if } d \geq d_{\text{start}} \\ 1.0 - \dfrac{d_{\text{start}} - d}{d_{\text{start}} - d_{\text{stop}}}, & \text{if } d_{\text{stop}} < d < d_{\text{start}} \\ 0.0, & \text{if } d \leq d_{\text{stop}} \end{cases}$$

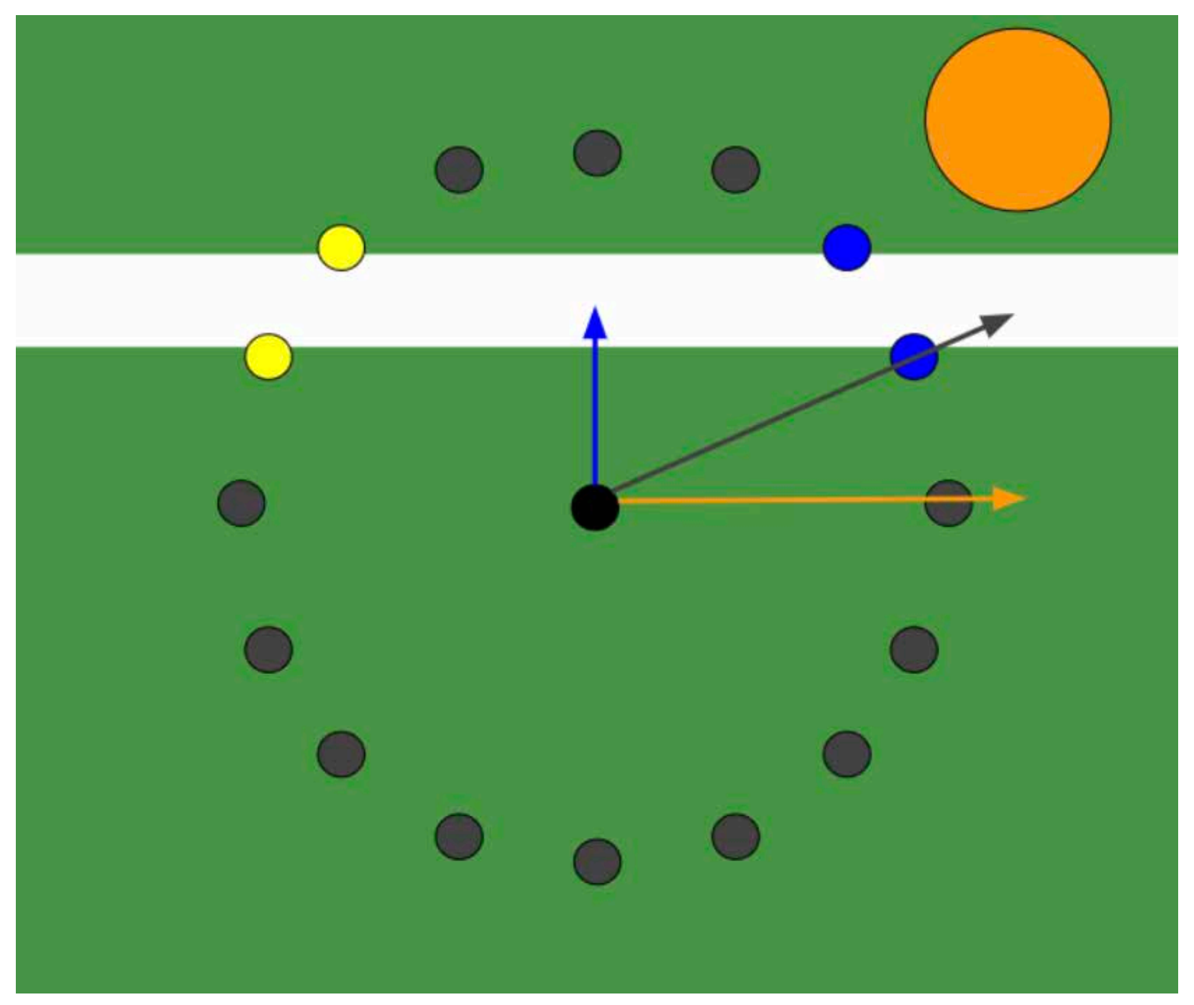
d_{start} : Distance at which repelling force begins

d_{stop} : Critical distance where repelling force is maxed

$F(d)$: Repelling force applied to the robot (0.0 to 1.0)

To reduce sensor noise, we applied a smoothing technique using a 5-value sliding window: the highest and lowest values were dropped, and we return the average of the middle three values. When the ultrasonics are blocked, we return a reading from the mouse sensor which shows the deviation from the last reliable localization.

White-line Following Algorithm



$$\vec{F} = \vec{W} + \lambda \cdot \vec{D}$$

$\vec{F}$: Final movement vector applied to the robot

$\vec{W}$: White line direction vector derived from grayscale sensor data

$\vec{D}$: Defensive direction vector (typically 90° or 270° in robot frame)

λ : Tunable weight that biases the robot toward defensive positioning

Live Demo

To follow the white line while maintaining optimal defensive positioning, the robot calculates a movement vector by combining the detected white line direction with a bias vector that points toward the ideal defensive angle (left or right of the ball). The scalar weight λ adjusts the influence of the defensive intent, allowing the robot to smoothly align itself between the goal and ball while staying anchored to the field’s white boundary.

The white line direction vector is derived from our 32 light sensor board, which we then calculate a smoothed angular estimate of the line’s orientation relative to the robot’s chassis using clusters. This vector, normalized to unit length, enables consistent behavior regardless of speed. Simultaneously, the defensive direction vector is set to either 90° or 270° in robot-relative coordinates, depending on the lateral position of the ball. A threshold is applied to the angular difference between the ball and the robot’s heading to determine when to switch sides. Finally, we have a tuningRatio which we can always change based on the line angle.

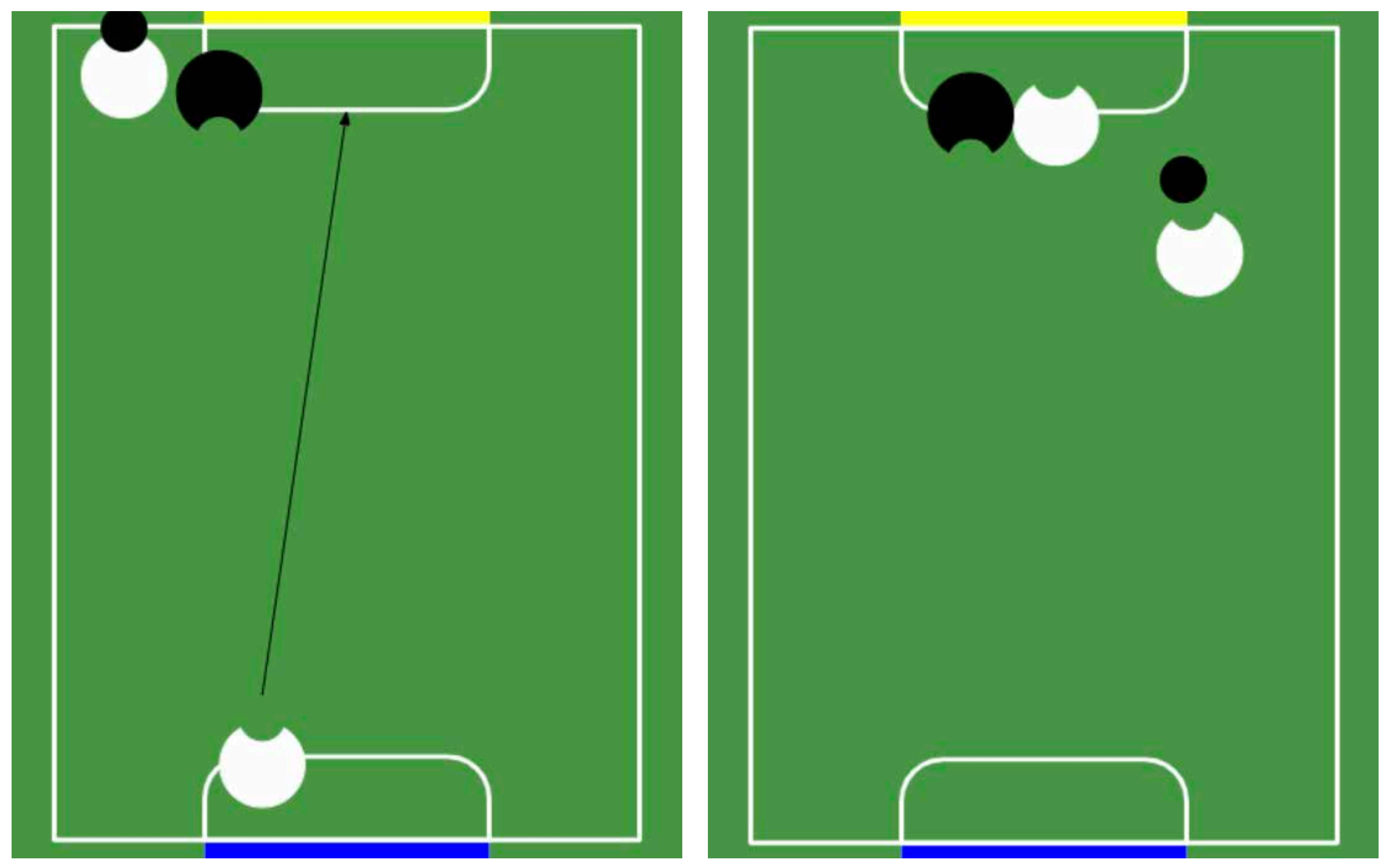
3D Printed Omniwheels

Feature	Commercial	3D Printed
Material	Metal	PLA
Price	$47	$12
Weight	32g	24g
Customization	None	High
Size	54mm	66mm
Durability	High	Medium

To reduce cost and improve modularity, we developed our own 3D printed omniwheels using a PLA hub and rollers from the GTF commercial omniwheels. Compared to commercial metal alternatives, our wheels are significantly cheaper and approximately 25% lighter (24g vs. 32g), enabling improved acceleration and reduced inertial load on the drivetrain. The hexagonal bore allows for direct motor shaft mounting without additional adapters, and the modular CAD design supports rapid iteration and field replacement. While durability is moderately reduced, we opted for the 3D printed wheels due to their performance benefits and customizability.

This design was inspired by Croatia’s robot, which we competed against at SuperRegionals. After observing their lightweight omniwheel setup in action, we adapted the concept to fit our own design constraints, leveraging accessible materials and rapid prototyping to implement a similar configuration.

Block Goalie Strategy



Live Demo

Pushing Rule:
If an attacking and a defending robot touch each other while at least one of them is at least partially inside the penalty area, **and** at least one of them has physical contact with the ball, this may be called ”pushing” at the referee’s discretion. In this case, the ball will be moved to the furthest unoccupied neutral spot immediately.

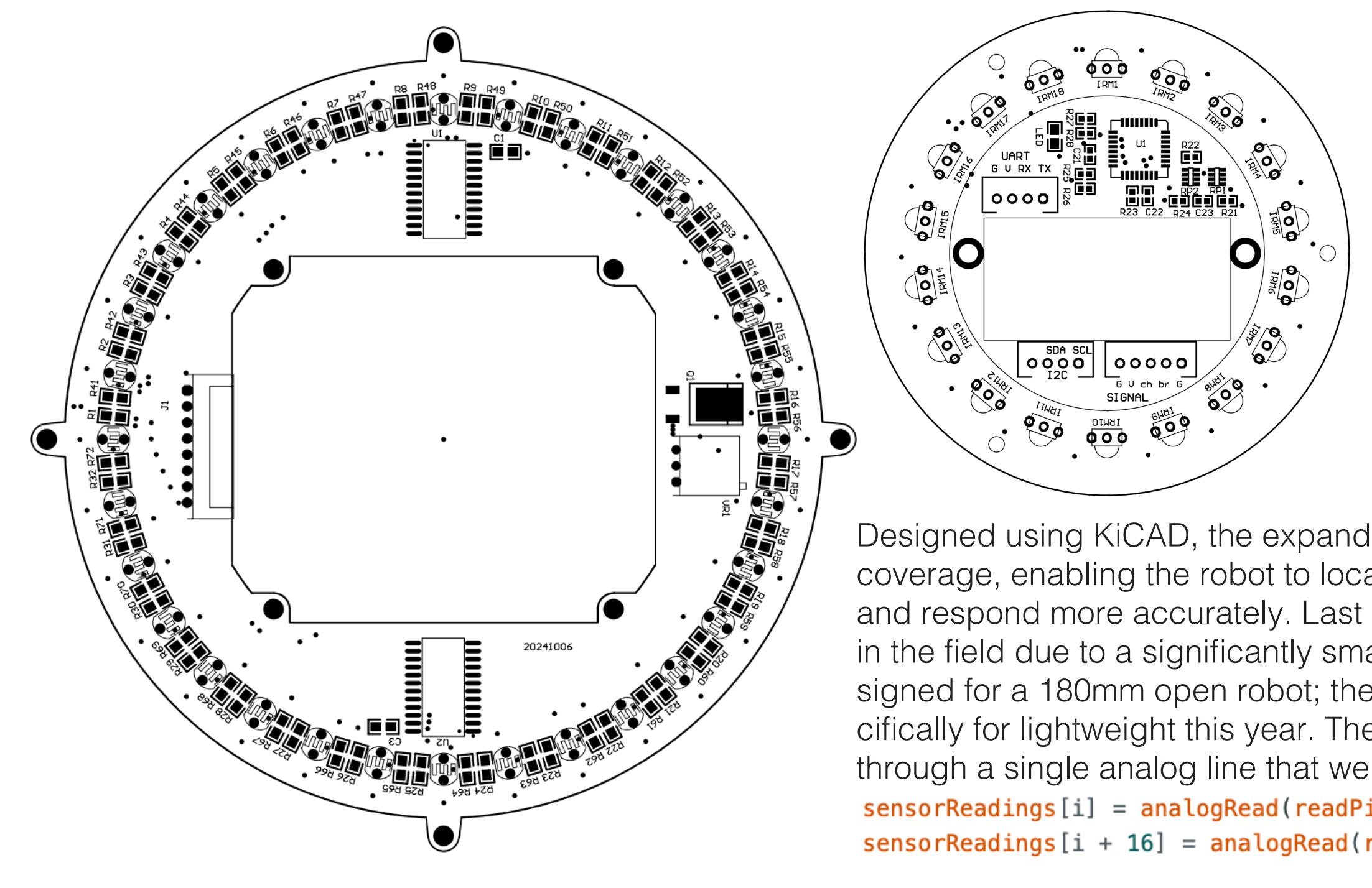
In recent years, the Lightweight division has evolved into a game dominated by near-immovable defensive robots, turning matches into wars of attrition where the outcome often hinges on which robot gets pushed out first. This meta favors low-risk, high-defense play and introduces a level of variance where the stronger team may not always win within the limited 20-minute match time. We’ve found that this defensive-heavy style leads to stale gameplay and outcomes that don’t always reflect overall performance or coordination.

To address this, we developed a block goalie strategy that operates within the rules while giving us a consistent scoring edge. According to the RoboCup Junior rules, a robot is only considered to be “pushing” if it is simultaneously in contact with both the opposing defender and the ball. If our defense robot makes contact only with the opposing defender—without touching the ball—it is not considered pushing under the rule definition.

Our implementation uses a Bluetooth communication protocol between our offense and defense robots. When the offensive robot detects a clear opportunity or is in scoring position, it wirelessly signals the defense robot to break from the goal and block the opposing defender. By positioning itself between the opponent’s defender and the ball, our robot effectively freezes their ability to respond—without violating the pushing rule. This reliably results in a free goal, leveraging both the rulebook and our coordination to reduce variance and control the tempo of the game. From a technical standpoint, the strategy relies on seamless coordination between robots and precise localization. Once the offense robot sends the Bluetooth trigger, the defense robot exits its standard goalie position and begins searching for the opposing defender.

HC-05 Bluetooth Module

Electrical Circuits

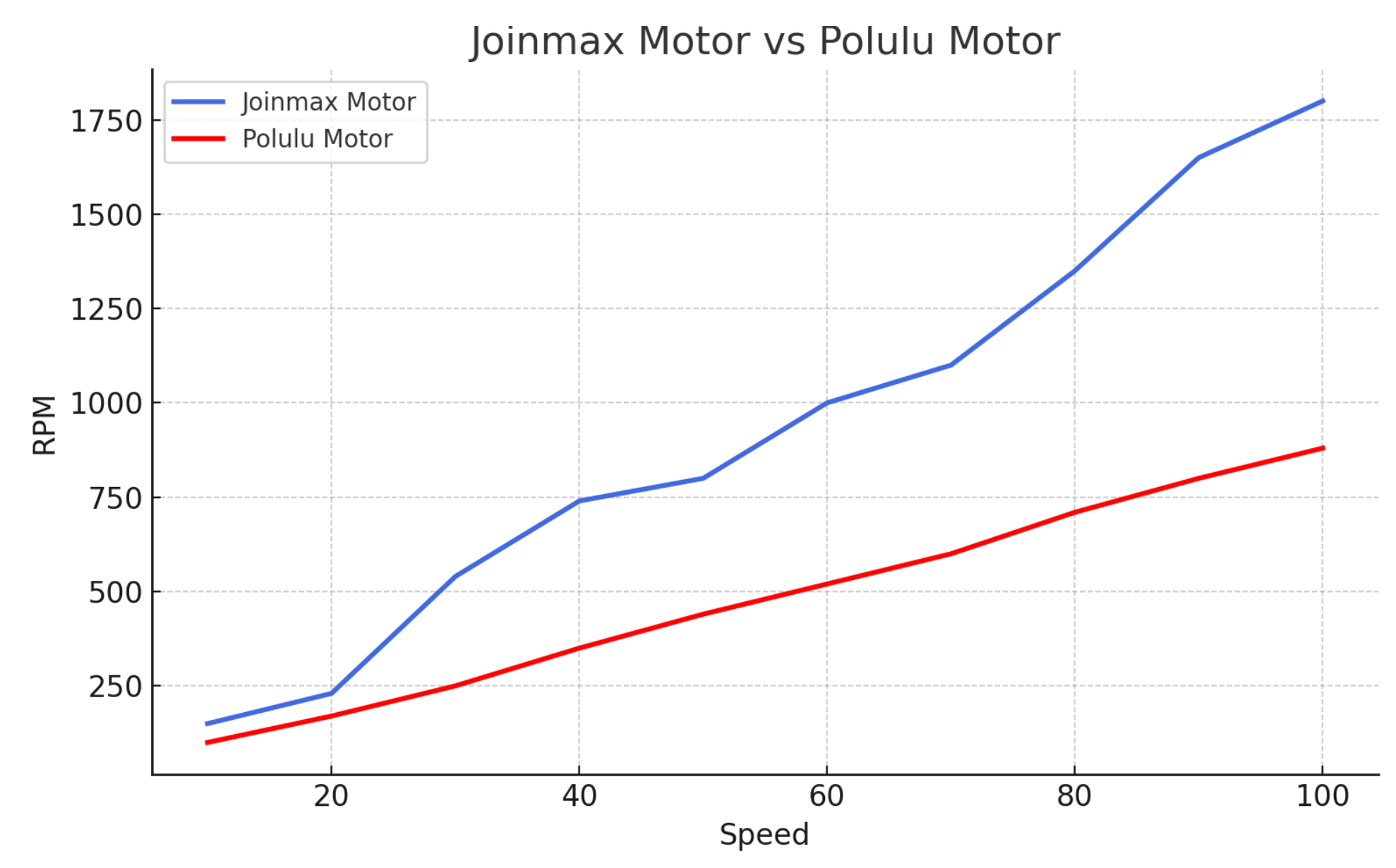


Our robot consists of two new PCBs this year: a redesigned 18-channel compound eye along with a 4-channel front IR sensor and a new, larger 32 circular light sensor. The compound eye greatly improves angular resolution compared to commercial 14 channel sensors, minimizing blind spots and allowing smoother ball tracking. We achieved nearly four times the detection range of commerical eyes.

Designed using KiCAD, the expanded 32-sensor light ring increases coverage, enabling the robot to localize edges from any orientation and respond more accurately. Last year, we struggled to stay within the field due to a significantly smaller light sensor originally designed for a 180mm open robot; the new design was tailored specifically for lightweight this year. The PCB communicates efficiently through a single analog line that we can easily read into an array:

```
sensorReadings[i] = analogRead(readPins[0]);
sensorReadings[i + 16] = analogRead(readPins[1]);
```

Motor RPM Accuracy

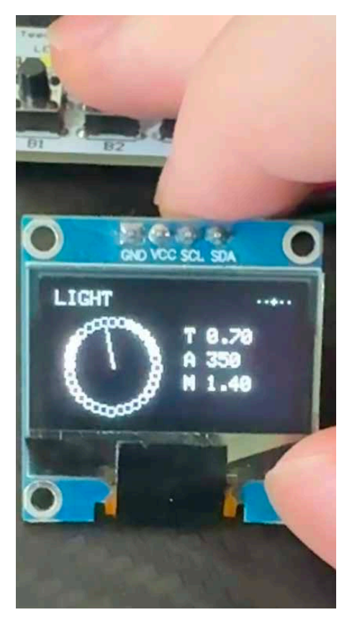


We initially used Joinmax Motors (JMP 12 DC motors) in our robots. However, during tuning, we noticed inconsistencies: the speed didn’t seem to scale linearly, causing problems like going at the wrong angle by a few degrees. To verify this, we used a speedometer to measure the rotational speed of the motor at different speeds.

The RPM results for the Joinmax clearly showed significant inconsistencies. The RPM wasn’t linear, and was quite erratic. Due to these irregularities, we ultimately decided to explore alternative motor options. After testing, it became clear that the Pololu motors offered much greater consistency, with a RPM that was very close to linear.

Display UI

Live Demo

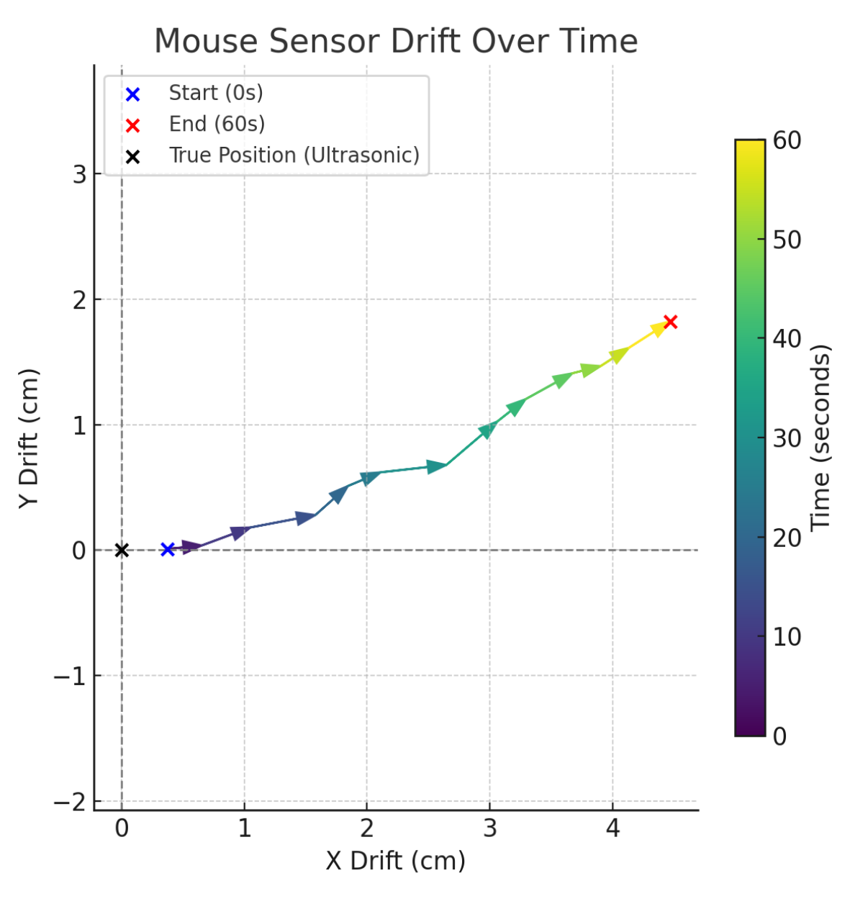


To streamline debugging, testing, and in-game configuration, we developed a five-page OLED display UI directly integrated into the robot. The system allows us to toggle between offense and defense, switch team sides, and monitor key sensor data in real time—all without needing a laptop or reflashing code.

This became especially important during matches, where rapid adjustments and clear feedback are critical. Instead of relying on print statements or guesswork, we can visually verify compass heading, motion direction, ball detection, ultrasonic distances, and white line angles at a glance. The UI not only speeds up development by reducing iteration. By centralizing all diagnostics into a compact interface, we’ve dramatically improved our ability to troubleshoot in the field, fine-tune behavior during testing, and ensure the robot’s configuration is always correct before a game starts.

In high-pressure situations where time is limited, the display provides immediate clarity about what the robot is “seeing” and how it’s behaving. This visibility has helped us catch subtle bugs, confirm sensor functionality, and stay flexible in dynamic match conditions. Its portability and simplicity have made it an essential part of our debugging workflow.

Mouse Sensor



Graph generated using Python’s Matplotlib library with numpy-based simulation of cumulative drift.

We use a downward-facing optical mouse sensor to estimate the robot’s relative position on the field by tracking surface displacement in X and Y directions. Unlike wheel encoders, which suffer from slippage and mechanical backlash, the mouse sensor provides smooth, high-frequency motion data directly from the turf surface. This information is critical—particularly when ball handling , rotating in place, or at high speeds.