TEAM JORDAN

DARK EAGLES

6045

Future Engineers

TABLE OF CONTENTS

Chapter One

Introduction

This project involves a collaborative effort by aspiring engineers to design and build a state-of-the-art robot.

The team consists of members with diverse skills and expertise in areas such as programming, mechanical design, and electronics.

This document details the journey of the project, from the initial concept to the final product, highlighting the challenges and successes encountered along the way.

Chapter TWO

Our Team

The team comprises an expert programmer, a skilled 3D modeler, a proficient electronics specialist, and a seasoned coach. Their combined expertise ensures high-quality results.

MOHAMMAD ALKHALILI

Handles all electronics aspects, from selecting components to circuit design and testing. Their knowledge ensures optimal integration of electronic parts, leading to reliable and functional systems that meet project specifications.

MAYSAM ALSHWAYAT

Responsible for developing, testing, and maintaining the software that drives our projects, ensuring smooth functionality and high performance. Their expertise in coding and troubleshooting brings the logic and intelligence behind every project to life.

YEHIA HIMMO

Focuses on creating precise and visually accurate 3D models for our designs, from initial concepts to final prototypes. Their mastery of 3D modeling software allows for seamless visualizations and aids in efficient project planning and execution.

Chapter THREE

Robot Design

Our robot, Sensoria, is a testament to innovative design. is fully 3d printed, and 2d laser cutting, which sets it apart from the crowd. The 3D modeling process allowed us to make a full costume robot for certain tasks.

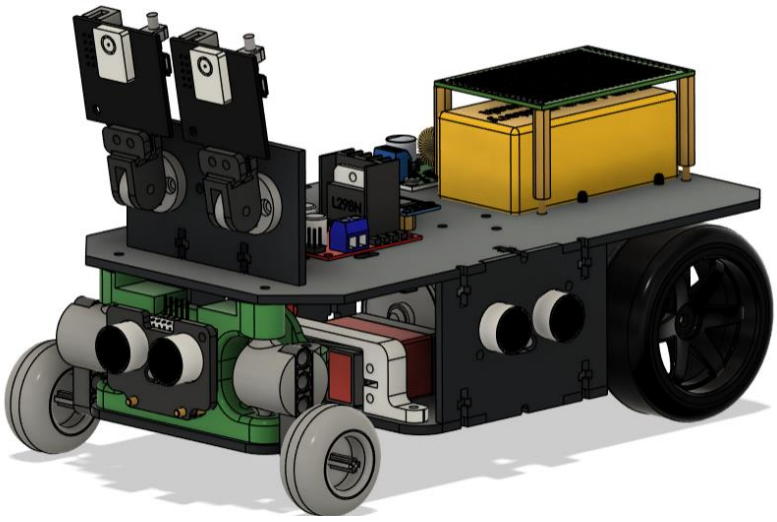
benefits of fully 3D modeled design

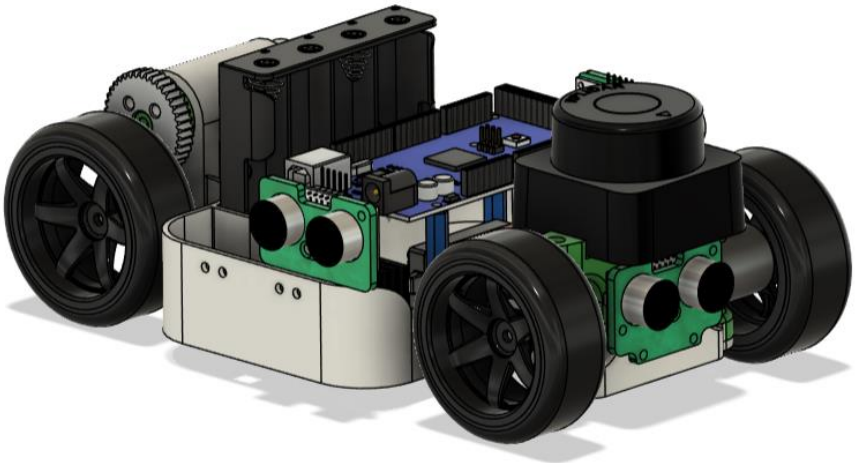
Fully 3D design enhances visualization, improves accuracy, enables rapid prototyping, and fosters collaboration, leading to more innovative solutions and streamlined production processes.

Fully 3D design allows for easier modifications, saving time and reducing costs during the development process.

A fully modeled design enhances accuracy, reduces errors, and improves communication during project execution.

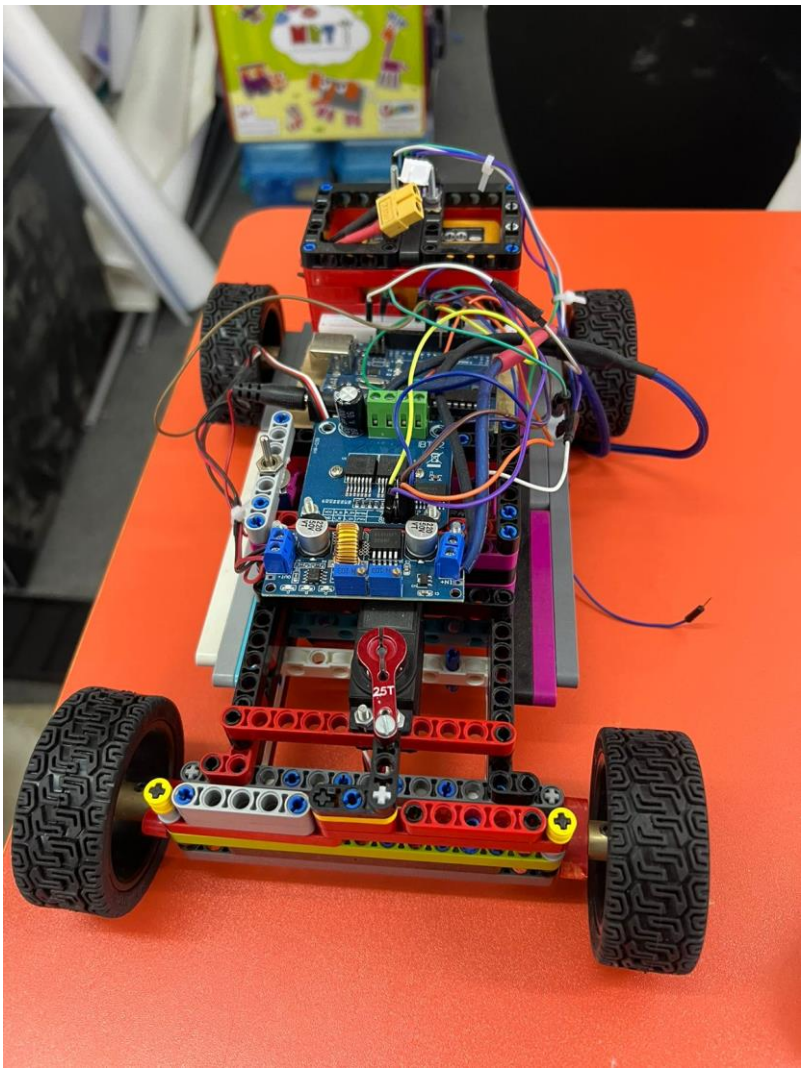
Our robot went through multiple prototype iterations before reaching perfection. Each prototype allowed us to identify areas of improvement and make necessary adjustments. We needed to print every prototype in order to physically test and validate our design ideas. By doing so, we were able to ensure that our final product was not only functional but also met all of our desired specifications. The iterative process of prototyping ultimately led us to a successful and refined end result that exceeded our initial expectations

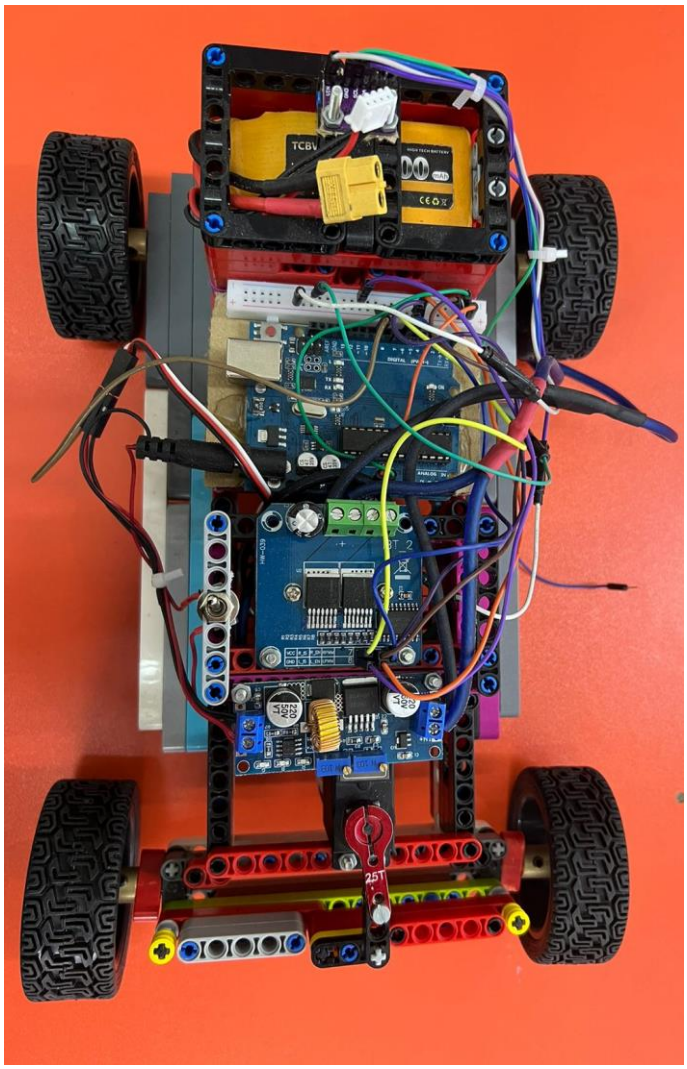




- **First Generation:**

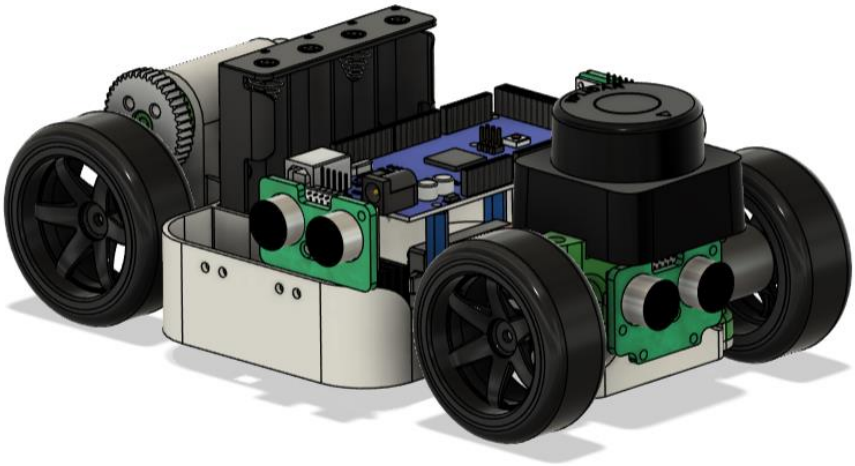
For our first-generation design, we used LEGO Technic parts to build the robot, with an Arduino as the microcontroller. However, we encountered several challenges with this setup: the LEGO Technic system offered limited design flexibility, resulting in a heavier structure. Additionally, cable management was difficult, and the overall size of the design was larger than desired.

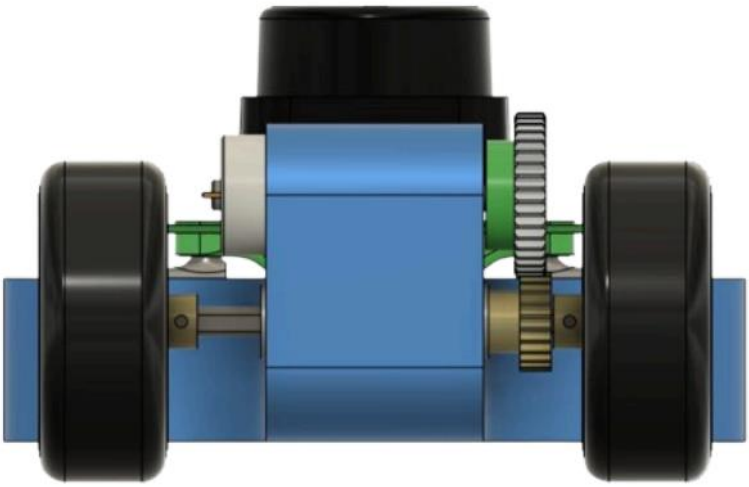




- **Second Generation:**

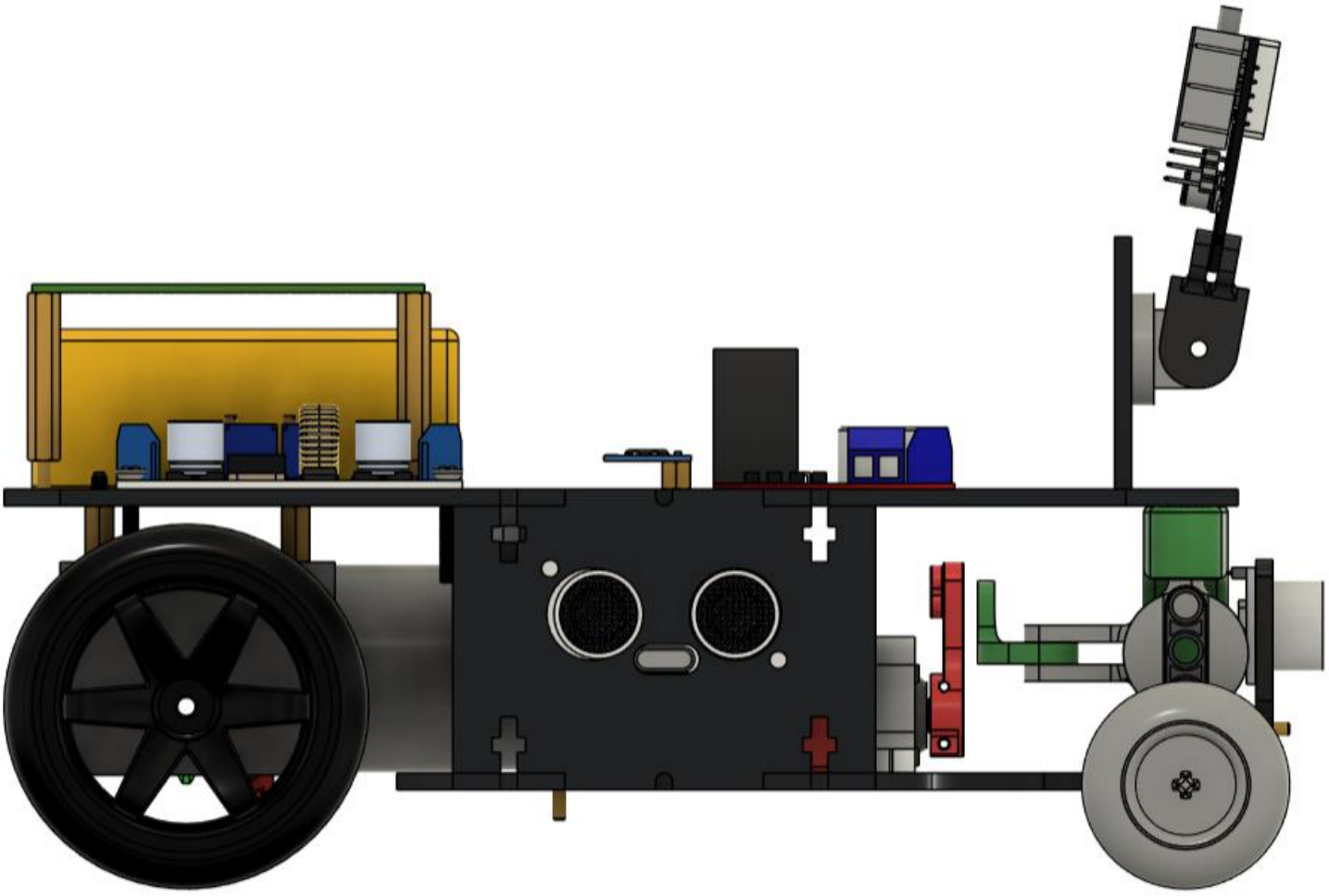
The second-generation design showed significant improvement as we built it entirely with 3D-printed parts. However, some enhancements were still needed: the fully 3D-printed structure made it challenging to modify components without reprinting the entire body. Additionally, we used a large wheel at the front, which wasn't ideal for a steering-drive robot and affected maneuverability.

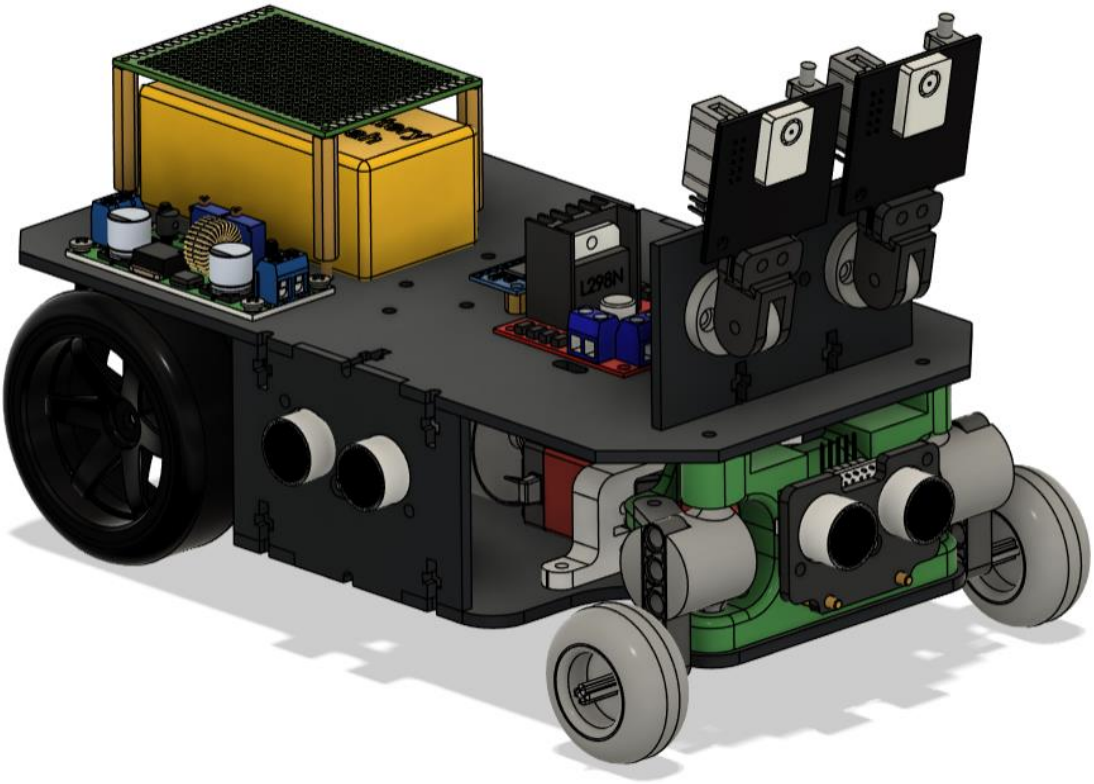


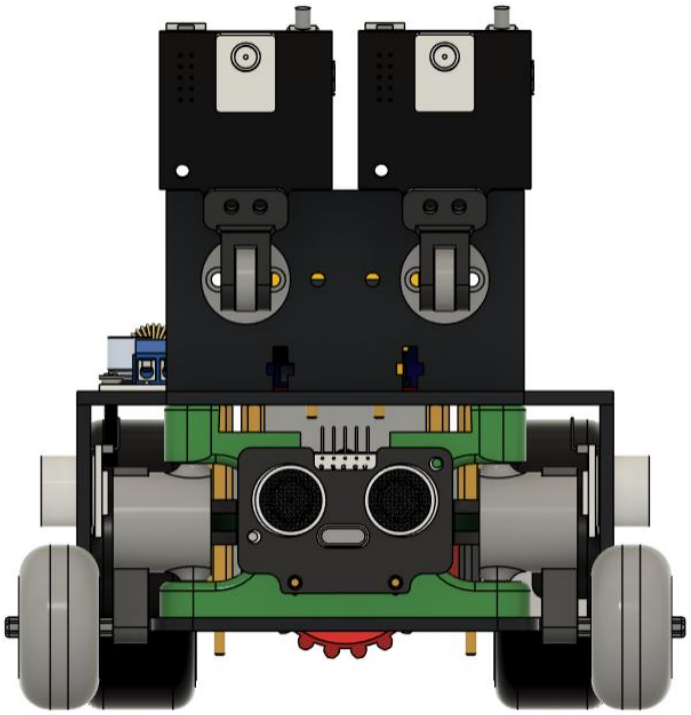


- **Third and Final Generation:**

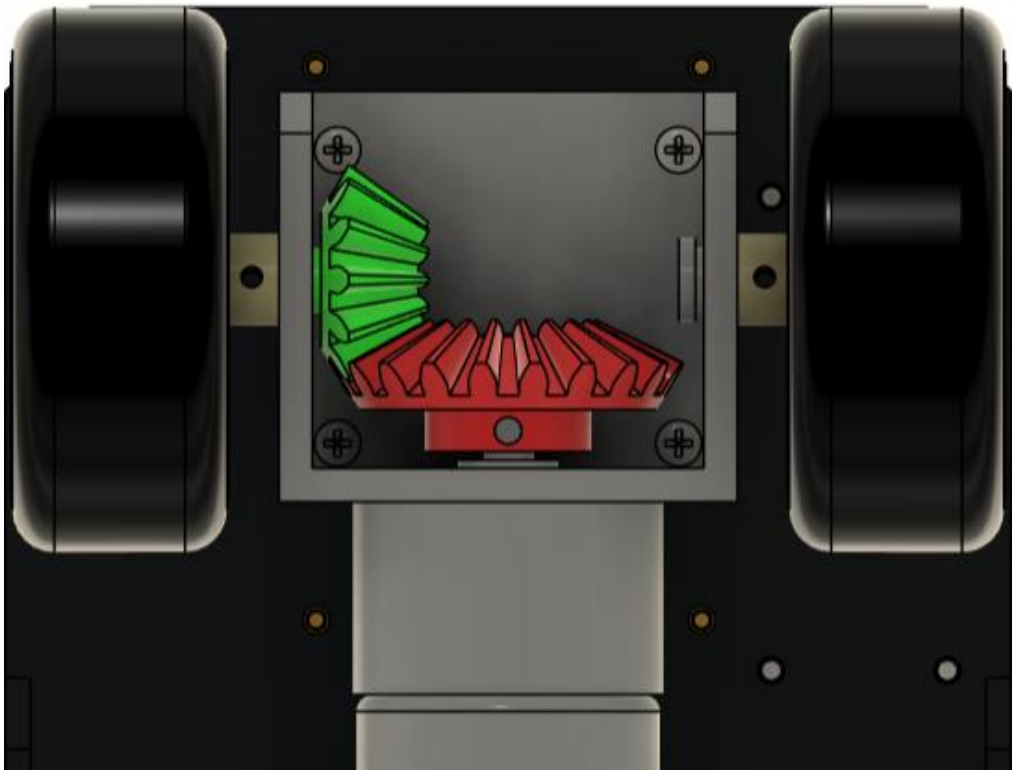
The final design exceeded our expectations. We reduced the robot's size to a compact 22cm x 12cm x 8cm. For this version, we crafted the body from laser-cut acrylic, allowing for easy modifications and additions. To further enhance efficiency, we incorporated a few 3D-printed components, which saved time on adjustments. Assembly was streamlined with 3mm screws, and we opted for a small front wheel, providing smoother and more precise rotation for improved maneuverability.

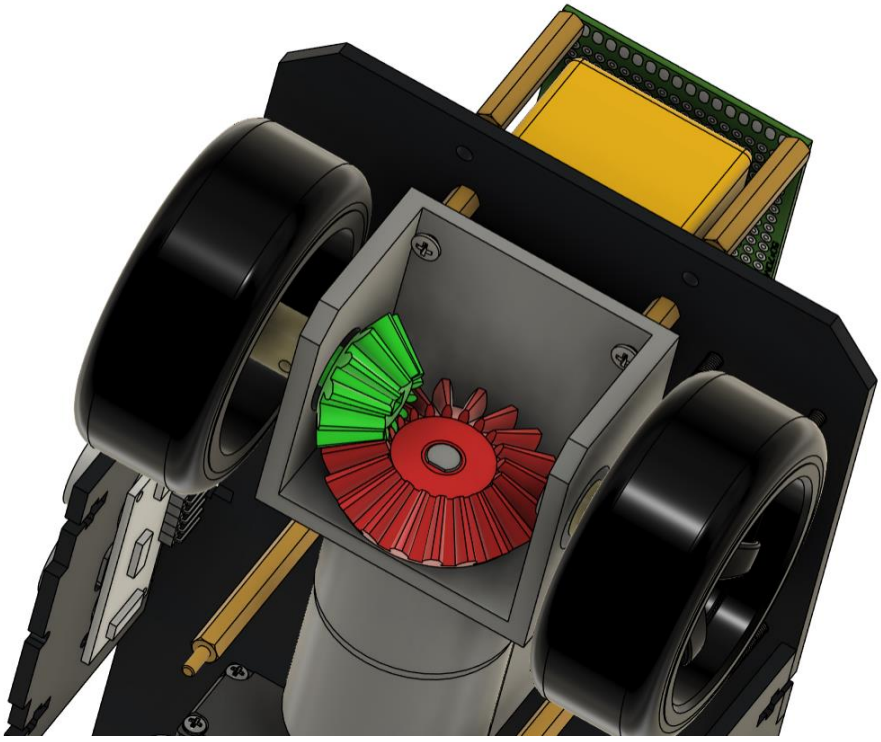



L298N



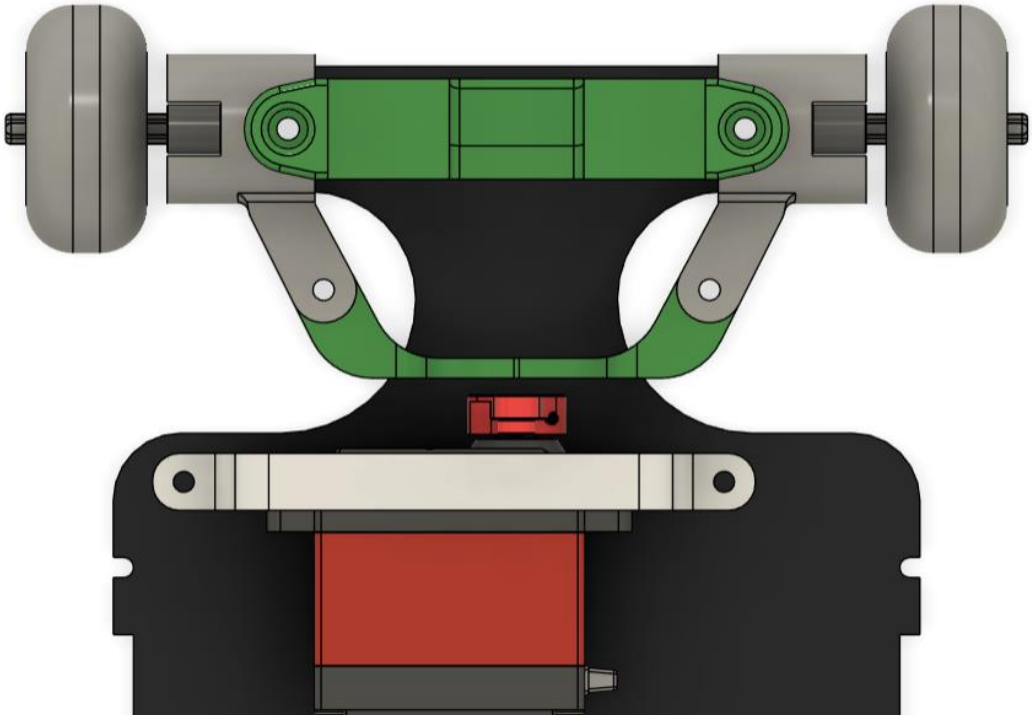
Driving System

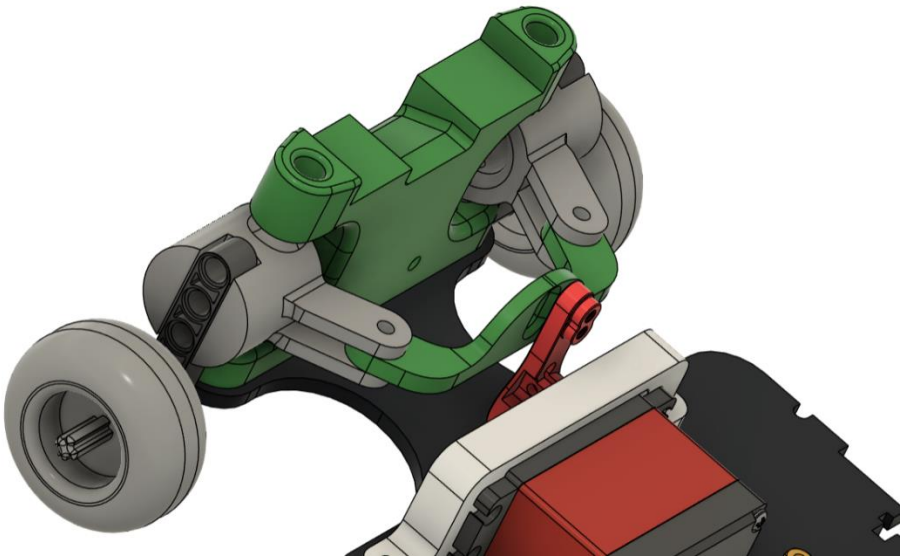




Steering System

Our steering system is meticulously designed and fully 3D modeled to ensure precision and reliability. By leveraging the capabilities of 3D printers, we are able to create intricate components that are not only durable but also highly accurate. With a torque of 20 kg, our steering
system is able to provide stable and responsive control, making it ideal for a wide range of applications. Our advanced design ensures that our steering system is not only efficient but also built to last.





Chapter FOUR

Design and Prototype

- **Software**

Designing our robot using Fusion 360 was an incredible journey that combined creativity, engineering, and a lot of trial and error. Fusion 360 allowed us to create detailed 3D models, simulate movements, and visualize our design before building anything physical. This software was essential for prototyping and refining our ideas.

- **Technology**

To bring our robot design to life, we relied heavily on advanced 3D printing technology and high-quality printers. The complexity of
our design required precision and durability that standard printers simply couldn't provide. We needed a printer capable of handling various materials, such as strong plastics and flexible components, to
accurately replicate our Fusion 360 models.
The advanced 3D printer allowed us to create intricate geometries and tight tolerances that were crucial for our robot's performance.

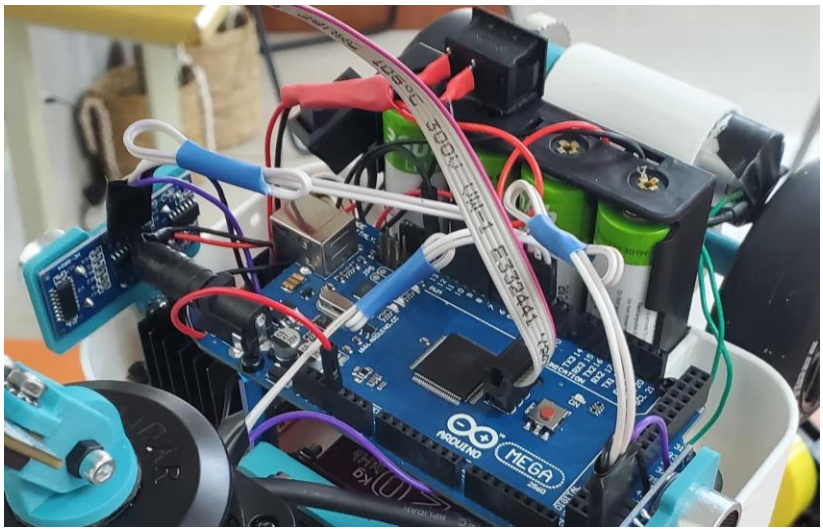
Bambu Lab

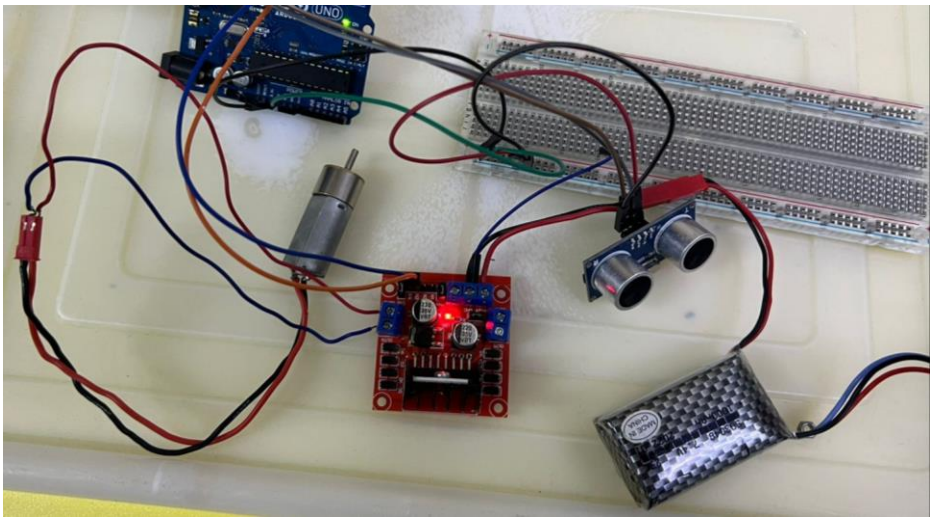
Chapter FIVE

Electronics

For the electronics, we used an Arduino Nano as the microcontroller. The drive system features powerful 500 RPM DC motors paired with an L298N motor driver, all powered by a 3-cell 11.1V LiPo battery. For steering, we incorporated a 20kg servo motor, ensuring precise control and responsiveness.

To accurately monitor rotation angles, we integrated an MPU6050 IMU (Inertial Measurement Unit). The Pixy 2 camera connects to the Arduino via ICSP pins for efficient communication, and we used a voltage regulator to step down the 11.1V battery output to 5V to safely power the Arduino.





below is the datasheet for all robot components:

Component	Description
Arduino Nano	1. Microcontroller: ATmega328 2. Operating Voltage: 5V 3. Input Voltage (recommended): 7-12V 4. Input Voltage (limits): 6-20V 5. Digital I/O Pins: 14 (6 of which provide PWM output) 6. Analog Input Pins: 8 7. DC Current per I/O Pin: 40 mA 8. Flash Memory: 32 KB (ATmega328) of which 2 KB used by bootloader 9. SRAM: 2 KB (ATmega328) 10. EEPROM: 1 KB (ATmega328) 11. Clock Speed: 16 MHz 12. Dimensions: 45 mm x 18 mm 13. Weight: 7 g
Titan Dc Gearhead Motor 12V 500 RPM SP	1. Nominal Voltage: 12V 2. No-Load Speed: 500 RPM 3. No-Load Current: ~200-300 mA 4. Stall Torque: Approximately 12-15 kg·cm 5. Stall Current: ~3-5 A 6. Rated Torque: ~8 kg·cm 7. Gear Ratio: Typically around 45:1 to 50:1 for 500 RPM 8. Output Shaft Diameter: ~6 mm (may vary) 9. Motor Type: Brushed DC with gearbox 10. Body Dimensions: Generally 37 mm in diameter, with length varying by specific model 11. Shaft Type: D-shaped or round shaft, depending on model 12. Mounting Holes: Standard mounting holes for secure attachment
DS3218 digital servo 20KG 270 degrees	1. Operating Voltage: 4.8V - 6.8V 2. Stall Torque: • 17 kg·cm at 4.8V • 20 kg·cm at 6.8V 3. Operating Speed: • 0.16 sec/60° at 4.8V • 0.14 sec/60° at 6.8V 4. Rotation Range: 270 degrees 5. Control System: Pulse Width Modulation (PWM) 6. Pulse Width Range: 500-2500 µs

	7. Gear Type: Metal gears for high durability 8. Motor Type: Coreless motor 9. Connector Wire Length: ~30 cm 10. Dimensions: 40 mm x 20 mm x 40.5 mm (approximate) 11. Weight: ~60g 12. Connector Type: 3-pin (Signal, VCC, Ground)
hc-sr04 ultrasonic sensor	1. Operating Voltage: 5V DC 2. Operating Current: ~15-20 mA 3. Range: 2 cm to 4 meters (depending on conditions) 4. Measurement Angle: 15° (typical) 5. Accuracy: ±3 mm 6. Resolution: 0.3 cm 7. Frequency: 40 kHz 8. Output Pins: • VCC: Power supply (5V) • GND: Ground • Trigger (Trig): Input pin to send a pulse • Echo (Echo): Output pin that gives the time duration of the echo pulse 9. Pulse Width: • Trigger: Minimum 10 µs • Echo: 5V pulse duration corresponds to the distance 10. Dimensions: 45 mm x 20 mm x 15 mm 11. Weight: ~8g
L298N DC Motor Driver	1. Operating Voltage: • Motor Voltage (V_s): 4.5V - 36V • Logic Voltage (V_sense): 4.5V - 5.5V 2. Output Current: • Continuous: 2A per channel • Peak: 3A per channel (for short durations) 3. Output Power: Up to 25W (depending on input voltage and load) 4. Control Type: Dual H-Bridge 5. Input Pins: • IN1, IN2, IN3, IN4: Control inputs to drive the motors • ENA, ENB: Enable pins for channels A and B 6. Output Pins: • OUT1, OUT2: Motor A output

	• OUT3, OUT4: Motor B output 7. Logic Control: TTL (0V or 5V) for controlling motors 8. Current Sensing: Integrated sense resistors for current feedback (if needed) 9. Thermal Shutdown: Yes (with over-temperature protection) 10. Dimensions: 43 mm x 43 mm x 27 mm 11. Weight: ~30 g
3-Cell LiPo Battery (11.1V, 1500mAh)	1. Nominal Voltage: 11.1V (3.7V per cell) 2. Fully Charged Voltage: 12.6V (4.2V per cell) 3. Capacity: 1500mAh (1.5Ah) 4. Discharge Rate: • Continuous: 20C (30A) • Peak: 30C (45A) 5. Charge Voltage: 12.6V (4.2V per cell) 6. Discharge Voltage: 9V (3.0V per cell) 7. Dimensions: Typically around 70 mm x 35 mm x 20 mm (varies by manufacturer) 8. Weight: ~100-150g (depending on the manufacturer) 9. Connector Type: Typically JST-XH or XT60, depending on the model 10. Max Charge Current: 1C (1.5A) 11. Battery Type: Lithium-Polymer (LiPo) 12. Operating Temperature: • Charging: 0°C to 45°C • Discharging: -20°C to 60°C
pixy2 camera	1. Sensor Type: CMOS image sensor 2. Resolution: 640x400 pixels (VGA) 3. Field of View (FoV): 75° horizontal, 50° vertical 4. Frame Rate: 50 frames per second (FPS) 5. Interface: • I2C: Communication with microcontrollers like Arduino • SPI: For faster data transfer (can be used in advanced setups) • UART: For serial communication • PWM: To control motors or other peripherals 6. Power Supply: 5V DC (via USB or external power) 7. Operating Current: ~100mA (no load), higher under load 8. Lens: Fixed lens with 3.6mm focal length

	9. Image Processing: On-board object detection (color-based tracking, object signatures) 10. Color Detection: Detects up to 7 colors in real-time (user-defined "signatures") 11. Communication Speed: • I2C: Up to 400kHz • SPI: Up to 10 Mbps 12. Dimensions: 35 mm x 50 mm x 27 mm 13. Weight: 20g 14. Storage: Onboard flash memory for storing color signatures and configuration settings 15. Software Support: PixyMon software for configuration and testing, compatible with many programming environments (Arduino, Raspberry Pi, etc.)

Chapter SIX

Camera

what are the reasons the Pixy2 camera is our main sensor?

1. Light Sensitivity: Cameras can capture a wide range of light intensities, allowing them to function in various lighting conditions, from bright daylight to dimly lit environments.

2. Dynamic Range: They can record details in both highlights and shadows, providing a more complete representation of a scene compared to the human eye.

3. Color Accuracy: Cameras can reproduce a wide spectrum of colors, helping to capture the richness and vibrancy of a scene.

Chapter SIX

Our Code

We utilized Arduino IDE and Pixymon, with their cutting-edge technology, to program the robot efficiently and effectively. The user-friendly interface of the IDE and Pixymon allowed us to easily write and upload code to the robot's microcontroller. The advanced features of the IDE, such as the serial monitor and debugging tools, enabled us to troubleshoot and fine-tune the robot's behavior.

Overall, the Arduino IDE played a crucial role in the successful programming of the robot, showcasing the power and versatility of modern technology in robotics.

Using functions in the Arduino IDE significantly improves both the efficiency and reliability of your code. By encapsulating specific tasks into functions, you make the code more modular, which reduces repetition and makes debugging easier.

Functions allow you to organize your code logically, breaking it down into manageable, reusable blocks. This not only shortens the code but also makes it easier to maintain and scale for more complex projects.

The uniqueness of the code comes from how we design custom functions tailored to our specific project requirements. Every system or project has its unique parameters, inputs, and outputs, so structuring functions around these specifics makes the code fully individualized. Instead of relying on generic approaches, we create functions that directly address our goals, making the code more efficient for our use case.

Reliability is enhanced by simplifying the main loop and isolating potential sources of error within specific functions.

When each function handles only one aspect of the project, it's easier to test and ensure its correct performance, thus reducing bugs or unexpected behavior. This method of coding increases the overall robustness of the system.

- **Camera Code**

Using a camera to extract color signatures for robot navigation is an effective technique often employed in robotics for tasks like line following or object tracking.

The basic idea is to use the camera as a sensor to detect specific colors or patterns in the environment, which then guide the robot's movements.

```
  pixy.init();                // Initialize the Pixy camera
}

// Main Loop
void loop() {
  for (y = 0; y < 1; y++) { // Loop to execute the following block once
    servo1.write(175);      // Set the servo position
    delay(500); // Wait for 0.5 seconds
    digitalWrite(in1, HIGH);
    digitalWrite(in2, HIGH); // Turn on the motors
    delay(1000); // Run motors for 1 second
    servo1.write(85); // Reset servo position
    delay(500); // Wait for 0.5 seconds
    digitalWrite(in1, HIGH);
    digitalWrite(in2, HIGH); // Turn on the motors again

    // Check for detected color blocks
    if (pixy.ccc.numBlocks > 0) {
      // Control based on the signature of the detected block
      if (pixy.ccc.blocks[0].m_signature == 1) {
        RedControl();      // Call function to handle red block
      } else if (pixy.ccc.blocks[0].m_signature == 2) {
        BlueControl();     // Call function to handle blue block
      }
    } else {
      Serial.println("No blocks detected"); // Print debug message if no blocks are found
    }
```

- **The steering system code**

In the steering system code, we used the `HIGH` and `LOW`commands to control the direction of the motors responsible for steering. These commands allow us to directly control the output signals to the servo, which in turn controls the movement of the steering mechanism.

```
void loop() {
  for (y = 0; y < 1; y++) { // Loop to execute the following block once
    servo1.write(175);      // Set the servo position
    delay(500); // Wait for 0.5 seconds
    digitalWrite(in1, HIGH);
    digitalWrite(in2, HIGH); // Turn on the motors
    delay(1000); // Run motors for 1 second
    servo1.write(85); // Reset servo position
    delay(500); // Wait for 0.5 seconds
    digitalWrite(in1, HIGH);
    digitalWrite(in2, HIGH); // Turn on the motors again

    // Check for detected color blocks
    if (pixy.ccc.numBlocks > 0) {
      // Control based on the signature of the detected block
      if (pixy.ccc.blocks[0].m_signature == 1) {
        RedControl();      // Call function to handle red block
      } else if (pixy.ccc.blocks[0].m_signature == 2) {
        BlueControl();     // Call function to handle blue block
      }
    } else {
      Serial.println("No blocks detected"); // Print debug message if no blocks are found
    }

    speedControl(); // Call speed control function
    pixy.ccc.getBlocks(); // Retrieve block data from the Pixy camera
    delay(1000); // Wait before the next loop iteration
  }

  delay(1000); // Wait before the next loop iteration

  for (i = 0; i < 3; i++) { // Loop to execute the following block three times
    servo1.write(-5);      // Set the servo position
    delay(500); // Wait for 0.5 seconds
    digitalWrite(in1, HIGH);
    digitalWrite(in2, HIGH); // Turn on the motors
    delay(1000); // Run motors for 1 second
    servo1.write(85); // Reset servo position
    delay(500); // Wait for 0.5 seconds
    digitalWrite(in1, HIGH);
    digitalWrite(in2, HIGH); // Turn on the motors again

    // Check for detected color blocks
    if (pixy.ccc.numBlocks > 0) {
      // Control based on the signature of the detected block
```