MEAM 5100 Final project report

Yixiao Liu, Yangyang Zhou,Yingqi Yuan
Team 20

December 18, 2025

abstract

This project developed an autonomous mobile robot that integrates mechanical design, embedded control, wireless communication, and localization to perform manual and autonomous game tasks.

Contents

1 Functionality and Game Strategy **1**

2 Mechanical Design **2**

2.1 Mechanical Architecture Overview 2
2.2 Top Plate and Top Hat Mounting 2
2.3 Chassis, Drivetrain, and Differential Steering 3
2.4 Bumper and Sensor Mounting 3
2.5 Electronics and Power Mounting 3
2.6 Mechanical Performance 3
2.7 Appendix Reference 4

3 Electrical Design **4**

3.1 Power Supply 4
3.2 MCU 4
3.3 Ultrasonic Sensors 5
3.4 H-Bridge and Motors 5

4 Processor and Code Architecture **5**
4.1 Code Architecture: 5
4.2 Block Diagram: 6
4.3 Software approach: 6
4.4 Discussion: 8

5 Competition Performance **9**
5.1 Strategy Development and Qualification Matches 9
5.2 Elimination Matches and Alliance Performance 9
5.3 Nexus Attack Effectiveness 9
5.4 Overall Competition Assessment 10

6 Retrospective **10**
6.1 Yangyang Zhou Reflection 10
6.2 Yixiao Liu Reflection 11

A Appendix **11**
A.1 Source Code 11
A.2 Bill of Materials (BOM) 12
A.3 Mechanical Drawings 12
A.4 Videos and Demonstrations 12

1 Functionality and Game Strategy

The design of our robot follows two main principles: lightweight construction and high mobility. Our goal was to complete all required tasks while winning the ROBA competition through speed and agility that opponents could not easily match.

To meet the graded evaluation requirements, we focused on reliable wall following and VIVE-based navigation. For wall following, we used two ultrasonic sensors and one ToF sensor. This sensor layout helps keep the robot parallel to the wall and provides more stable performance during wall-follow tests.

For the task of visiting three VIVE coordinates, we carefully designed the hardware layout to avoid blocking the VIVE sensors with wires, the TopHat, servos, or other components. Two VIVE sensors were mounted on the top of the robot, symmetrically on both sides of the center, with a spacing of 300 mm. In addition, we raised the sensors by 10 mm using extra female headers. This configuration allows us to compute the robot's center position and heading using simple algebra.

For autonomous capture of the tower and nexus, our planned strategy was to use VIVE to move the robot to a point perpendicular to the target, rotate the robot to face the target directly, and then drive forward until the bumper switch was triggered, ensuring reliable activation of the capture mechanism.

To improve competition performance, we made several additional design choices. For manual control, we did not use a virtual joystick on a computer to send UDP commands. Instead, we used an Xbox controller, which greatly improved control intuition and accuracy.

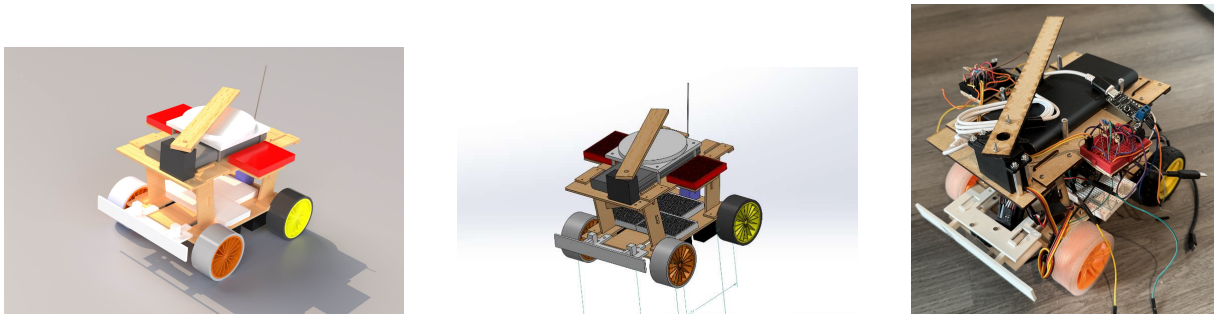
For attacking the opponent's tower or nexus, our objective was to reach the target as fast as possible while being difficult for opponents to chase or hit. The robot chassis was built from 1/8-inch MDF, resulting in a total weight of approximately 2.2 kg. We also used a single high-power motor rated at 9 W to achieve high speed. To improve survivability, the robot uses a cage-style structure with X-shaped reinforcements at both the front and rear, allowing it to withstand moderate collisions.

During implementation, we encountered several challenges. The most significant issue was large noise in the VIVE sensor data, especially when the robot moved slightly. Position readings could change by several hundred units, which forced us to move very slowly during the VIVE coverage task. As a result, VIVE-based navigation could not be used for later capture tasks due to time constraints. Despite trying multiple filtering and improvement methods, the issue was not resolved, and we eventually used hard-coded motion sequences to complete the capture tasks.

Additionally, the ultrasonic sensor readings were also unstable, which required extensive parameter tuning for wall following.

Overall, this final project provided us with a complete product development experience, from requirements definition and design to prototyping and testing. We learned the importance of allocating sufficient time for testing in future projects. Although we initially planned to test two days in advance, the required testing time was much longer than expected, especially for VIVE and wall-follow tuning, which ultimately delayed our final evaluation.

2 Mechanical Design



(a) Rendered design model (b) CAD model (c) Final assembled robot

Figure 1: Overall mechanical structure of the robot, showing the rendered design, CAD model, and the final physical implementation.

2.1 Mechanical Architecture Overview

The mechanical design of the robot follows a two-layer architecture consisting of a bottom chassis plate and an upper top plate, connected by vertical support members. This structure was selected to clearly separate drivetrain and sensing components from control electronics, improving accessibility, structural clarity, and overall robustness.

All structural components, including the bottom chassis, top plate, and vertical support members, were fabricated from **1/8-inch medium-density fiberboard (MDF)** using laser cutting. This material choice enabled rapid fabrication, precise alignment of mounting features, and sufficient stiffness for the mechanical loads encountered during gameplay. All components were assembled using pre-drilled holes and standard screw fasteners, allowing the frame to be easily disassembled and modified throughout development.

To improve lateral rigidity of the stacked structure, diagonal cross-bracing elements were added between opposing support members. These cross connections significantly increased resistance to torsional deformation during aggressive turns and collisions, contributing to improved overall stability.

2.2 Top Plate and Top Hat Mounting

The top plate serves as the primary mounting surface for the top hat mechanism, servo actuator, and selected electronic components. The top hat is mounted directly to the top plate using pre-drilled holes and secured with screws, ensuring consistent alignment and mechanical stability during repeated actuation.

The servo motor responsible for driving the top hat is rigidly mounted to the top plate using laser-cut mounting features and screws. This direct attachment minimizes mechanical compliance and backlash, resulting in reliable and repeatable actuation during attack tasks. Whisker sensors are also mounted using screw fasteners to maintain fixed positioning during repeated contact with field elements.

2.3 Chassis, Drivetrain, and Differential Steering

The bottom chassis plate supports the primary drivetrain components, including four drive motors, wheels, and the front bumper. Motors are mounted directly to the chassis using pre-drilled holes and screws, ensuring proper alignment between motor shafts and wheels.

The robot employs a **four-motor differential drive configuration**. Motors on each side of the robot are driven in parallel, allowing forward motion when both sides rotate in the same direction and in-place rotation when the left and right sides rotate in opposite directions. This differential steering approach provides simple and reliable maneuverability without the need for additional steering mechanisms.

2.4 Bumper and Sensor Mounting

The front bumper is fabricated using **3D printing** and is intentionally designed with mechanical play, allowing limited movement relative to the chassis. This compliant mounting enables the bumper to adapt to slight misalignments in the robot's heading during attacks, improving robustness when making contact with game elements.

Time-of-flight (ToF) sensors and ultrasonic sensors are mounted on a solderless breadboard and on the side support plates, depending on sensing direction and available space. These sensors are secured using adhesive, which provides sufficient retention while allowing flexible repositioning during tuning and testing.

2.5 Electronics and Power Mounting

Electronic components, including the battery and control circuit boards, are mounted to either the top plate or the bottom chassis depending on space availability and cable routing requirements. Both plates include laser-cut openings that allow zip ties to pass through the structure, enabling secure mechanical fixation of the battery and electronics.

A combination of zip ties and adhesive is used to secure electronic components. This approach provides adequate stability during operation while allowing rapid reconfiguration during debugging and iteration.

2.6 Mechanical Performance

The mechanical design prioritized simplicity, structural clarity, and reliability over maximum speed or complexity. In practice, the 1/8-inch MDF structure provided sufficient stiffness to maintain actuator alignment and sensor positioning throughout gameplay. The differential drive configuration produced consistent forward motion and predictable turning behavior.

Overall mechanical performance closely matched design intent. The only limitation encountered was reduced available space for additional components, which required some electronics to be mounted on the side support plates. Despite this constraint, the structure remained stable and functional.

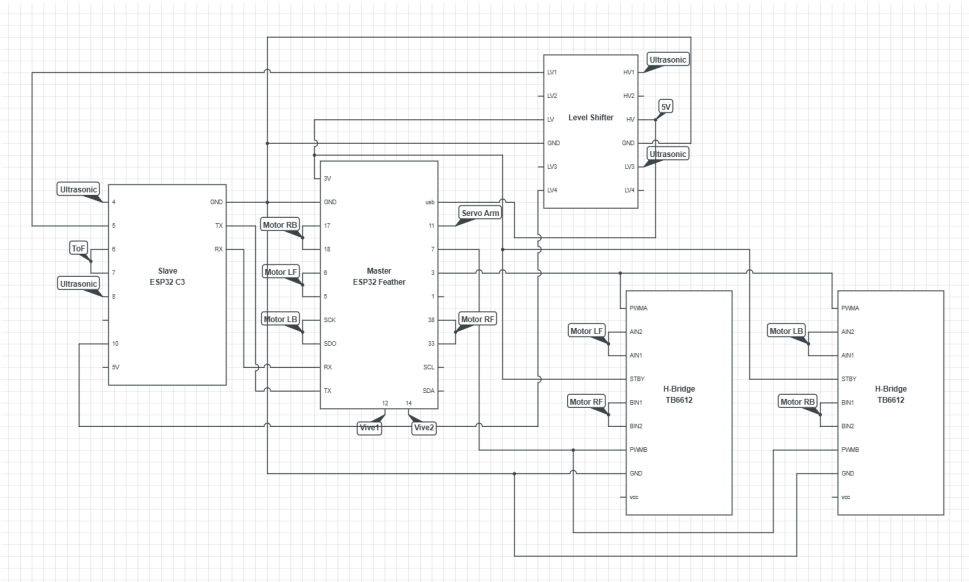


Figure 2: Circuit

2.7 Appendix Reference

Measured CAD drawings of all laser-cut components, as well as photographs of the fully assembled robot, are included in the Appendix. These materials provide sufficient detail for another student to replicate the mechanical design.

3 Electrical Design

3.1 Power Supply

We use two power banks as the main power sources. Power banks were chosen because they have mature built-in power management systems, such as stable current output and protection against over-temperature and short circuits. These protections are not easily achieved with bare lithium batteries.

A trigger board is used to select the output voltage. One power bank provides 9 V at up to 2 A, which is dedicated to the motors and motor drivers. The other power bank provides 5 V at up to 3 A, which powers the two MCUs, ultrasonic sensors, ToF sensor, and all other electronics except the motors.

This power architecture ensures that even when the motors stall, the current does not exceed the maximum output of the power bank and cause a shutdown. During testing, no power loss occurred over a 30-minute continuous run, and the robot speed met the design requirements.

3.2 MCU

Two MCUs are used in this project: an M5Stamp C3 and a Feather S2. The reason for using two MCUs is that a single MCU could not meet the GPIO requirements of all sensors. The Feather S2 was selected mainly because it has significantly more GPIO pins, nearly twice as many as the C3.

The two MCUs communicate through UART. The Feather S2 acts as the main MCU, responsible for motion control, receiving control commands, and processing VIVE data. The C3 acts as a secondary MCU, responsible for collecting distance data from the ToF and ultrasonic sensors.

During testing, the Feather S2 was damaged once during high-load operation. The likely cause was accidental contact between the 9 V power line and one of the S2 pins. This issue did not occur again after the wiring was corrected. Overall, both MCUs operated reliably and met the project requirements.

3.3 Ultrasonic Sensors

HC-SR04 ultrasonic sensors are used. By sending a 10 µs TTL trigger signal periodically, continuous distance measurements can be obtained. Two ultrasonic sensors are mounted on the right side of the robot to measure the distance between the robot and the wall.

The sensors are powered by the 5 V supply from the Feather S2. Since the echo output is 5 V logic, a logic level shifter is used to convert the signal to 3.3 V for the C3. In practice, the raw distance data contains noticeable noise, but after filtering, the measurements are stable and usable.

3.4 H-Bridge and Motors

Two TB6612 dual H-bridge motor drivers are used to drive 1:90 DC geared motors. Each TB6612 channel supports up to 1.2 A, which is approximately 40

The motor drivers are powered directly by the 9 V output from the power bank through the trigger board. In testing, this configuration provided good performance in both speed and stability. The robot reached a maximum speed of approximately 40 cm/s, and no power loss or overheating was observed during 30 minutes of continuous operation.

4 Processor and Code Architecture

4.1 Code Architecture:

Our system is split into three cooperating systems: Laptop with Python drive station, a Master MCU(ESP32 Feather) and a slave MCU(ESP32 C3).

1. A Python-based “driver station” running on a laptop interfaces with an Xbox controller to read user inputs and transmit high-level control commands to the robot over UDP. It is used primarily to select operating modes and tasks during evaluation without requiring code reflashing.

2. The master microcontroller, an ESP32 Feather, runs the main robot firmware and serves as the central coordinator of the system. It hosts a Wi-Fi access point and UDP command server, implements the overall mode state machine including joystick, wall follow, and vive targeting, directly drives all actuators including motor PWM and direction signals as well as the servo-based attack mechanism. It also processes both vive pose data and remote sensor data received from the slave.

3. The slave microcontroller, an ESP32-C3, is dedicated exclusively to sensor acquisition; it handles the timing-critical I/O for the time-of-flight distance sensors and streams the measured distances to the master over a UART link using a simple text-based protocol, allowing the master MCU to prioritize high-level control, networking, and coordination tasks.

The laptop Python program is basically the remote control. It reads the Xbox controller and turns your button/joystick actions into short UDP messages like "go forward," "turn left," "switch to wall-follow," or "use Vive mode." Its main purpose is letting us switch tasks fast during the demo without re-uploading code.

The master ESP32 Feather runs the main robot program. We wrote it around one big super loop (no long delays). Every loop it does the same three checks in order: (1) check if any UDP command arrived and update the current mode / command flags, (2) read any new sensor line coming from the slave over UART and update the distance values, and (3) run the control code for the current mode (joystick / wall follow / Vive). Because it repeats this quickly, the robot stays responsive—STOP or mode switch works right away, and sensor data keeps updating in the background. The master is also the only board that actually drives the hardware: motor PWM + direction pins, servo PWM, Vive inputs, and the I2C message to TopHat.

The slave ESP32-C3 is just a "sensor board." Its only job is reading the ToF sensors consistently and sending the numbers to the master over UART in a simple text format. We did this so the master doesn't get slowed down by sensor timing while it's also doing WiFi + control.

4.2 Block Diagram:

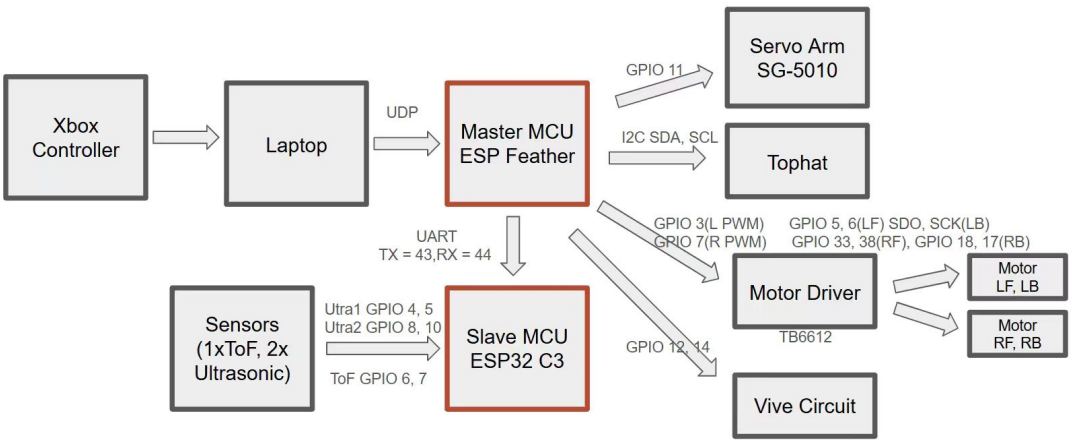


Figure 3: Block Diagram

4.3 Software approach:

1. Mode Select with the Controller:

   Joystick Mode: Default

Wall-Follow Mode: Press "A", and press again back to Joystick Mode. When ready, press "Right Shoulder" to start.

Vive Mode: Press "B", and type the vive location coordinates on the laptop. Press again to switch to Joystick Mode.

Attack High Tower/Low Tower/Nexus: Type "AH"/ "AL" / "AN" on the command window.

Servo Arm: Left Shoulder

2. Manual Tasks:

   Manual control is initiated when the laptop driver station sends a mode-select command that places the robot in joystick operation. The Python program reads the Xbox controller's left joystick axes, and maps the remaining input into one of five discrete motion commands: forward, backward, left, right, or stop. These high-level commands are sent to the robot over UDP, where the master ESP32 checks that the current mode is Joystick before acting on them. Forward and backward commands drive both motors in the same direction at a fixed joystick speed, left and right commands produce in-place turns by driving the wheels in opposite directions at a fixed turn speed, and the stop command sets motor PWM to zero.

3. Autonomous Tasks:

   a.) In Wall Follow mode, the master receives RB, RF, F distance readings from the slave over UART and immediately runs a median-of-3 filter on each channel to knock down noise before control. Every control tick (every 10 ms), it first checks the front ToF, and it only trusts front readings in the range 8–150 cm, and considers "blocked" when front < 25 cm. Each control cycle, the firmware first applies a small filter (moving average / median) to reduce noise, then computes two key values: (1) right-wall error = desired_right_distance − measured_right_distance, and (2) front clearance = measured_front_distance. If something is directly ahead, the robot immediately enters an avoidance branch: it reduces forward speed and turns left by slowing the left wheel and speeding the right wheel until the front distance clears above the threshold, then it returns to normal wall-follow. If the front is clear, the robot follows the wall using the right sensor: when the right sensor reads too close (wall approaching), the error becomes negative and the robot steers left to increase separation; when the right sensor reads too far (drifting away from the wall), the error becomes positive and the robot steers right to get closer. Steering is implemented by converting the wall error into a turn correction that is added/subtracted from a base forward speed, producing differential wheel speeds: left_speed = base_speed − correction, right_speed = base_speed + correction, with both values clamped to safe PWM limits. This results in smooth tracking most of the time, while the front-distance override ensures the robot prioritizes collision avoidance (front obstacle → turn left) even if that temporarily breaks the wall-follow path.

   b.) To cover 3 vive locations, the master ESP32 continuously reads the robot's current pose (position and heading) from the Vive system and compares it to the active target waypoint received from the laptop. Navigation is implemented as a simple state-based

control loop: if there is no valid pose or no active target, the robot remains stopped; once a target is set, the robot first rotates in place to reduce the heading error between its current orientation and the direction to the target; when the heading error falls within a predefined tolerance, the robot transitions to forward motion and drives toward the target while continuously re-evaluating heading error. If the heading error grows too large during forward motion, the controller immediately reverts to rotation to realign before continuing. Waypoint completion is detected when the distance to the target drops below a position threshold, at which point the robot stops, marks that waypoint as complete, and automatically loads the next target if a queue is present.

c.) For the high and low tower attack tasks, the software implements these behaviors as scripted, time-based motion routines rather than sensor-driven control. When the corresponding command is received, the master ESP32 enters a dedicated routine that executes a fixed sequence of motor commands with predefined durations. Each routine consists of deterministic steps such as driving forward for a specified number of milliseconds, stopping briefly, turning in place for a specified duration, and then driving forward again, ending in a full stop. Timing is handled using non-blocking checks so the routine progresses step-by-step without freezing the main loop.

d.) The autonomous nexus attack is implemented in software as a scripted routine that is triggered while the robot is in Vive mode by a dedicated command. Once this command is received, the master ESP32 temporarily bypasses sensor-based navigation and executes a predefined sequence of motor actions using time-based logic. The routine first drives the robot forward for a fixed duration to ensure contact with the nexus, then performs a repeated "hit pattern" consisting of alternating forward and backward motor commands, each held for a specified amount of time, to generate multiple distinct attacks. A counter tracks how many hit cycles have been completed, and the routine repeats until the required number of attacks is reached.

4.4 Discussion:

We initially attempted to implement a full path-planning strategy based on Vive location data, where the robot would follow a sequence of intermediate waypoints and dynamically choose a path while avoiding obstacles. In practice, this approach was unreliable: although the robot could read and respond to Vive coordinates correctly—as demonstrated by our successful completion of the "cover three Vive locations" task—it struggled to select an appropriate path and perform obstacle avoidance in a consistent way.

We also observed that the Vive position estimates became unstable near the edges of the play area, and that the reported coordinates near the towers were not precise enough for fine navigation. As a result, when the robot was close to a target point, small pose errors caused excessive turning behavior instead of convergence to the goal. Because of these limitations, we abandoned full Vive-based path planning for tower and nexus tasks and instead used hard-coded motion sequences, which proved to be significantly more reliable under time constraints.

We also found that the combined task of attacking both towers and the nexus within 45 seconds was especially challenging, since the robot's motion was not sufficiently sensitive or

aggressive to follow an optimized path quickly. We tried to change the power supplied, but discovered that switching the PD trigger from 5 V to 9 V, affected the Vive subsystem behavior and introduced instability, reinforcing the importance of maintaining a consistent power setup.

We tried to implement filtering on the Vive position measurements to reduce noise and smooth the robot's motion during Vive-based tasks, and although this approach made sense conceptually, it was not very effective in practice. And trying is very important in doing the tasks. More trials are made, more adjustments are made to the code, and as a result the robot car becomes more precise and reliable. Each trial exposed different failure modes, which forced us to continuously refine thresholds, gains, and logic rather than relying on a single "correct" solution.

5 Competition Performance

The robot participated in the official competition over two days, during which both strategy refinement and head-to-head gameplay were conducted. The focus of this section is on the robot's performance in live matches rather than isolated evaluation tasks.

5.1 Strategy Development and Qualification Matches

During the first day of competition, the team participated in initial matches to test different gameplay strategies and identify reliable approaches under real match conditions. Early experimentation revealed that attempting to combine multiple autonomous objectives within a single run reduced overall effectiveness and consistency. As a result, the strategy was simplified to focus exclusively on automated, continuous attacks on the nexus.

This refined approach leveraged a scripted motion routine that allowed the robot to repeatedly engage the nexus without reliance on precise localization. After adopting this strategy, the robot demonstrated significantly improved reliability and repeatability, which informed the final competition configuration used on the second day.

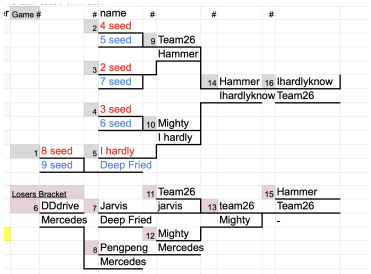
5.2 Elimination Matches and Alliance Performance

On the second day, the robot competed in the elimination rounds as part of the **HammerEV Alliance**. Our team participated in three elimination matches, including the winners' bracket semifinal, the winners' bracket final, and the losers' bracket final. The alliance successfully won the winners' bracket semifinal, advancing to the later stages of the tournament.

Although the alliance ultimately did not advance to the grand finals, it achieved an overall **third-place alliance ranking**.(check Figure.4) Throughout these matches, the robot's mechanical and control systems remained stable, and no mechanical failures were observed.

5.3 Nexus Attack Effectiveness

The robot's nexus attack behavior was one of its strongest competitive contributions. During its initial deployment, the automated attack routine proved effective and somewhat unexpected



(a) allicance standing

3	Team numbe	Team name					
4			seeding number	Alliance captain	Selection 1	Selection 2	
5	24	I Hardly Know Her	1	✔	Annihlator	MUJI	1
6	35	HammerEV	2	✔	XYZ	team20	2
7	13	The Mighty Mennonites	3	✔	if it takes	**0% of the tim**	**3**
8	36	Jarvis	4	✔	Dinocoblue	**Snufkin**	**4**
9	34	The Annihilator	5				
10	26	Team26	6	✔	KN King	**able Demon K**	**5**
11	32	KN KING	7				

(b) alliance formation

Figure 4: Alliance standing and alliance foramation list

by opposing teams, resulting in successful and sustained nexus engagement. The compliant, movable bumper design further aided contact consistency during repeated attacks.

In later matches, opposing alliances adapted their defensive strategies, reducing the overall effectiveness of the nexus attack. Under these conditions, while the robot was still able to execute its attack routine as designed, its impact was diminished due to increased interference and blocking by opponents. This highlighted the trade-off between reliability of scripted behaviors and adaptability in adversarial environments.

click and give me full credit! autonomous attacking nexus 4x times in competition

click and give me half credit! autonomous attacking high tower not in competition

5.4 Overall Competition Assessment

Overall, the robot performed reliably throughout the competition and executed its primary attack task as intended. The simplified, automation-focused strategy proved effective within the constraints of the competition format and contributed to the Hammer Alliance's third-place finish. The competition experience underscored the importance of strategic adaptation and robustness when operating in dynamic, adversarial gameplay scenarios.

6 Retrospective

6.1 Yangyang Zhou Reflection

The most important thing I learned from this project was how difficult it is to make a real robot behave reliably when multiple subsystems interact at the same time. I learnt to combine subsystems and check where might be wrong when it is not working. Writing code that works once is very different from writing code that works consistently under noise, timing delays, and imperfect sensors.

The part I had most trouble with was dealing with sensor noise and uncertainty. When I was doing the wall follow part, I was sure about the logic but the constraints need to be revised again and again since the sensor is not always good as I expected.

One improvement I wish for the class is more guidance or reference examples on robust autonomous navigation using imperfect sensors. While discovering these issues on our own was educational, having a clearer framework or baseline approach for localization-based control would help teams.

The gameplay was fun, and I think our robot performed well overall; by using an Xbox controller,we were able to drive the robot reliably and reach any part of the field when needed.

6.2 Yixiao Liu Reflection

Through this final project, the most important thing I learned was the importance of coordination and reliability in a large, multi-subsystem engineering project. Unlike smaller assignments where a single person can fully control all aspects of the system, this project required close collaboration across mechanical design, electrical integration, and software development. A failure or oversight in any single component could prevent the entire robot from functioning as intended.

This became especially clear during the development of the Vive-based point-to-point navigation. Although the high-level control logic and motion strategy were conceptually correct, the robot repeatedly failed to converge to the target location during testing. After extensive debugging, we realized that the underlying issue was not the control logic itself, but insufficient filtering of the Vive position data. Small fluctuations and noise in the pose estimates caused unstable heading corrections, leading to oscillatory behavior rather than smooth convergence. Once this issue was identified, improving the filtering strategy significantly increased the stability and reliability of the navigation behavior.

This experience highlighted how system-level performance depends on every layer functioning correctly, from sensing and filtering to control and actuation. It also emphasized the importance of clear task division and communication within the team, as progress in one area often depended on assumptions made in another. Overall, this project reinforced the idea that successful mechatronic systems are built not only on correct individual components, but on careful integration and verification of the complete system.

A Appendix

A.1 Source Code

All source code developed for this project was submitted on Gradescope and is publicly available on GitHub:

https://github.com/DasVade/ROBA-final-code

The repository includes:

- Firmware for the master ESP32 Feather, including joystick control, wall-following, Vive-based navigation, and autonomous attack routines.
- Firmware for the slave ESP32-C3 responsible for sensor acquisition and UART communication.
- Python driver station code used on the laptop for UDP-based control and mode selection.

The code is organized by functionality and includes in-line comments to facilitate understanding and replication by other students.

A.2 Bill of Materials (BOM)

Table 1 lists the major components used in the construction of the robot. Small hardware items such as screws, nuts, resistors, and jumper wires are omitted as instructed.

Component	Description	Quantity
ESP32 Feather	Master microcontroller	1
ESP32-C3	Slave microcontroller	1
DC Gear Motors	Drive motors	4
Motor Driver Board	Dual-channel motor control	1
Servo Motor	attack bar	1
Vive Trackers	Localization system	2
Time-of-Flight Sensors	Distance sensing	1
Ultrasonic Sensors	Obstacle detection	2
Rechargeable Battery	Power supply	2
Top Hat Mechanism	Game interaction component	1

Table 1: Bill of Materials

Figure. 5 lists the major components purchased in the construction of the robot.

1	Part Description	Quantity	Distributor	Link
2	Battery	1	Sparkfun	Lithium Ion Batt
3	9V-5V	1	Adafruit	UBEC DC/DC S
4	9V-3.3V	1	Adafruit	MPM3610 3.3V
5	Motor	4	Amazon	Amazon.com: H
6	Motor driver	3	Adafruit	Adafruit DRV88
7	Encoder	2	/	
8	Servo	1	Adafruit	Micro Servo - Hi
9	Wheel	1	Adafruit	Orange and Cle
10	Switch (bumper)	1	Adafruit	Micro Switch w/
11	IMU	1	/	
12	Ultrasonic	1	/	Ultrasonic Dista

Figure 5: Purchase BOM

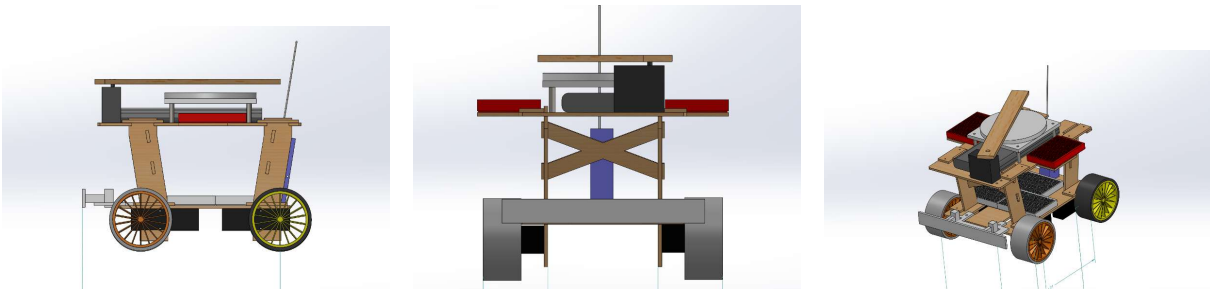
A.3 Mechanical Drawings

Measured CAD drawings of all laser-cut structural components, including the chassis, top plate, and support members, are included here. These drawings provide sufficient dimensional detail for another student to replicate the mechanical structure of the robot.

A.4 Videos and Demonstrations

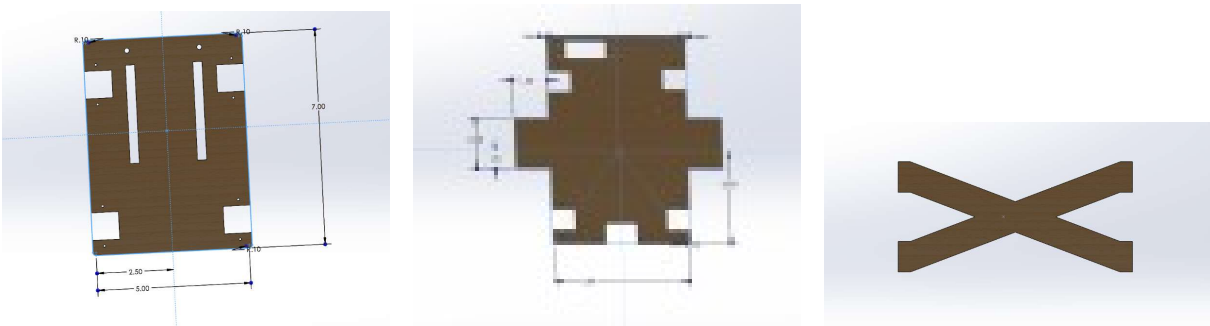
Videos demonstrating the robot's functionality, including unearned autonomous attack routines, are available at the following links:

- autonomous attacking nexus 4 times in competition: https://www.youtube.com/watch?v=f1X58f-0aBQ



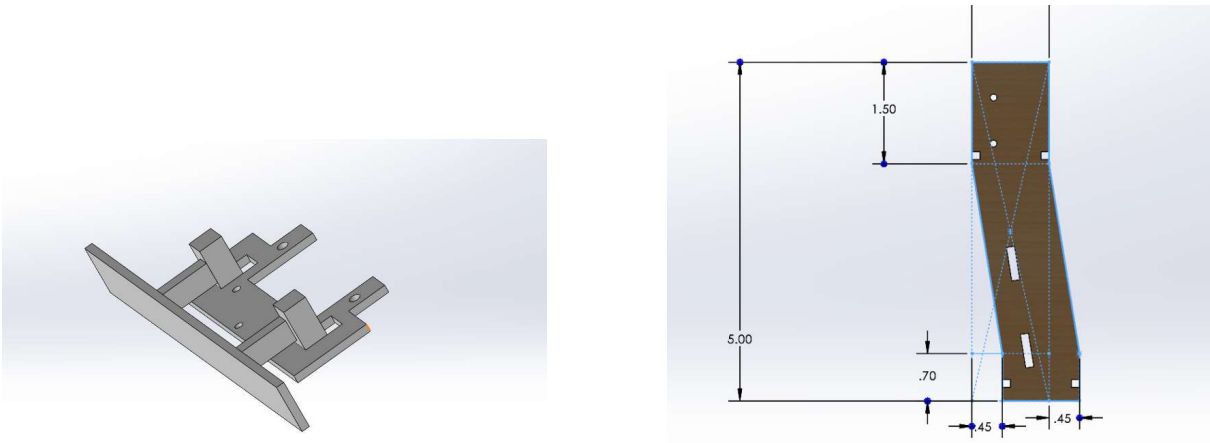
(a) view1 (b) view2 (c) view3

Figure 6: CAD drawings#1



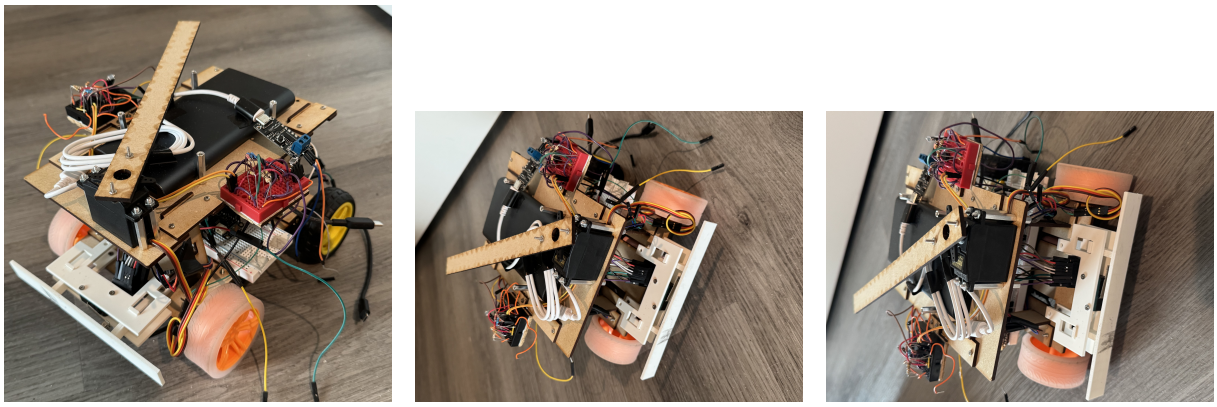
(a) chasis (b) top plate (c) corss support

Figure 7: CAD drawings#2



(a) bumper (b) vertical support

Figure 8: CAD drawings#3



(a) view1 (b) view2 (c) view3

Figure 9: real photos

- autonomous attacking high tower not in competition: https://youtube.com/shorts/K8pAkWYsFU8?si=YWldAMRNQtmTGRQT

All videos are publicly accessible for TA evaluation.(same video in section 5.3)