Mechatronics Project Design Report
Team Eggcelsior

MCEN 4115/5115 Mechatronics

Abstract

The goal of this project was to create a robot that could compete alongside other teams in a Mario Kart Balloon Battle-style game. Robots must be subject to specific regulations to keep matches fair. This included: robots must be fully autonomous, robots must be battery powered, team expenditure must be below $200, robots must have a “crown” of 3 balloons equidistant from each other, and the robot must fit into a 14” x 12” footprint.

A successful robot would be able to navigate the field and use a lance and/or a shooter to pop 4 balloons on either the opposing robot or the team’s bases. To accomplish this our robot was designed with a lance as the main weapon and a disk shooter as a strong distance weapon. The robot also has many sensors for navigation including an IMU, 2 ultrasonic sensors, 4 line following sensors, and a PixyCam for balloon recognition. The chosen method for movement was omnidirectional mecanum wheels that enabled the robot to move diagonally and strafe.

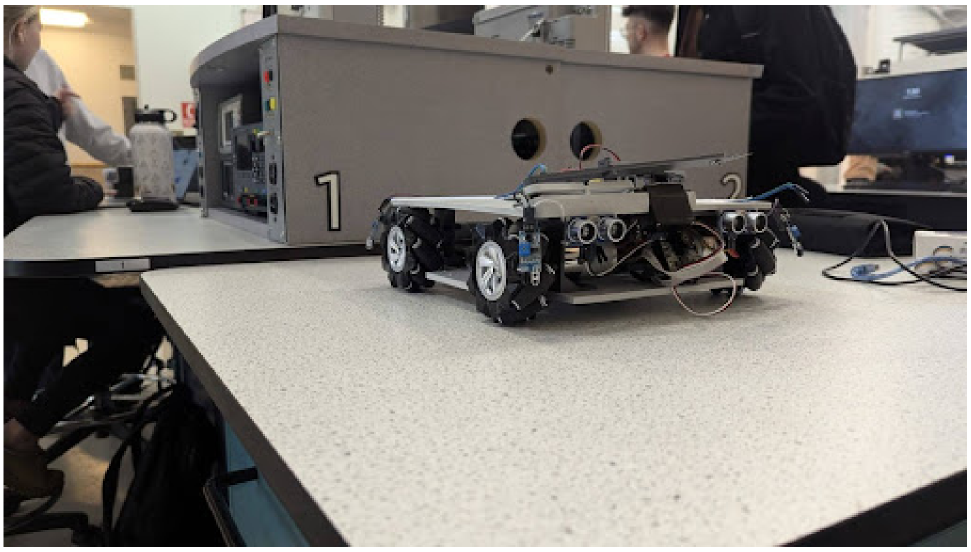


Figure 1: Robot

Contents

1 Introduction

The objective of this project was to design and build an autonomous robot capable of competing in a balloon-popping game, a dynamic and interactive challenge that required precision, adaptability, and efficient execution. The project posed several technical and design constraints that guided the development process. These constraints included:

- Autonomy: The robot needed to operate entirely without human intervention, relying on onboard sensors, control algorithms, and real-time decision-making.
- Power Source: The robot had to be powered exclusively by onboard batteries, ensuring sufficient energy supply for motors, sensors, and computing systems while maintaining operational efficiency.
- Size Constraints: The robot's footprint was limited to 14 inches by 12 inches to ensure mobility and compatibility with the competitive arena.
- Budget Limit: The total cost of the robot could not exceed $200, requiring careful selection of components and materials to maximize functionality within financial limits.
- Balloon Crown: Each robot was required to carry a "crown" consisting of three balloons mounted equidistantly. These balloons served as the primary targets for opponents and needed to be integrated securely into the robot's design.

1.1 Robot Design Features

To meet these objectives, the robot incorporated innovative mechanical, electronic, and software systems designed to excel in the game. The key design features included:

- Omnidirectional Mecanum Wheels:
  - The robot used a four-wheel mecanum drive system, providing omnidirectional movement. This allowed the robot to strafe, rotate, and move diagonally with precision, offering a significant advantage in maneuvering within the confined game arena.
  - The wheels were powered by four 300 RPM gearmotors, ensuring a balance between speed and torque for quick and responsive navigation.
- Lance for Close-Range Attacks:
  - A lance mechanism was developed as the robot's primary weapon for close-range balloon-popping. The lance used a scissor-like extension system powered by a servo motor, enabling rapid deployment.
  - The tip of the lance was equipped with an X-Acto blade, ensuring reliability in puncturing balloons with minimal force and precise targeting.
- Disk Shooter for Long-Range Strikes:
  - To complement the lance, a high-speed disk shooter was incorporated for long-range attacks. The shooter utilized a 74,000 RPM flywheel motor, salvaged from a custom Nerf gun, to launch lightweight disks with high velocity.

  - The disks were coated with limonene to increase their effectiveness in bursting balloons. A solenoid-based loader mechanism allowed automated reloading, enabling continuous operation during gameplay.

- Compact and Modular Chassis:
  - The robot's chassis was designed for modularity and ease of assembly. The lower section housed motors, batteries, and electronics, while the upper section provided mounting points for sensors and the weapon systems.
  - The 3D-printed chassis parts were lightweight yet robust, ensuring structural integrity while maintaining compliance with the size constraints.

By integrating these systems into a cohesive design, the robot was well-equipped to navigate the game environment, detect targets, and engage effectively in competitive scenarios. The combination of advanced mobility, versatile weaponry, and autonomous decision-making allowed the robot to meet its objectives within the defined constraints.

2 System Overview

The robot's design was carefully structured into three major subsystems: Mechanical Components, Electronics and Sensors, and Software and Control Algorithms. Each subsystem was developed independently to maximize functionality and reliability while ensuring seamless integration into a cohesive and autonomous robot capable of competing in the balloon-popping game.

- Mechanical Components The mechanical subsystem formed the structural foundation of the robot. It included:
  - Chassis: A lightweight, 3D-printed, modular frame that provided a robust yet compact structure. The chassis was designed to securely house motors, sensors, and weapon systems while adhering to the 14" x 12" size constraint.
  - Four omnidirectional mecanum wheels enabled advanced mobility, allowing the robot to strafe, rotate, and maneuver diagonally. This flexibility was essential for navigating tight spaces and tracking moving targets effectively.
  - Weapon Systems:
    - A lance mechanism for close-range balloon-popping, designed with a scissor-style extension powered by a servo motor.
    - A disk shooter for long-range attacks, equipped with a high-speed flywheel motor capable of launching lightweight disks with precision and speed.

  The mechanical components were strategically mounted on the chassis to ensure balanced weight distribution, stability, and efficient use of space. The modularity of the design allowed for easy assembly, maintenance, and future upgrades.

- Electronics and Sensors
  - Microcontrollers:
    - A Raspberry Pi 4 acted as the central processing unit, executing high-level control algorithms and managing data from the sensors.

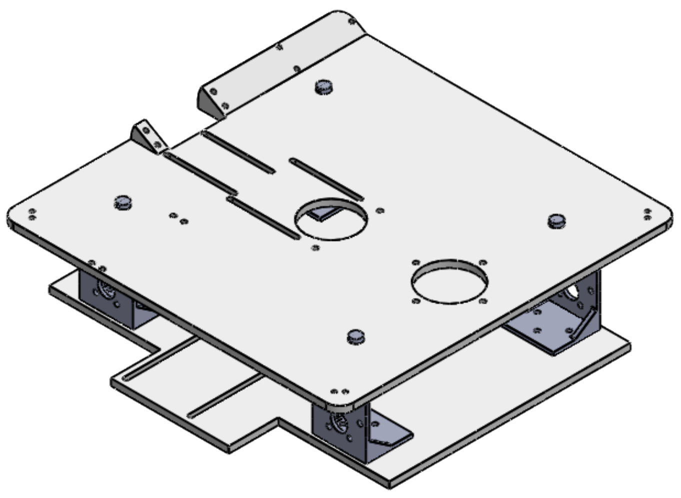
        * An Arduino Mega controlled the motor drivers and weapon systems, while an Arduino Uno managed sensor data acquisition.
    - Sensors:
        * Ultrasonic Sensors for obstacle detection, providing distance measurements to avoid collisions with walls or other robots.
        * Infrared (IR) Sensors to detect boundaries and ensure the robot remained within the designated game area.
        * A PixyCam for balloon recognition, enabling the robot to identify and track balloon targets based on color.
    - Power System: A 12V LiPo battery powered the motors and solenoids, while a USB power bank supplied the Raspberry Pi. Voltage regulators ensured stable power distribution to all components.

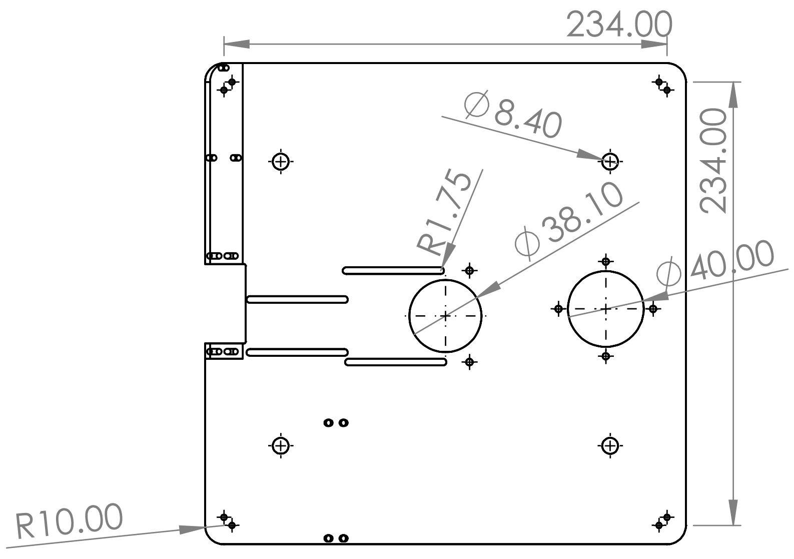
  The electronic subsystem was designed with careful attention to wiring and layout to minimize electromagnetic interference and ensure reliable communication between components.

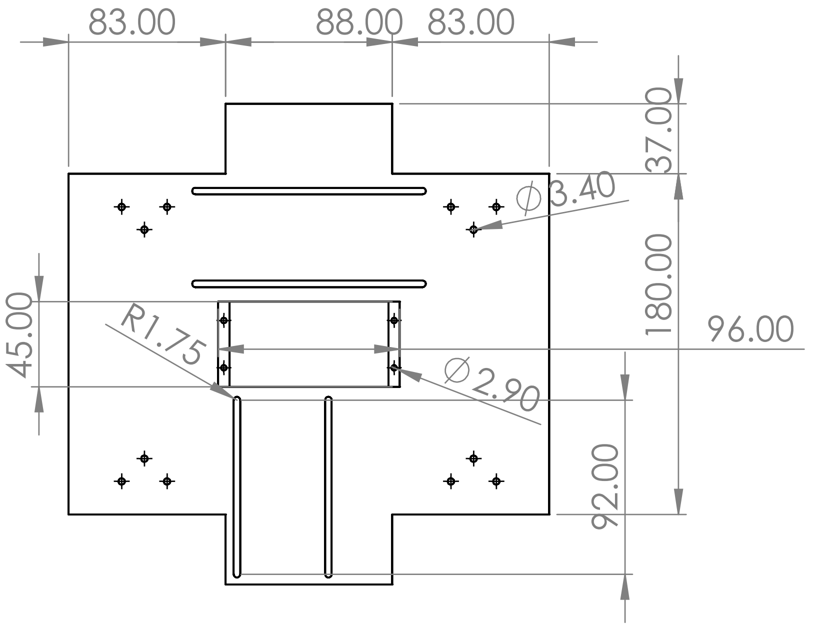
- Software and Control Algorithms The software subsystem acted as the brain of the robot, orchestrating the actions of the mechanical and electronic subsystems to achieve autonomous operation. Key aspects included:
    - Control Logic: High-level algorithms running on the Raspberry Pi managed navigation, obstacle avoidance, and weapon activation based on sensor inputs.
    - Sensor Integration: Data from the PixyCam, ultrasonic sensors, and IR sensors were fused to create an accurate perception of the robot's surroundings, enabling informed decision-making.
    - Motion Control: Algorithms for omnidirectional movement ensured smooth and precise maneuvers, leveraging the capabilities of the mecanum wheels.
    - Serial Communication: A robust protocol facilitated real-time data exchange between the Raspberry Pi and the Arduino boards, ensuring synchronization of sensor readings, motor commands, and weapon actions.

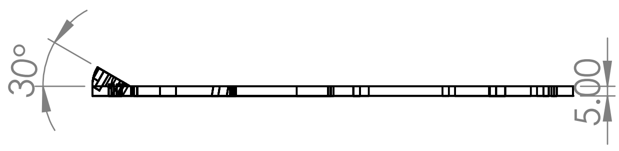
The integration of these three subsystems was a critical aspect of the robot's design. The mechanical components provided a stable and functional platform, the electronics facilitated perception and actuation, and the software ensured intelligent and coordinated operation. Rigorous testing and debugging were conducted to ensure that each subsystem interacted seamlessly, resulting in a fully autonomous robot capable of meeting the demands of the competition.

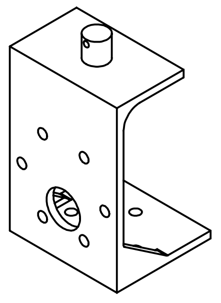
By effectively combining these subsystems, the robot was able to navigate the game environment, detect and target balloons, and engage them with precision and reliability.

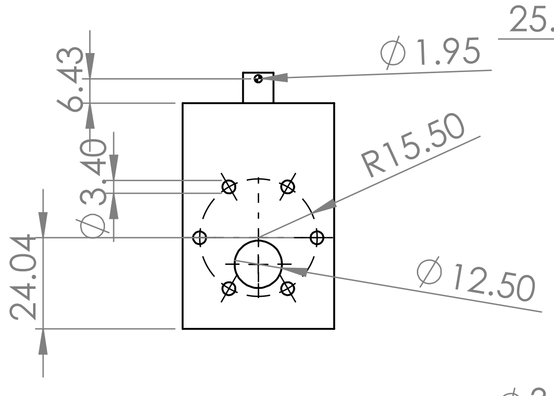


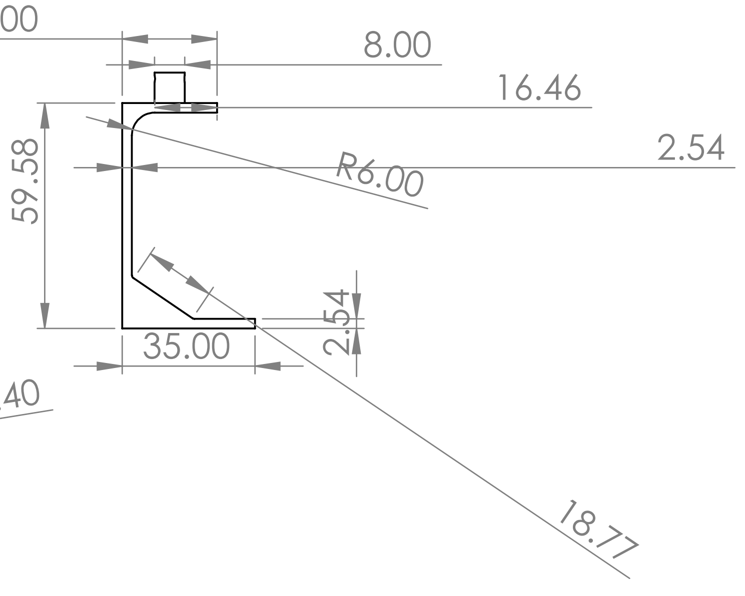

234.00
Ø8.40
R1.75
Ø38.10
234.00
Ø40.00
R10.00

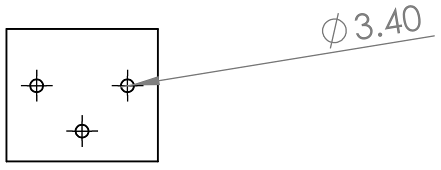

83.00
88.00
83.00
37.00
Ø3.40
180.00
45.00
R1.75
96.00
Ø2.90
92.00


30°
5.00




6.43
Ø1.95
Ø3.40
R15.50
24.04
Ø12.50


25.00
8.00
16.46
2.54
59.58
R6.00
2.54
35.00
18.77


Ø3.40

3 Mechanical Components

3.1 Chassis

The chassis is responsible for holding all subcomponents of the robot, so the design is heavily mount-conscious. Ease of assembly and 3D printing were at the forefront of the design considerations so the body was split into a top and bottom. The bottom is where most of the electronics and motors were housed. This was a flat plate with mounting points for motor brackets the Arduino and Raspberry Pi.

The top plate of the robot was used to house more easily accessible components like the battery and the IMU which needed to be farther away from the motors to not pick up any interference. The top plate also featured passthrough holes for the motor mounts so that body clips could be used to secure the top plate to the rest of the body, but also allow for easy removal and access to the lower level.

3.2 Drive System

For the robot's movement, it was decided to use a classic 4-wheel design for simplicity. From here, mecanum wheels were chosen over regular wheels to allow for advanced movement. This includes moving diagonally and strang sideways

The above figure shows the movement patterns of the mecanum wheels, illustrating their ability to translate in any direction through combinations of independent wheel rotation. Being able to rotate wheels independently was crucial as it meant we could rotate about the center of the robot rather than limiting ourselves to turning over a wide radius. Though this could have been achieved with standard wheels using traditional tank steering, having the flexibility to simultaneously translate sideways allowed us to track a target by rotating while moving about in any direction.

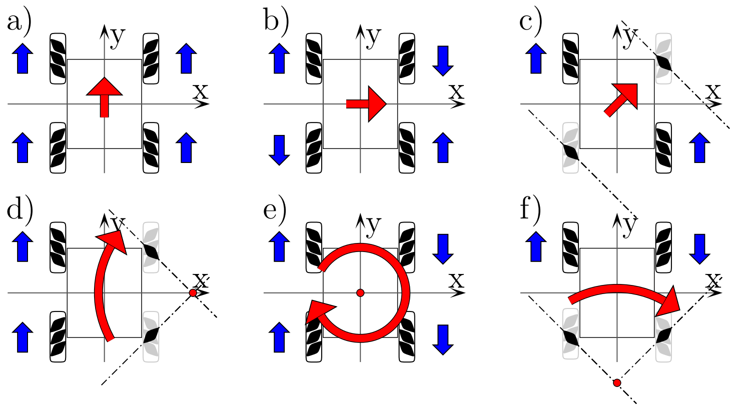


Figure 2: `https://en.wikipedia.org/wiki/Mecanum_wheel`

3.2.1 Motors and Controls

Having originally been provided with a set of four 60RPM DC gearmotors, we opted to upgrade to 300 RPM motors as it simplified the implementation of our drive system at the speed we desired. Put simply, in order to take full advantage of the maximum motor speed when using mecanum wheels to move in one direction, you must sacrifice your ability to move in another direction simultaneously. For example, if you are moving forward at full speed, you can not simultaneously rotate at that speed because all 4 wheels are already spinning at their highest RPM.

The simplest way to manage this was to design the drive system in such a way in which the speed of each mode of motion (forwards, sideways and rotational) was bound by a third of the total motor speed. The drive system incorporated one function for each mode of motion, which independently added or subtracted to the speed of each wheel depending on the motion. Simply put, the speed of the four wheels was controlled by a linear combination of the three motion patterns which govern each mode of motion (a,b and e in the above figure), in such a way that the maximum motor speed would only be achieved if all three modes of motion were maximized simultaneously.

Physically, the drive system was controlled by an Arduino, and its overall footprint was greatly reduced through the use of a protoboard mounted above the arduino, housing two motor drivers, connections to each motor pin and to the battery.

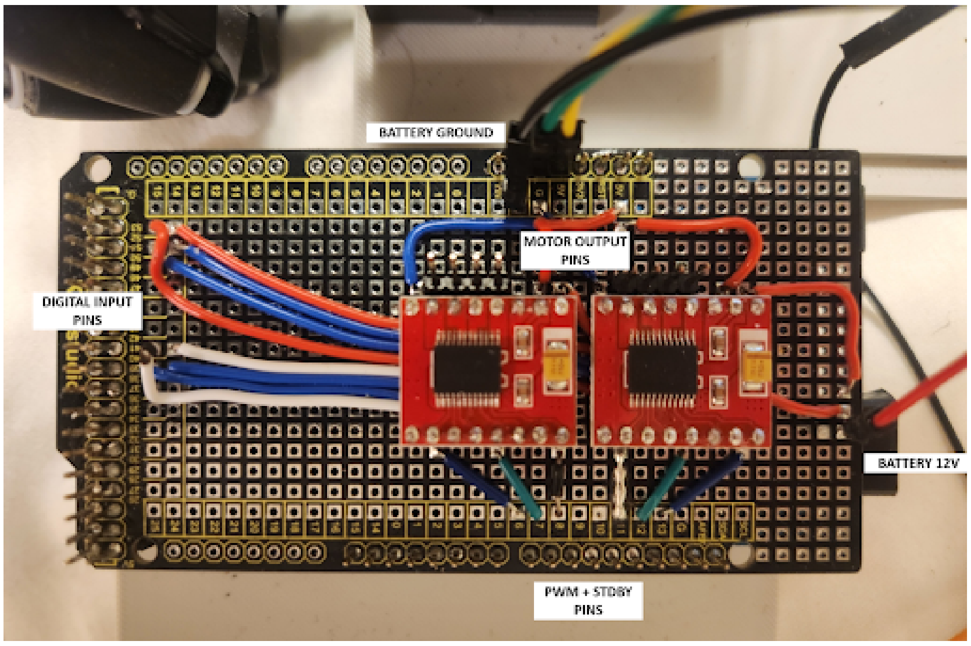


Figure 3: Proto board for motor control

3.3 Ballon Pooping Mechanisms

we have devised two Mechanisms for pooping the baloon one links based actuator that fixed with Xacto knife (Lance) and other a shooting mechanism(Disk Shooter)

3.3.1 Lance

The lance was decided to be our main weapon to be used against balloons because it is more reliable, consistent, and easier to integrate than the shooter. The design used a simple scissor mechanism that relied on a single servo to extend the lance along a rail. The idea behind this design was to strike a balance between speed and control of extension, complexity, footprint, and durability. This mechanism proved to be very easy to design and implement into the design of the robot, due to its small form factor. An Xacto blade was used at the end of the lance, which provided a reliable way to pop a balloon with minimal contact

3.3.2 Disk Shooter

The shooting mechanism was designed around the idea of using limonene to pop balloons. Limonene is a nonpolar hydrocarbon extracted from the peels of oranges that is capable of penetrating the thin latex of balloons. In order to incorporate this idea into the design, disks were developed to be extremely lightweight and feature a large groove on the outside that a bicycle front fork o-ring could slip onto. This o-ring is designed to be oil absorbent and held the limonene liquid inside, but was exposed enough so that if the disk impacted a balloon, it would release the popping agent.
The firing mechanism used a motor salvaged from a custom Nerf gun, that at maximum speed could reach 74,000 rpm. This was mounted on an adjustable motor mount and used a tapered flywheel with rubber o-ring, to grip onto disks and send them down a modified rail.
The last piece of the disk shooter was the loading mechanism. A solenoid was mounted to the back of the shooter that would actuate an arm forcing a pusher to send a disk into the flywheel. The pusher would restrict another disk from dropping down until it fully retracted, upon which when it did fully retract a new disk would be gravity-fed into the chamber ready for firing.
To power and run the solenoid and motor, 2 relay circuits were made to take in a 5V logic from an Arduino and direct the flow of the 12V lipo battery power to or away from the flywheel and solenoid.

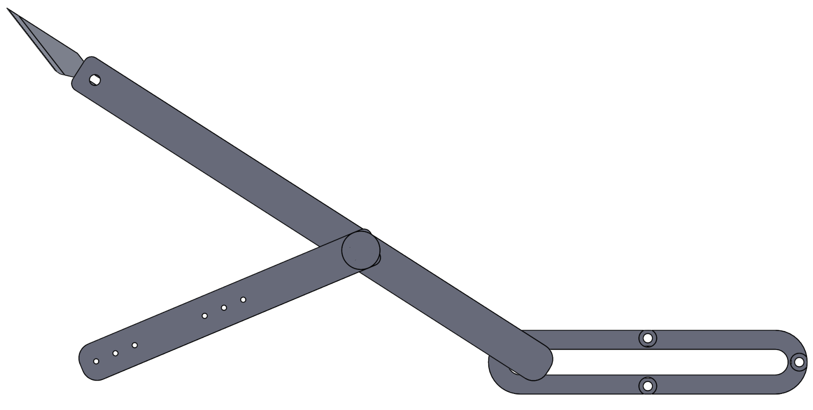
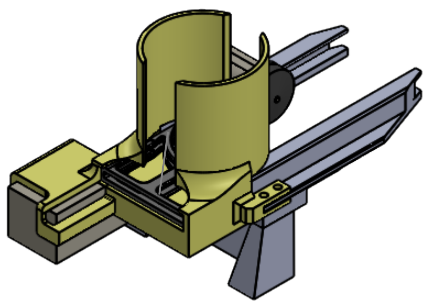
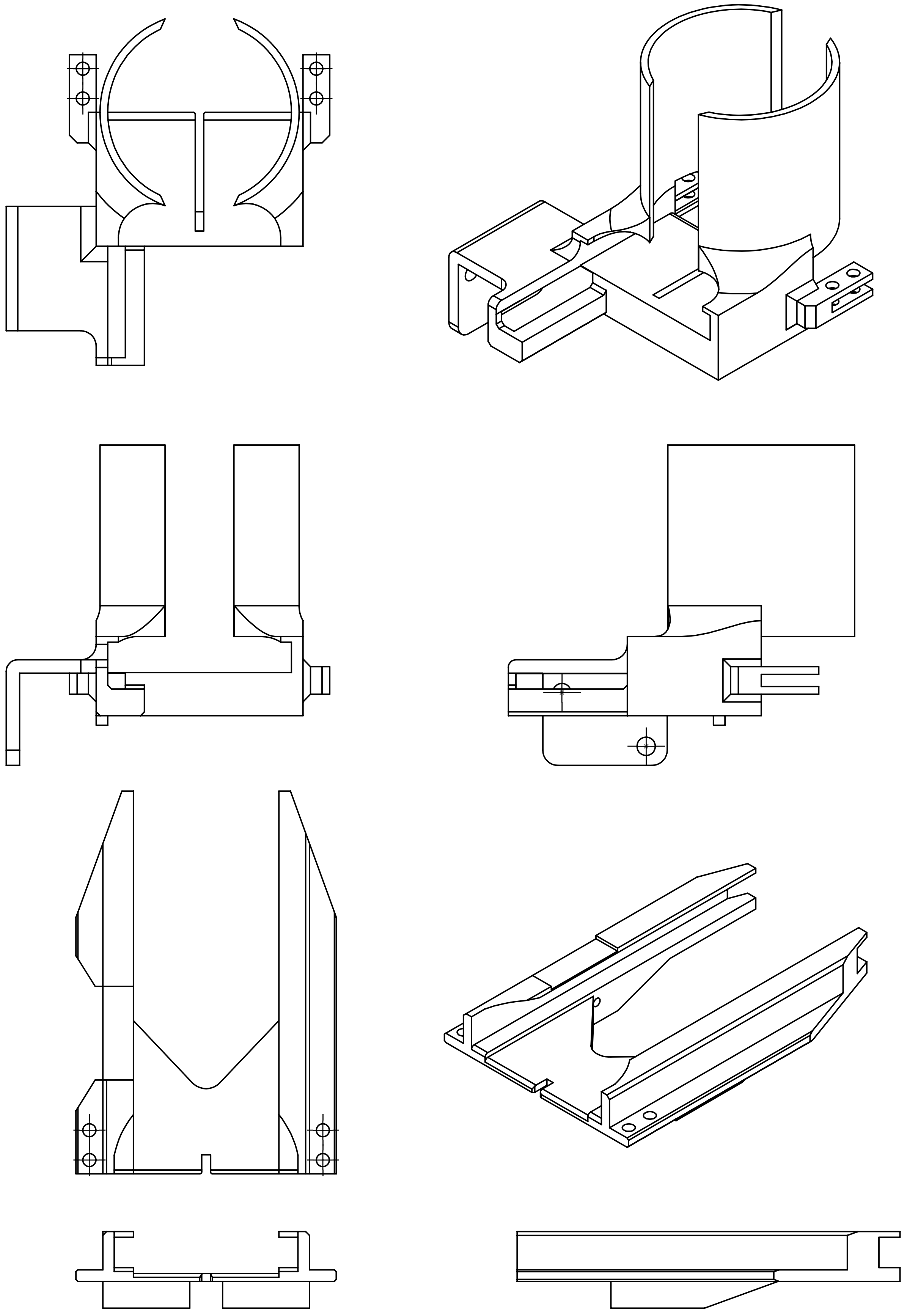
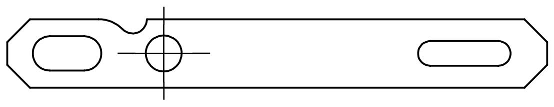
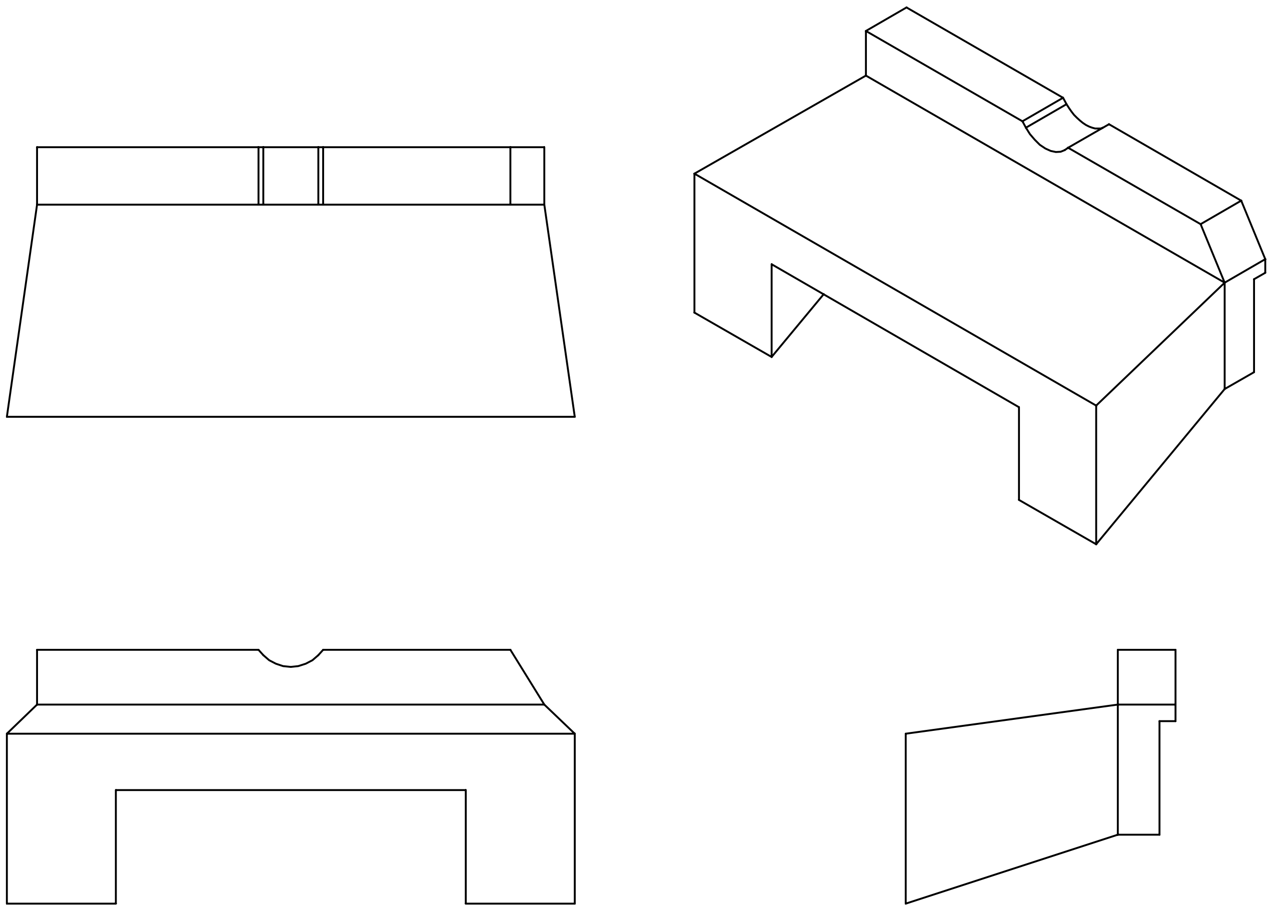
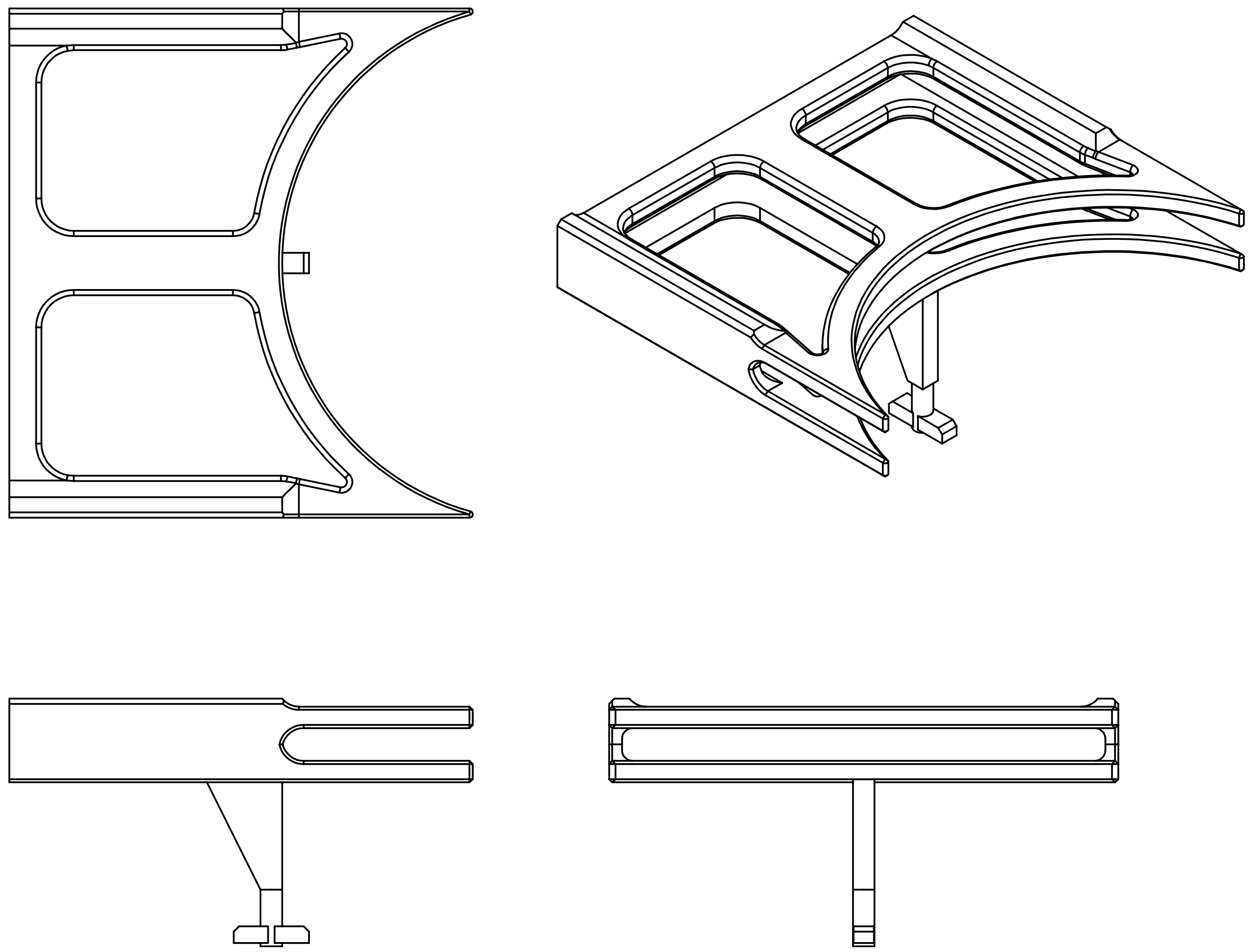


Figure 4: lance





4 Electronics and Sensors

4.1 Microcontrollers and Control Boards

The robot's primary logic system was managed by a Raspberry Pi, while an Arduino Mega controlled the motors and weapon systems. The Arduino Uno was used for sensor integration and communication.

4.2 Sensors

The robot was equipped with:

- Four infrared sensors for obstacle detection.
- Two ultrasonic sensors for proximity measurements.
- A PixyCam for balloon recognition.

Four infrared sensors which were placed on each of the four corners of the chassis. This allowed us to precisely determine when the robot was approaching a wall or arena bound line, which were marked with reflective white tape. Based on which sensors were triggered, It could be determined which side of the robot was approaching an obstacle and adjust accordingly.

Additionally, two distance sensors were used, each of which was placed at the front of the robot. This choice was made on the assumption that the robot would be primarily moving forward, and that the sensors would provide additional data which could be used both as a supplement and contingency for our other sensors. For example, one use case was avoiding approaching the edge of the wall in such a way that the IR sensors would straddle the marking tape on either side. The distance sensors would provide us with an alternative way to detect this obstacle. They would also allow us to detect the proximity of other robots. Having two allowed us to compare their readings to determine which side of the robot was closest to an obstacle.

4.3 Power System

The robot used a 12V LiPo battery to power the motors and solenoids, while a USB power bank supplied the Raspberry Pi.

5 Software and Control Algorithms

5.1 Control Logic

The Raspberry Pi handled the overall control logic, processing sensor inputs and transmitting commands to the Arduino. Key algorithms included:

- Obstacle avoidance using infrared and ultrasonic sensors.
- Balloon tracking using PixyCam.
- Targeting and weapon activation based on balloon detection.

The control logic of the robot is implemented in Python, where various sensor data is read, decoded, and then used to make decisions about the robot's movement and behavior. The logic includes functions for handling sensor data (e.g., pan, tilt, and tap states) and performing tasks such as collision detection and response.

5.1.1 Sensor Data Decoding

The sensor data is received as a single byte (`pan_and_tap`) from the Arduino, which is then decoded into multiple individual components. The relevant bits in the byte represent different sensor states such as pan, tilt, and tap positions. The following function `decode_values` extracts and interprets the sensor information:

```
def decode_values(pan_and_tap):
    data_array = [0, 0, 0, 0]
    pan = pan_and_tap & 0xF0
    tilt = (pan >> 6) & 0x01
    pan = pan & 0xBF
    tap = pan_and_tap & 0x0F

    if pan == 0x80:
        print("No object detected")
    elif pan == 0x00:
        print("Zero offset")
    elif pan == 0x30:
        print("Object on right")
    elif pan == 0x10:
        print("Object on left")
    else:
        print("Invalid")

    if tilt:
        print("Poke")

    tap_RR = tap & 0x01
    tap_LR = (tap >> 1) & 0x01
    tap_RF = (tap >> 2) & 0x01
    tap_LF = (tap >> 3) & 0x01

    if tap_RR:
        print("Right rear tap detected")
    if tap_LR:
        print("Left rear tap detected")
    if tap_RF:
        print("Right front tap detected")
    if tap_LF:
        print("Left front tap detected")

    return pan, tilt, tap_LF, tap_LR, tap_RF, tap_RR
```

5.1.2 Collision Detection

The robot uses a combination of `tap` sensors and distance sensors to detect obstacles in front and behind it. The function `if_frontcollision` checks for a front collision based on sensor values. If a collision is detected, the robot will attempt to maneuver to avoid it. The logic for rear collision detection is handled similarly in the `if_rearcollision` function.

```
def if_frontcollision(tap_LF, tap_LR, tap_RF, tap_RR, dist_L, dist_R):
    if tap_LF or tap_RF:
        return 1
    if dist_L > 0 and dist_L < 40:
        return 1
    if dist_R > 0 and dist_R < 40:
        return 1
    return 0

def if_rearcollision(tap_LF, tap_LR, tap_RF, tap_RR, dist_L, dist_R):
    if tap_LR or tap_RR:
        return 1
    return 0
```

5.1.3 Movement Logic

Based on the sensor data, the robot decides on actions such as rotating, moving forward, or avoiding obstacles. The following pseudo code illustrates the decision-making process:

- **Pan and Tilt Sensor Input:** If no object is detected (`pan == 0x80`), the robot stops or moves in a default direction. If the object is detected to the left (`pan == 0x10`) or right (`pan == 0x30`), it adjusts the movement accordingly.
- **Collision Handling:** If the front or rear collision functions return `1`, the robot adjusts its movement. Random directions are chosen to avoid obstacles.
- **Tilt Sensor Input:** If the tilt sensor is active, the robot performs an action such as "poking" a balloon, based on the tilt status.

The robot's movement is determined by a state machine that reacts to sensor input:

```
# Pseudo Code for Movement Logic
if front_collision == 1:
    # Move backward or to a random direction
    if dist_L >= dist_R:
        move_left()
    elif dist_R > dist_L:
        move_right()
    else:
        random_direction()

if rear_collision == 1:
    # Move forward or adjust direction
    if dist_L < dist_R:
        move_forward()
    else:
        move_backward()

# Regular movement based on pan data
if pan == 0x10:
    move_left()
elif pan == 0x30:
    move_right()
else:
    stop_moving()
```

5.1.4 Data Transmission

The robot sends movement commands through the `SerialCommunication` class. After decoding the sensor values and processing the movement logic, the data array is sent to the microcontroller to control the robot's motors.

```
ser_mega.send_data_to_mega(data_array)
ser_mega.reset_input_buffer()
```

5.2 Full Pseudo Code

Here is a full overview of the control logic in pseudo code:

- Read sensor data (`pan_and_tap`, `dist_L`, `dist_R`).
- Decode the sensor values to extract relevant states.
- Check for front or rear collision using `if_frontcollision` and `if_rearcollision`.
- Based on the decoded values, determine movement commands (e.g., left, right, forward, backward).
- Send the movement commands to the motor controllers.
- Repeat the process in a loop.

```
while True:
    pan_and_tap, dist_L, dist_R = read_data()
    pan, tilt, tap_LF, tap_LR, tap_RF, tap_RR = decode_values(pan_and_tap)

    if front_collision == 1:
        handle_front_collision()
    if rear_collision == 1:
        handle_rear_collision()

    # Movement based on pan
    if pan == 0x10:
        move_left()
    elif pan == 0x30:
        move_right()

    ser_mega.send_data_to_mega(data_array)
```

5.3 Communication

The Raspberry Pi communicated with the Arduino using a serial interface, sending motion commands and receiving sensor data. A three-byte encoding scheme was used for efficient data transfer between components.

To efficiently represent the balloon's state, IR sensor activations, and distance sensor readings, the Arduino Uno firmware encodes the data into three bytes. The structure is designed to maximize information density while maintaining clarity. Below is a detailed breakdown:

Byte Structure

- Byte 1: Balloon State and IR Sensor States
- Bits 7-4: Balloon state information:
- Bit 7: Balloon detected (0 = detected, 1 = not detected).
- Bit 6: Balloon in range (1 = in range, 0 = out of range).
- Bit 5: Balloon offset direction (1 = left, 0 = right).
- Bit 4: Balloon offset state (1 = centered, 0 = offset).
- Bits 3-0: IR sensor states: Each bit corresponds to a binary state (1 = active, 0 = inactive) of up to 4 IR sensors.
- Byte 2: Distance Sensor 1
- Bits 7-0: Scaled value of the first distance sensor (8-bit resolution).
- Byte 3: Distance Sensor 2
- Bits 7-0: Scaled value of the second distance sensor (8-bit resolution).

Encoding Logic

1. Balloon State Packing: Balloon status is represented by assigning specific bits for detection, range, offset direction, and offset state.
For example:
$balloon\_status = (detected \ll 7)|(in\_range \ll 6)|(offset\_dir \ll 5)|(offset\_state \ll 4);$

2. IR Sensor Packing: IR sensor states are combined into the lower 4 bits of Byte 1:
$tap\_status = (tap1 \ll 3)|(tap2 \ll 2)|(tap3 \ll 1)|tap4; Byte1 = balloon\_status|tap\_status;$

3. Distance Sensors: Sensor readings are scaled and directly stored in Bytes 2 and 3.

6 Integration

The integration of all subsystems was a critical step in ensuring that the robot functioned as a cohesive and autonomous unit. This involved the seamless assembly of mechanical, electronic, and software components into a single operational system, with rigorous testing to validate functionality.

6.1 Mechanical Integration

The mechanical components, including the chassis, wheels, and actuators, were meticulously designed and fabricated to provide structural support and mounting points for all other subsystems. The chassis was divided into two tiers:

- Lower Tier: Housed the drive motors, motor brackets, and control boards. The bottom plate was equipped with precise mounting holes to ensure alignment and secure attachment
- Upper Tier: Hosted the IMU, battery, and weapon mechanisms, ensuring that sensitive components were isolated from vibrations and electromagnetic interference from the motors.

6.2 Electronic Integration

The electronic components, including sensors, microcontrollers, motor drivers, and the power supply, were interconnected through well-organized wiring. Key aspects of the integration process included:

- Sensor Placement: Ultrasonic and IR sensors were strategically mounted on the chassis for optimal obstacle detection and navigation. Care was taken to avoid occlusion and interference between sensors.
- Wiring Management: Cable routes were planned to minimize clutter and prevent accidental disconnections. Heat-shrink tubing and zip ties were used to secure wires in place.
- Power Distribution: The 12V LiPo battery powered the motors and solenoids, while the USB power bank supplied the Raspberry Pi. Voltage regulators ensured safe and consistent power delivery to all components.

6.3 Software Integration

The software was developed to serve as the ”brain” of the robot, orchestrating the actions of the mechanical and electronic subsystems. The integration process involved:

6.3.1 Uploading Software

- The high-level control logic was uploaded to the Raspberry Pi, enabling it to act as the central processing unit.
- Sensor-specific and actuator-control routines were implemented on the Arduino boards, which handled tasks like reading sensor data, controlling the motor drivers, and managing the weapon system.

6.3.2 Communication Between Microcontrollers

- A serial communication protocol was established between the Raspberry Pi and the Arduino boards to synchronize actions. Sensor data from the Arduino was transmitted to the Raspberry Pi, where decisions were made, and corresponding commands were sent back.

6.4 Testing and Debugging

After assembling the subsystems, extensive testing was conducted to validate the integration:

- Subsystem Functionality: Each subsystem (mechanical, electronics, and software) was tested independently to confirm functionality before integration.

- End-to-End Testing: The fully integrated robot was tested in simulated environments to ensure proper interaction between all subsystems. For example: Movement commands from the Raspberry Pi were verified to control the motors correctly.

  - Movement commands from the Raspberry Pi were verified to control the motors correctly.
  - Sensor data, such as obstacle distances and balloon detections, were accurately processed to trigger the appropriate responses (e.g., weapon activation or avoidance maneuvers).

6.5 Challenges and Solutions

During integration, several challenges were encountered:

- Electromagnetic Interference: Noise from the motors affected sensor readings. This was mitigated by isolating sensitive components and adding filtering capacitors to the circuit.

- Power Supply Stability: Voltage fluctuations during high-current operations caused intermittent resets. Voltage regulators and capacitors were added to stabilize the power supply.

- Software Debugging: Serial communication timing issues led to delays in response. The software was optimized to include error-checking and efficient data encoding.

By addressing these challenges and ensuring tight integration between mechanical, electronic, and software components, the robot was able to operate autonomously and reliably in its intended environment.

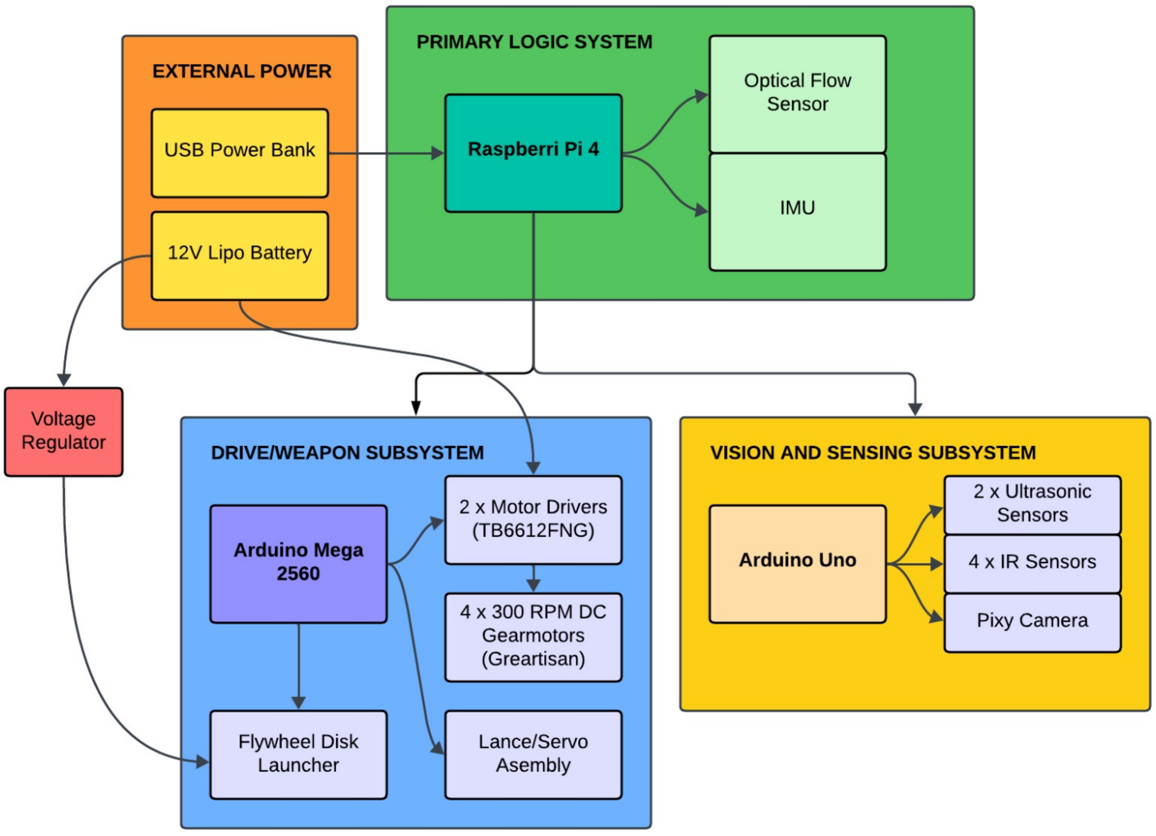


Figure 5: Integration

7 Future Improvements

While the robot successfully met its design objectives, several enhancements could further improve its performance and reliability. These improvements span sensor calibration, power management, algorithm development, and material selection.

Enhanced Sensor Calibration for More Accurate Obstacle Detection The robot's obstacle detection relied on ultrasonic and infrared (IR) sensors. While effective in many scenarios, their performance was sometimes hindered by environmental factors:

- Environmental Noise: Ultrasonic sensors occasionally misinterpreted reflections from nearby surfaces or other robots as obstacles. Similarly, IR sensors were susceptible to interference from ambient light sources, reducing their accuracy.

- Mounting Position: The sensor placement on the robot led to partial occlusions in certain orientations, creating blind spots.

To address these limitations, future improvements could include:

- Dynamic Calibration: Implementing real-time calibration algorithms that adjust sensor thresholds based on environmental feedback.

- Sensor Fusion: Combining data from multiple sensors (e.g., ultrasonic and IR) to cross-validate obstacle positions and enhance detection reliability.

- Improved Mounting: Adjusting sensor mounting angles and positions to ensure unobstructed fields of view and better coverage around the robot.

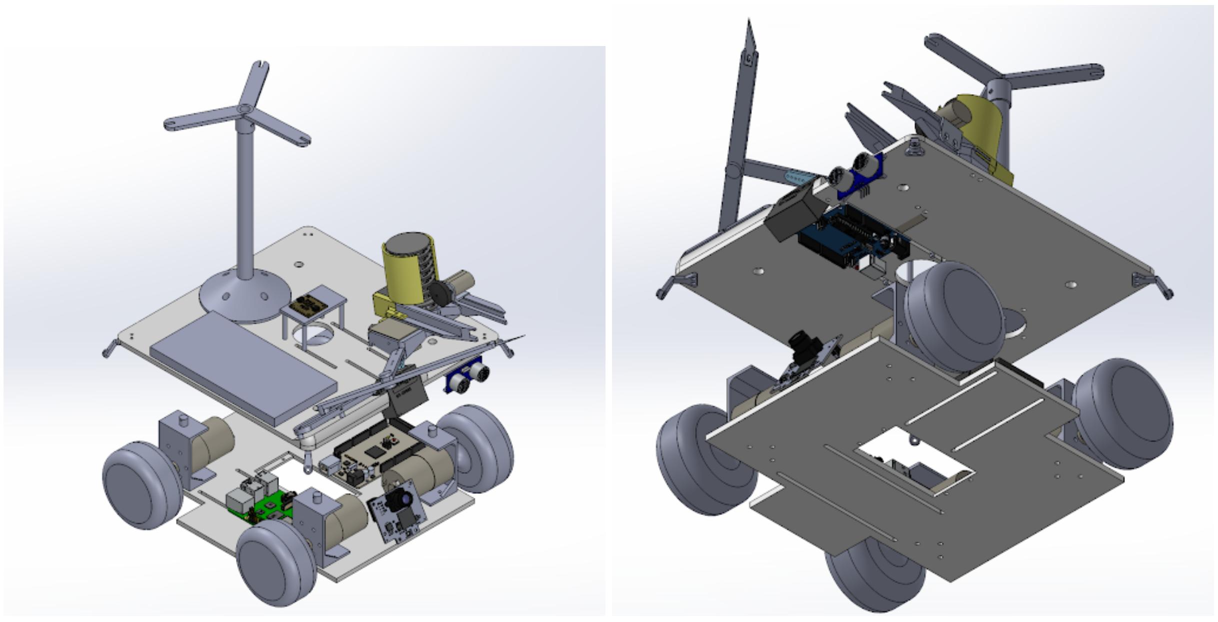


Figure 6: Chassis Design

7.1 Optimization of the Power System for Longer Operation Times

The robot's power system, while functional, presented limitations in extended operational scenarios. The 12V LiPo battery effectively powered the motors and solenoids, but voltage drops during high-current draws occasionally impacted performance. Additionally, the USB power bank for the Raspberry Pi limited runtime. Future optimizations could include:

- High-Capacity Batteries: Using higher-capacity LiPo batteries to extend operation times without increasing the robot's footprint.
- Power Monitoring and Regulation: Integrating a battery management system (BMS) to monitor voltage and current in real-time, preventing sudden power drops.
- Efficient Power Distribution: Optimizing power allocation between subsystems (e.g., using separate power rails for motors and logic circuits) to minimize interference and stabilize performance.

7.2 Incorporating Advanced Localization and Control Algorithms

Although the robot achieved basic autonomous navigation, several advanced algorithms and sensors were not utilized in this iteration:

- IMU and Optical Flow Sensors: While these sensors were included in the design, they were not used in the final implementation. Incorporating these sensors in the future could significantly enhance the robot's localization capabilities. For instance:
  - IMU: Provide precise angular velocity and acceleration data, enabling accurate pose estimation and inertial navigation.

- Optical Flow Sensors: Measure relative motion between the robot and the ground, helping refine position tracking and enhance dead reckoning accuracy.

- Algorithmic Enhancements:
  - Model Predictive Control (MPC): Integrating MPC would allow the robot to predict and optimize its movements over a time horizon, ensuring smoother navigation and better obstacle avoidance.
  - Rapidly-Exploring Random Trees (RRT): Implementing RRT-based path planning algorithms would improve the robot's ability to navigate complex environments by efficiently finding collision-free paths.
  - Sensor Fusion for Localization: Using data from IMU, optical flow, and existing sensors in a cohesive manner would enhance accuracy and robustness in localization tasks.

By leveraging these advanced techniques, the robot could transition from basic navigation to a more sophisticated autonomous system capable of operating in dynamic and unstructured environments.

7.3 Material Improvements and Accounting for Deflections

The chassis and structural components of the robot were primarily 3D-printed using lightweight materials. While this approach was cost-effective and enabled rapid prototyping, it introduced challenges in terms of rigidity and durability:

- Material Stiffness: The current materials showed slight deflections under high stress, particularly during rapid maneuvers or when the lance mechanism was activated. This affected overall precision and stability.

- Future Solutions:
  - Stiffer Materials: Switching to materials with higher stiffness-to-weight ratios, such as carbon-fiber composites or aluminum, would improve structural rigidity while maintaining a low weight.
  - Deflection Analysis: Conducting finite element analysis (FEA) during the design phase to predict and minimize deflections under load, ensuring consistent performance.
  - Reinforced Joints: Strengthening connections between components, such as motor mounts and chassis plates, to prevent wear and loosening over time.

8 Conclusion

The project successfully met its primary design objectives, culminating in a fully functional autonomous robot capable of navigating its environment, detecting balloons, and effectively popping them using the integrated lance and disk shooter mechanisms. The robot demonstrated its ability to execute complex tasks such as maneuvering with omnidirectional wheels, detecting obstacles, and interacting with its surroundings based on sensor inputs.

While the robot performed as anticipated in terms of functionality, it did not win any competition rounds. This was attributed to factors such as the competitive dynamics of the game, the need for faster decision-making under time constraints, and slight delays in response during high-pressure situations. Despite these challenges, the robot's core systems—movement, sensing, and weaponry—functioned reliably, validating the design approach.

The project has highlighted areas for improvement, particularly in optimizing sensor calibration and power management, as well as incorporating advanced algorithms to enhance performance. Refining these aspects in future iterations could significantly improve the robot's effectiveness in competitive scenarios.

Overall, the project served as an excellent learning experience, demonstrating the successful integration of mechanical, electronic, and software systems into a cohesive autonomous platform. The outcomes underscore the potential for further development, offering a solid foundation for exploring more advanced robotics concepts.

9 References

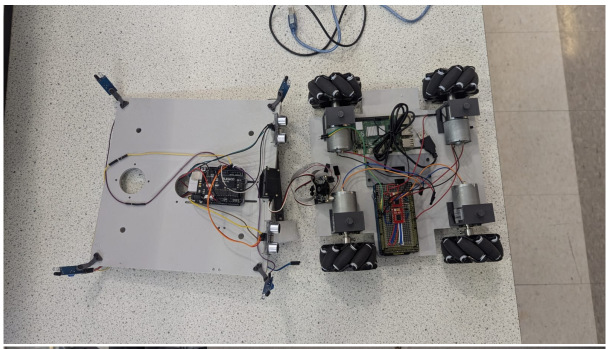
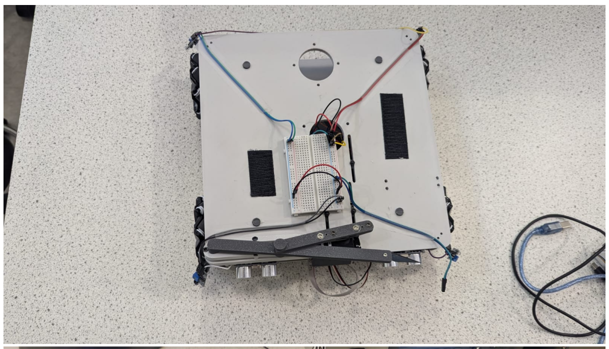
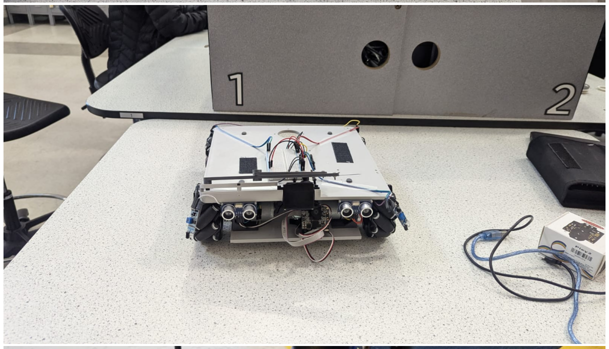
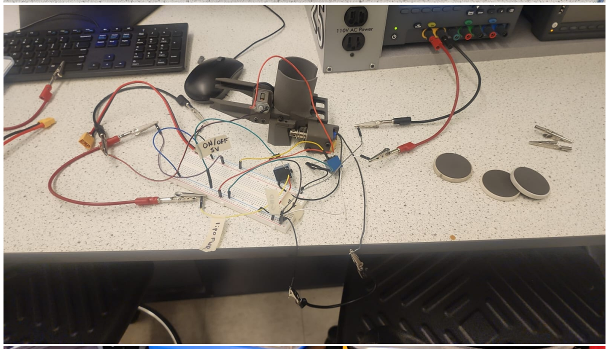
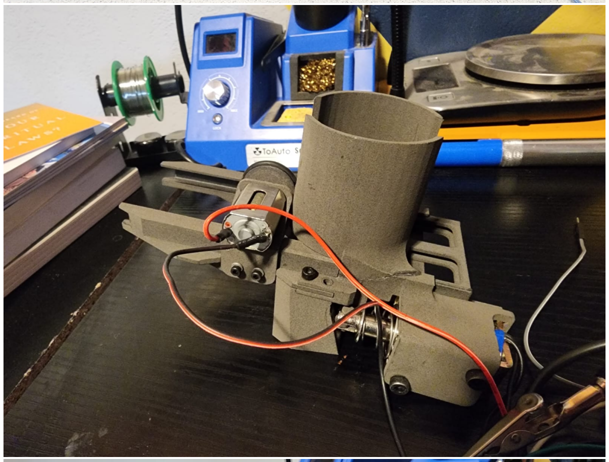
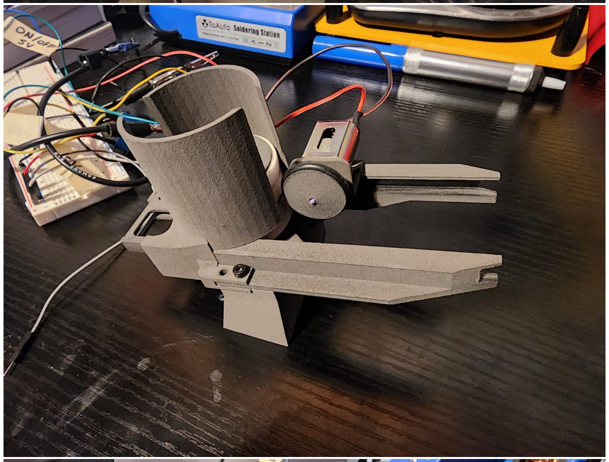
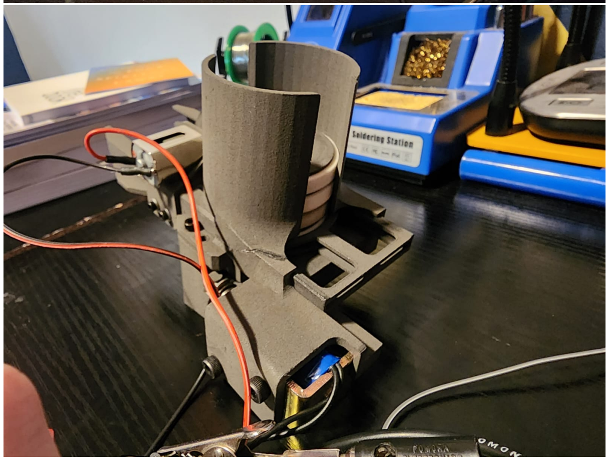
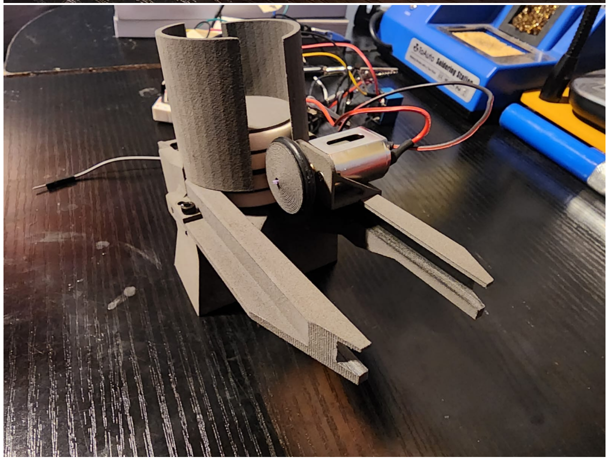
- Github repo of complete project,`https://github.com/srikanth-444/Mechatronics`
- Wikipedia, "Mecanum Wheel", `https://en.wikipedia.org/wiki/Mecanum_wheel`
- Arduino, "Motor Control", `https://www.instructables.com/Using-the-Sparkfun-Motor-Dri`
- Pixy pan and tilt, `https://pixycam.com/pixy2-pan-tilt-kit/`
- Servo control, `https://www.instructables.com/Arduino-Servo-Motors/`

A Bill of Materials (B.O.M)

Part	**Part Number**	**Qty.**	**Price**	**Source**
Primary Logic System				
Raspberry Pi 4	-	1	-	Provided
IMU	-	1	$20	Amazon
Flow Sensor	-	1	$8	Amazon
Drive/Weapon Subsystem				
Arduino Mega 2560	-	1	-	Provided
Motor Drivers (TB6612FNG)	-	2	$8	Amazon
Greartisan 300 RPM Gearmotors	-	4	$60	Amazon
Mecanum Wheels	-	4	$40	-
74,000RPM Flywheel Motor	-	1	-	Salvaged
Solenoid	-	1	14	Amazon
Vision and Sensing Subsystem				
Arduino Uno	-	1	-	Provided
Ultrasonic Sensors	-	2	-	Provided
IR Sensors	-	4	-	Provided
Pixy Camera	-	1	-	Provided
External Power				
USB Power Bank	-	1	-	Provided
12V Zeee Lipo Battery	-	2	-	Provided
Miscellaneous				
Voltage Regulator	-	1	$6	Amazon
Chassis Parts	-	N/A	$18	3D Printed
Hardware (screws, pins, etc)	-	N/A	$10	Amazon
			TOTAL	$184

Table 1: Bill of Materials (B.O.M)

B Images




1
2

ON/OFF
5V


ON/OFF
5V
Soldering Station

Soldering Station

Soldering Station