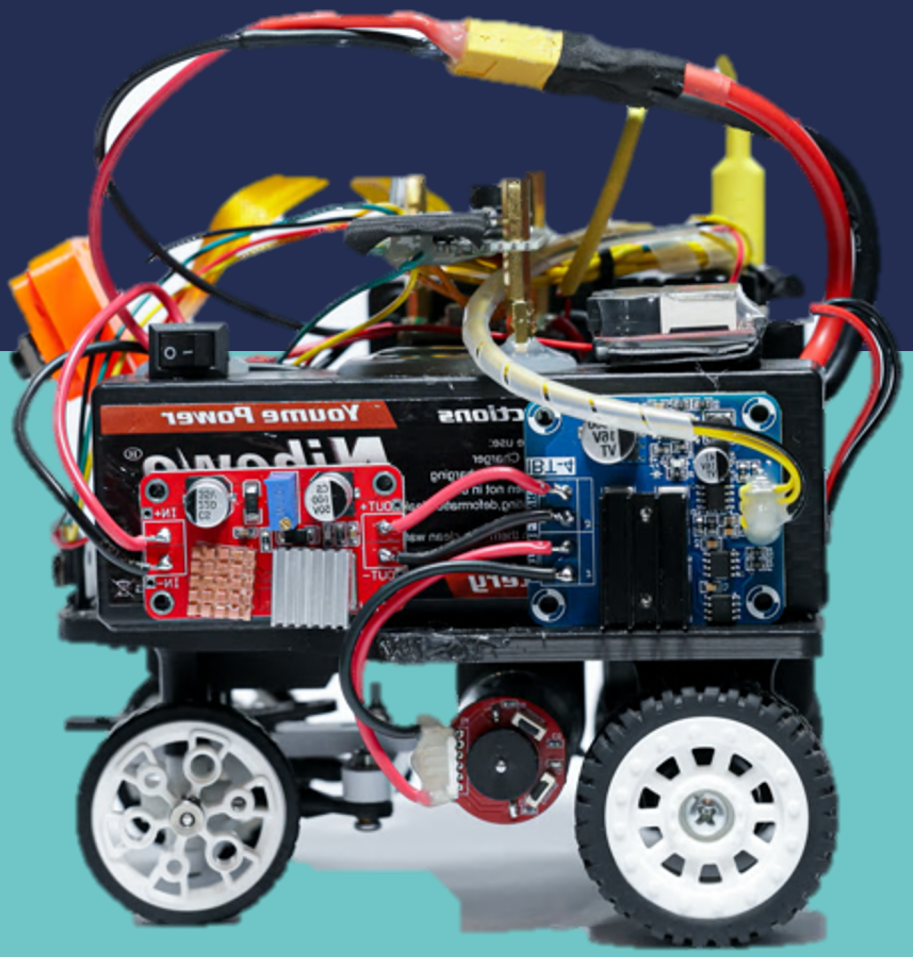
WRO
WORLD ROBOT OLYMPIAD™

#435

RayBot

2030



ENGINEERING BOOK

Table of Contents

1.Overview of the Robot Project

1. Design of a Small Car Movement System Using a Belt Drive

Our WRO Future Engineering Robot was designed to autonomously navigate and solve tasks such as obstacle avoidance, open challenges, and precision parking. By combining innovative hardware and sophisticated programming, the robot excels in functionality and adaptability.

Key components such as the Raspberry Pi 5, motors, and sensors work in tandem to create a high-performance machine. Python, programmed via Visual Studio Code, and computer vision techniques using OpenCV provide the logical backbone for its operations. The project focuses on the design and development of a robust movement system for a compact robot, powered by a single motor that drives the rear wheels by integrating 3 pairs of pulleys transmission using built, that to enhance best power, torque and vibration management in vehicle.

After evaluating various movement of these mechanisms, a belt drive system was chosen for its durability, efficiency, and long-term reliability. Unlike movement systems crafted using 3D printing, which can deteriorate with consistent use, the belt drive system offers enhanced resistance to wear and tear.

The design employs a 4 mm metal rod as the axle to secure the wheels, effectively transferring power from the motor to the wheels through a pulley and belt mechanism. To address potential issues with belt loosening during prolonged operation, a tension mechanism was integrated into the system to maintain belt stability and ensure continuous functionality.

The result is an energy-efficient system that maximizes the transfer of motor power to the wheels while minimizing failure risks.

This practical and sustainable solution demonstrates a balance between performance and reliability, aligning with the project's goal of creating a durable and efficient small car movement system.

2. Introduction

We are team **RayBot 2030**, a dynamic duo of passionate robotics enthusiasts. Based off the nature of the task, we divided the team, and each took on a key aspect of the challenge.

The robot project aims to address complex challenges in dynamic environments. This robot documentation outlines the integration of advanced hardware, software, and design methodologies to meet competition requirements.

Core goals include reliability, efficiency, and adaptability, achieved through iterative testing and refinement.

Introduction

The development of an autonomous robot for the **WRO Future Engineering Challenge** represents an ambitious endeavor that combines cutting-edge technology, innovative design, and strategic problem-solving. This project was conceptualized with the goal of creating a high-performing, adaptable robot capable of navigating complex environments, solving real-world challenges, and excelling in competitive scenarios.

At the core of the project is the integration of advanced computational and mechanical systems. The **Raspberry Pi 5**, serving as the robot's brain, facilitates real-time decision-making and computational tasks. Complemented by the **Camera Module 3** and OpenCV-based vision processing, the robot is equipped to detect obstacles, navigate through dynamic environments, and perform precision tasks such as parking. High-torque motors, a gyroscope, and

ultrasonic sensors ensure stability, responsiveness, and accuracy, making the robot a versatile solution for a wide range of tasks.

This project emphasizes a multidisciplinary approach, integrating engineering principles, robotics, and programming. From initial conceptualization to prototyping and final testing, the team employed iterative development to refine each subsystem and ensure seamless integration. Decisions regarding component selection, chassis design, and power management were guided by rigorous testing and data-driven insights.

The goal of this project is not only to build a robot capable of competing at the highest level but also to push the boundaries of robotics by demonstrating innovative approaches to design, implementation, and testing. This report outlines the processes, challenges, and achievements of the project, showcasing the collective effort and expertise of the team in delivering a competition-ready robot that reflects both precision engineering and creative problem-solving.

3.Team Photographs

RayBot 2030

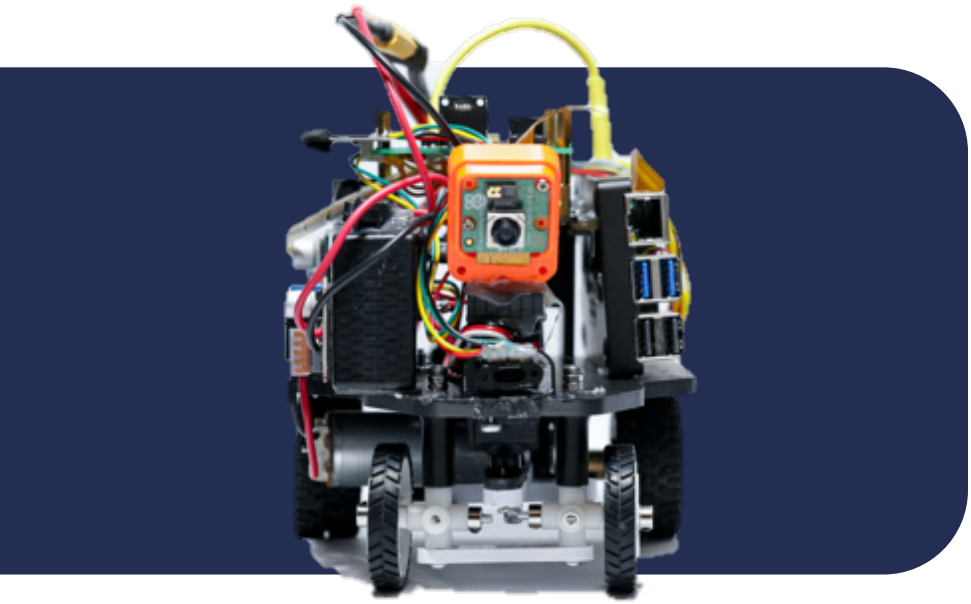
Aseel & Aljoharah

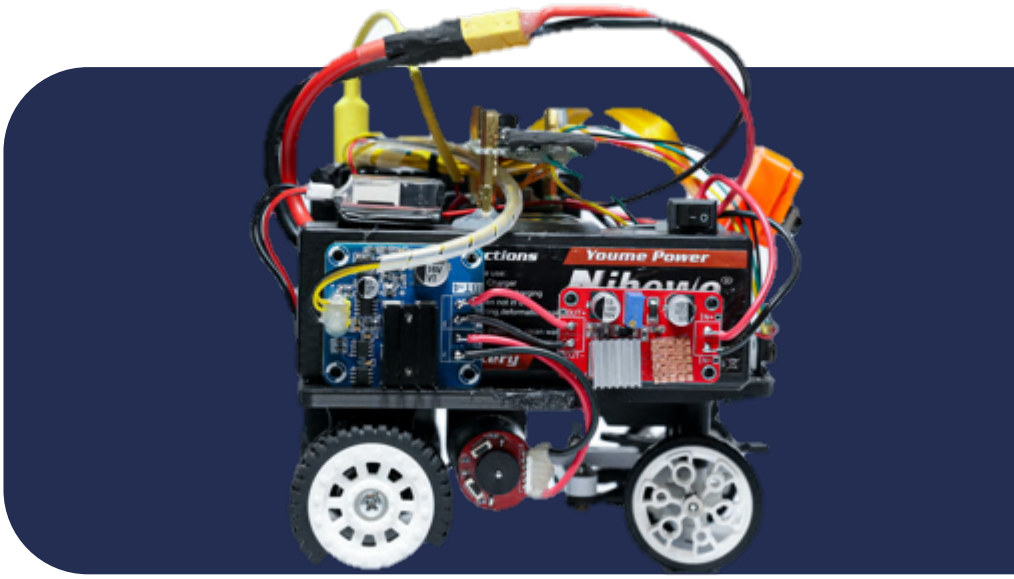
4.Robot Photographs

TOP & BOTTOM VIEWS

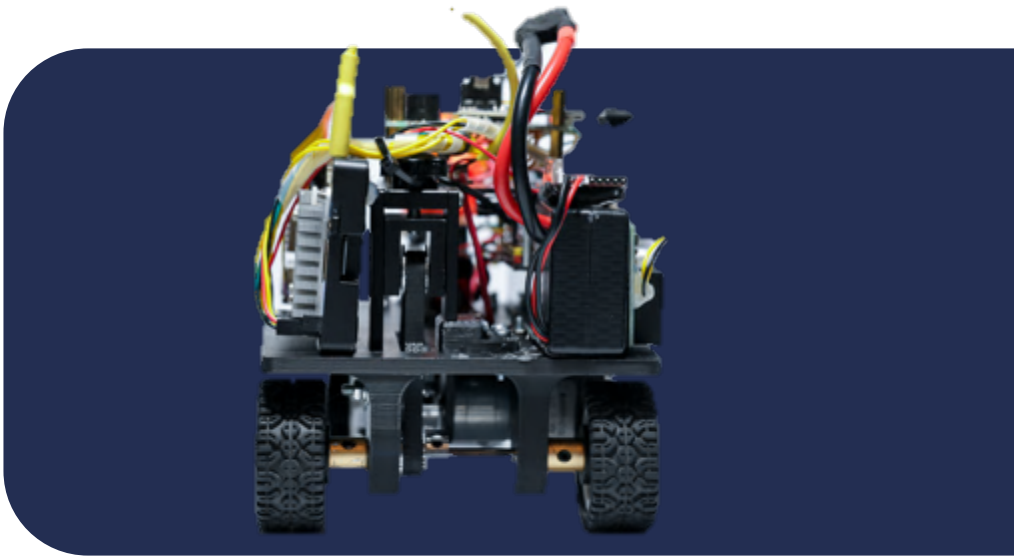
FRONT & BACK VIEWS

LEFT & RIGHT VIEWS

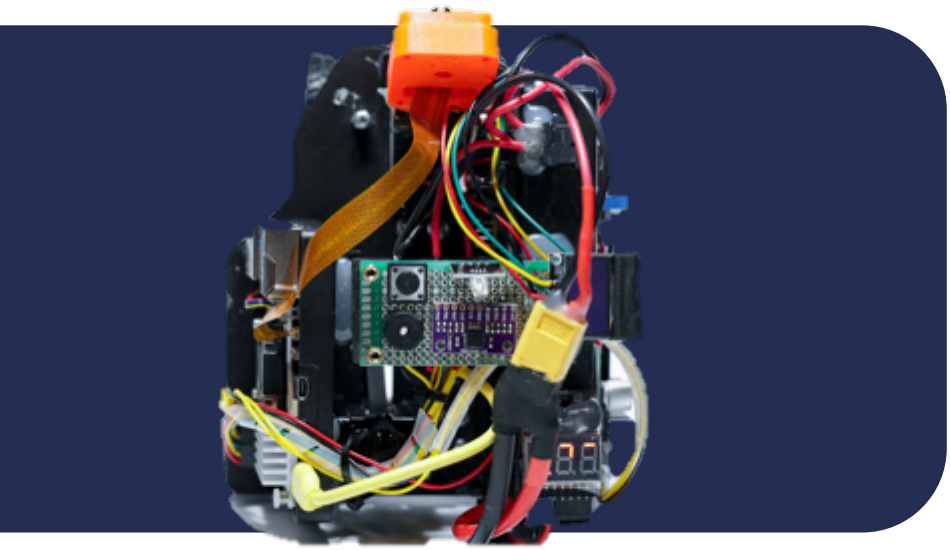


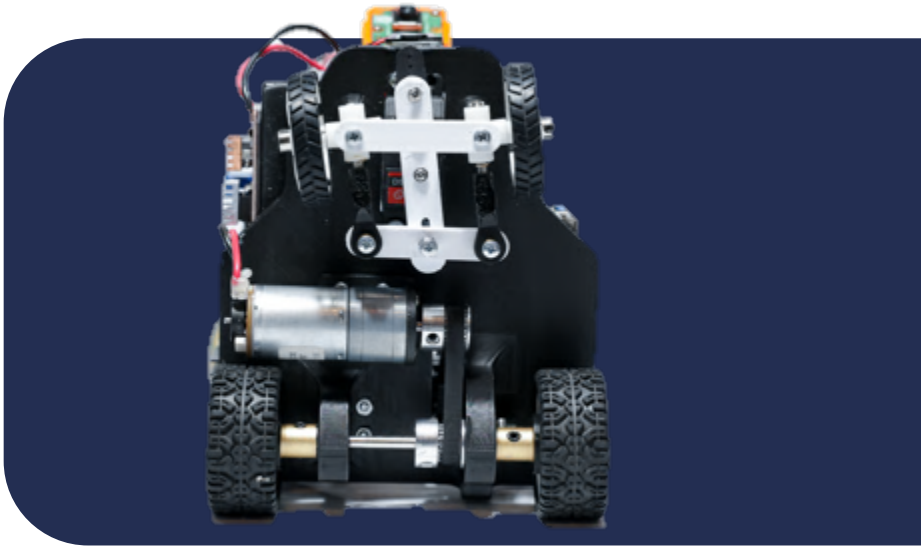


The Robot

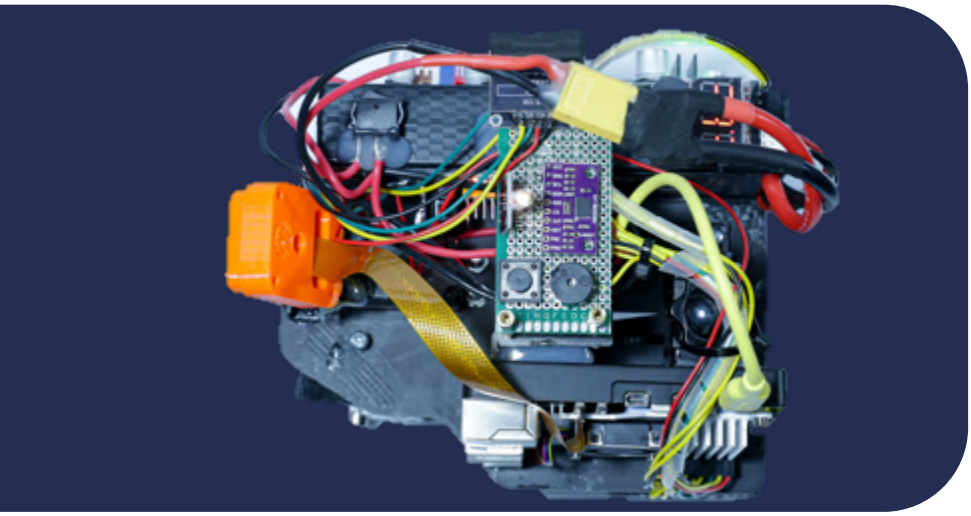


The Robot



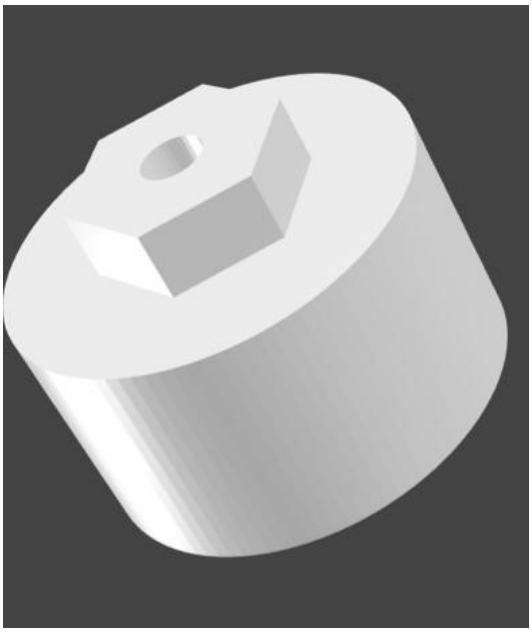


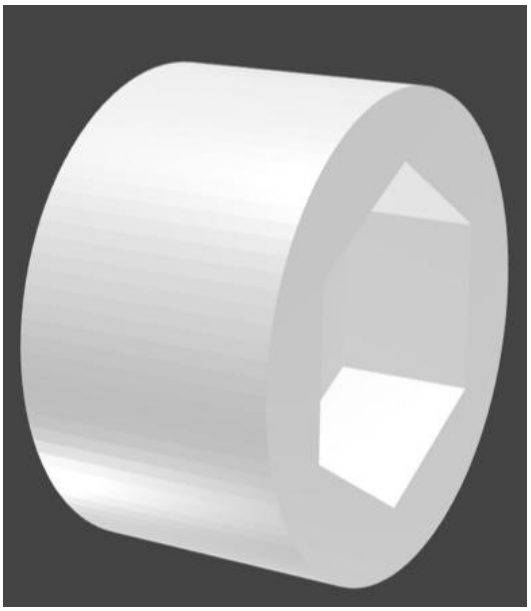
The Robot

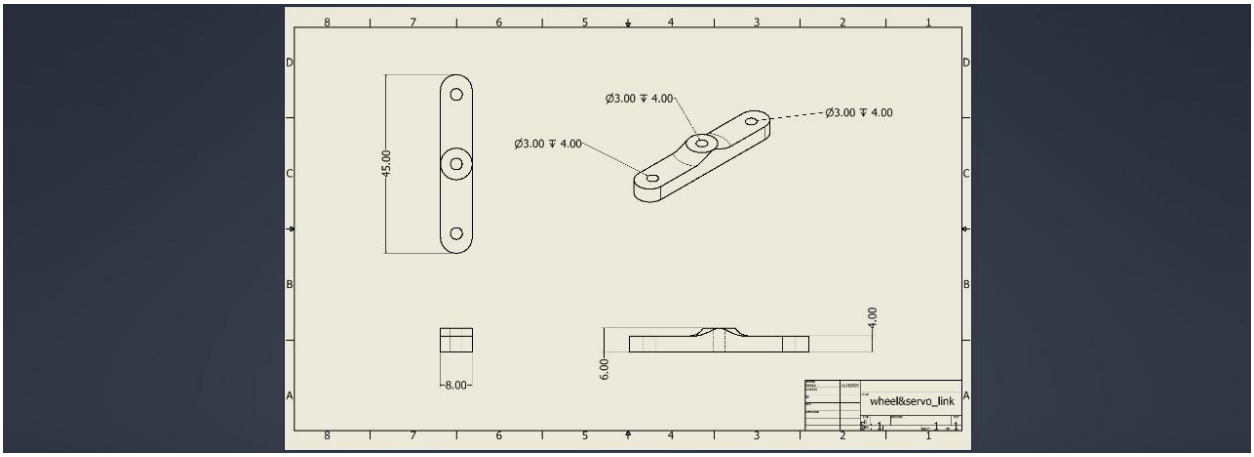


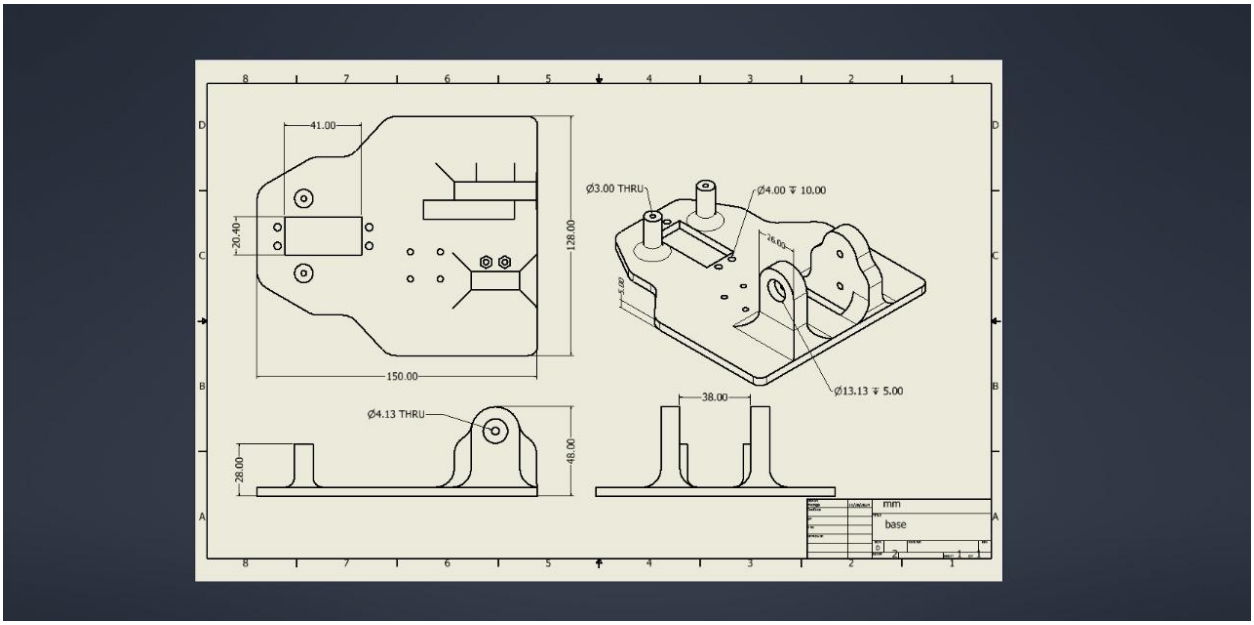
5. Robot Project's components

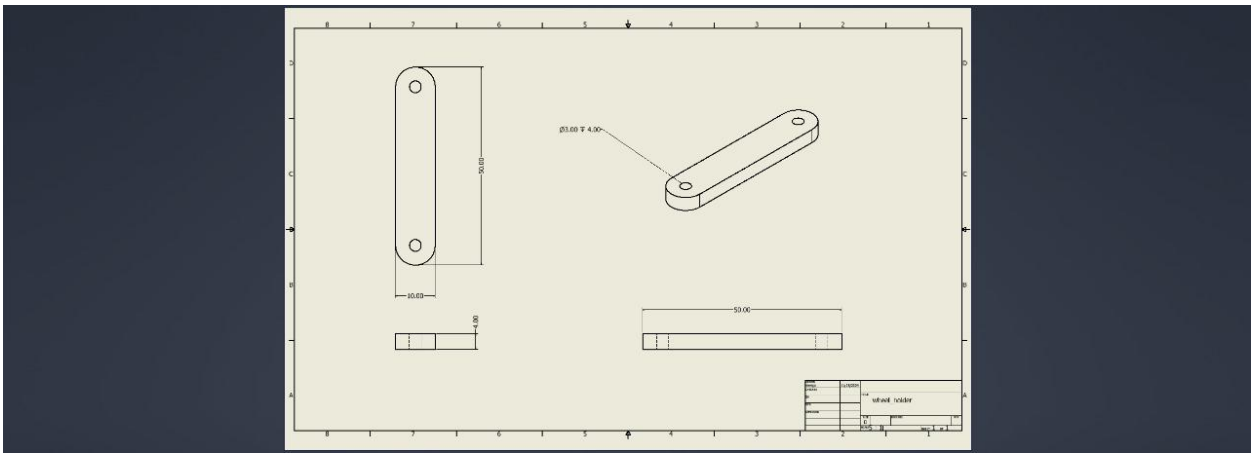
Mechanical Parts:

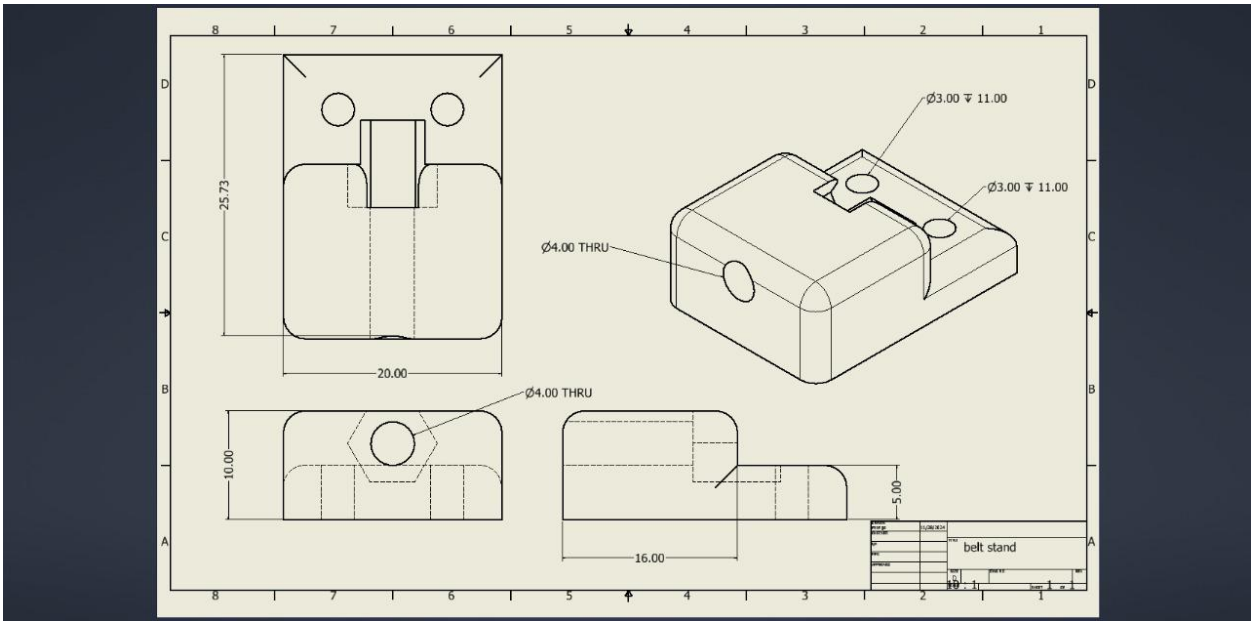





Ø3.00 ↧ 4.00
Ø3.00 ↧ 4.00
Ø3.00 ↧ 4.00
45.00
8.00
6.00
4.00
wheel&servo_link


41.00
20.40
128.00
150.00
Ø3.00 THRU
Ø4.00 ↧ 10.00
16.00
Ø13.13 ↧ 5.00
38.00
Ø4.13 THRU
28.00
48.00
mm
base





Electricals Parts:

1. Raspberry Pi 5

Overview

The Raspberry Pi 5 is the latest iteration in the Raspberry Pi family, offering significant performance upgrades and enhanced features for developers, educators, and hobbyists. Its 8GB RAM configuration, 64-bit Raspbian OS, and ample 128GB SD card storage make it an ideal choice for resource-intensive applications.

Key Specifications and Features

1. **Processor and Performance**:
    a. **CPU**: Powered by a quad-core **ARM Cortex-A76** processor running at a clock speed of up to 2.4 GHz.
    b. **GPU**: Advanced VideoCore VII GPU for improved graphics performance.
    c. **Performance**: Enhanced computational capability compared to previous versions, enabling smooth multitasking, real-time data processing, and high-speed applications.
2. **Memory (RAM)**:
    a. **8GB LPDDR4X**: This configuration provides substantial memory for running multiple processes, handling large datasets, and performing complex computations, such as AI/ML workloads and image processing.
3. **Operating System**:
    a. **Raspbian OS 64-bit (Full with Desktop)**:
        i. A Debian-based Linux distribution optimized for Raspberry Pi devices.
        ii. Includes a desktop interface for user-friendly navigation and application management.
        iii. Ideal for coding, software development, and general computing tasks.

4. **Storage**:
    a. **128GB SD Card (SanDisk)**:
        i. High-capacity storage for installing the operating system, software libraries, and storing project files.
        ii. Sufficient space for resource-intensive applications, such as media servers, databases, or large-scale coding projects.
5. **Connectivity**:
    a. **USB Ports**: Multiple USB 3.0 and USB 2.0 ports for peripherals like keyboards, mice, and external drives.
    b. **Networking**: Gigabit Ethernet and Wi-Fi 6 for high-speed internet connectivity.
    c. **Bluetooth**: Latest Bluetooth version for low-latency wireless communication.
6. **IO Interfaces**:
    a. **GPIO**: 40-pin GPIO header for connecting sensors, actuators, and other peripherals, ideal for robotics and IoT projects.
    b. **Display Output**: Dual micro-HDMI ports supporting resolutions up to 4K at 60fps.
7. **Power Supply**:
    a. **USB-C Power Input**: Requires a high-quality 5V/5A power adapter for optimal performance.
    b. **Efficiency**: Improved power management ensures stable operation even under heavy loads.

Applications

- **Educational Tools**: Perfect for learning programming, electronics, and IoT concepts.
- **Development Platform**: Suitable for software development, AI/ML projects, and prototyping.

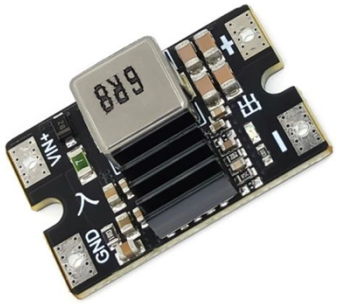
- **Media Center**: Can be configured as a 4K media server or entertainment hub.
- **IoT Hub**: Acts as a reliable base for Internet of Things projects due to its GPIO capabilities and wireless connectivity.

Advantages

- Powerful CPU and GPU for demanding tasks.
- Ample 8GB RAM for seamless multitasking and handling of large datasets.
- Pre-installed Raspbian OS ensures quick setup and a robust development environment.
- Generous 128GB SD card provides plenty of storage for data-heavy projects.
- Extensive connectivity options support a wide range of peripherals and use cases.

Our selective Raspberry Pi 5 with 8GB RAM, Raspbian OS, and 128GB SD card offers a balanced blend of performance, versatility, and ease of use, making it a standout choice for both beginners and professionals in the tech community.

2. DC-DC Step Down Converter



The DC-DC Step Down Converter is a critical power management component in our robot, ensuring stable voltage regulation for various subsystems. Below is the detailed component information:

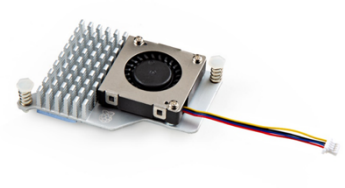
Model: Mini560 DC-DC Step Down Converter
Details:

- **Output Voltage Options**: Configurable output of 3.3V, 5V, 9V, and 12V to accommodate diverse power requirements of connected components.
- **Current Capacity**: Supports up to 8A, providing sufficient power for high-demand devices.
- **Voltage Regulation**: Designed to step down higher input voltages to stable, lower levels, ensuring consistent performance for sensitive electronics.
- **Compact Design**: Miniature size allows for easy integration into tight spaces, maintaining the robot's compact structure.
- **Efficiency**: High-efficiency design minimizes power loss, reducing heat generation and improving overall energy management.

Weight (g): 7g
Component Role: Voltage Regulator for stabilized power delivery to various electronic subsystems.

This entry ensures the DC-DC Step Down Converter is documented clearly, emphasizing its role and contribution to the robot's power management system

3. Raspberry Pi Active Cooler



Overview:

The Raspberry Pi Active Cooler is designed to prevent overheating of the Raspberry Pi 5. It ensures that the CPU and GPU run optimally during extended operation, especially when performing intensive tasks such as image processing or running AI models.

Key Specifications and Features:

- Cooling Type: Active (fan-based cooling)
- Compatibility: Designed specifically for Raspberry Pi models, including Raspberry Pi 5
- Size: Compact and fits on top of the Raspberry Pi without interfering with GPIO pins.
- Power Supply: Powered by the Raspberry Pi itself via GPIO pins.

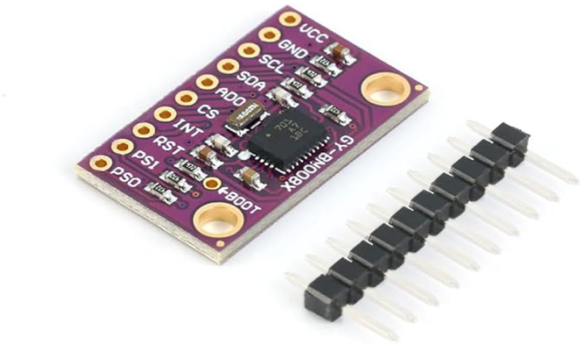
Advantages:

- Prevents overheating, maintaining system stability.
- Compact design makes it easy to install.
- Ideal for long-running tasks, ensuring the Raspberry Pi operates at peak performance.

Disadvantages:

- Requires GPIO pins for power, reducing the number of available pins for other peripherals.
- Might increase overall system noise due to fan operation.

4. Gyroscope (BNO085)



Overview:

The BNO085 is a high-performance 9-axis sensor that combines an accelerometer, gyroscope, and magnetometer into a single unit, offering precise motion tracking and orientation detection. It's ideal for applications like robotics, motion capture, and navigation systems.

Key Specifications and Features:

- Sensor Type: 9-axis (3-axis accelerometer, 3-axis gyroscope, 3-axis magnetometer)
- Communication Interface: I2C, SPI
- Output: Provides data on angular velocity, acceleration, and magnetic field
- Power Supply: 3.3V to 5V

Advantages:

- Combines three sensors (accelerometer, gyroscope, magnetometer) into one module for accurate 3D motion tracking.
- Offers high precision and low drift for reliable orientation data.
- Small form factor, easy to integrate into various projects.

Disadvantages:

- Requires proper calibration to ensure accuracy.
- May require additional processing power to handle data from multiple sensors.

5. OLED Display

Overview:

The OLED display is a small and high-quality screen used for displaying information or output. It is commonly used in various DIY electronics projects and offers vibrant colors and high contrast.

Key Specifications and Features:

- Size: Typically, 0.96 inches
- Resolution: 128x64 pixels
- Interface: I2C or SPI
- Power Supply: 3.3V or 5V

Advantages:

- High contrast, making it readable in various lighting conditions.
- Low power consumption, ideal for portable projects.
- Simple interface for easy integration with microcontrollers like Raspberry Pi.

Disadvantages:

- Limited display area (small size).

- May have limited viewing angles compared to other displays.

6. Push Button

Overview:

A momentary switch that sends a signal when pressed, often used for user interaction in various embedded systems or IoT projects.

Key Specifications and Features:

- **Type: Normally open**
- **Actuation: Push to make**
- **Size: Compact design**
- **Durability: Long lifespan with millions of presses**

Advantages:

- **Simple, reliable interface for user input.**
- **Easy to integrate into projects.**

Disadvantages:

- **Limited functionality; typically used for simple on/off signals.**
- **Mechanical wear over time can degrade performance.**

7. Buzzer

Overview:

The buzzer emits sound signals, commonly used for notifications, alerts, or audible feedback in electronic circuits.

Key Specifications and Features:

- **Type: Active**
- **Operating Voltage: 3V to 5V**
- **Sound Output: Audible tones**

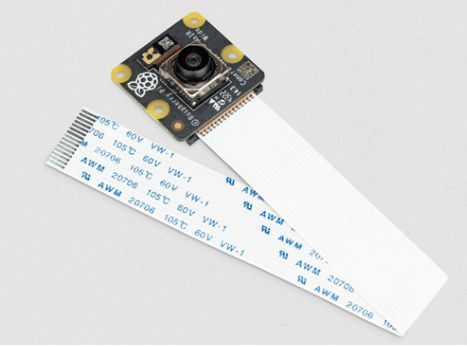
Advantages:

- **Simple to use and integrate into projects.**
- **Provides immediate auditory feedback.**

Disadvantages:

- **Can be loud or intrusive depending on the application.**
- **Limited in functionality (sound only).**

8. Raspberry Pi Camera Module 3 Wide



Overview:

The Raspberry Pi Camera Module 3 Wide is an upgraded camera module

designed for the Raspberry Pi platform, offering a wider field of view and improved image quality. It's perfect for applications requiring broad coverage, such as surveillance, drones, and outdoor robotics.

Key Specifications and Features:

- Resolution: 12 MP
- Field of View: Wide-angle lens (ideal for broader coverage)
- Video Capture: Up to 60 fps at 1080p
- Interface: CSI (Camera Serial Interface)
- Lens Type: Wide-angle lens with fixed focus

Advantages:

- Wider field of view provides better coverage for applications like security cameras or environmental monitoring.
- High-resolution imaging captures more details.
- Supports up to 60 fps video capture for smooth video performance.

Disadvantages:

- Fixed-focus lens limits depth-of-field adjustment for some close-up applications.
- Requires proper lighting for optimal image clarity in low-light conditions.

9. Step-Up Converter XL6019

Overview:

The XL6019 is a step-up converter that increases lower input voltages to higher output voltages, providing stable power for systems that require more voltage than the source can supply.

Key Specifications and Features:

- Input Voltage Range: 5V to 32V
- Output Voltage: 5V to 35V (adjustable)
- Current Capacity: Up to 4A
- Efficiency: Up to 90%

Advantages:

- Provides efficient voltage conversion for various power-hungry components.
- Adjustable output allows for versatile usage.

Disadvantages:

- Requires careful design to ensure stable power delivery at high currents.
- Can generate heat under high load.

10. RGB LED

Overview:

An RGB LED allows for displaying multiple colors by mixing red, green, and blue light. It's used in various electronics projects where color indication or display is needed.

Key Specifications and Features:

- Size: Standard 5mm
- Voltage: Typically, 3.2V to 3.4V per color
- Control: PWM (Pulse Width Modulation) for color mixing

Advantages:

- Capable of displaying a wide range of colors.
- Small and easy to integrate into projects.
- Power-efficient when using low current.

Disadvantages:

- Limited color resolution compared to full displays.
- Requires careful control for accurate color mixing.

11. Battery Nihewo 6500 mAh, 90C

Overview:

The Nihewo 6500 mAh, 90C battery is a high-capacity LiPo battery designed for high-demand applications like drones, robotics, and other power-hungry devices.

Key Specifications and Features:

- **Capacity: 6500 mAh**
- **Voltage: 7.4V**
- **Discharge Rate: 90C (high current output)**

Advantages:

- **High discharge rate supports power-hungry applications.**
- **Large capacity ensures longer run times.**

Disadvantages:

- **LiPo batteries can be sensitive to overcharging and overheating.**
- **Requires proper care and maintenance to ensure longevity.**

Motors Details

Overview:

The motors are critical components in any robotic or automated system, providing movement and control. These motors may include servo motors, DC motors with encoders, and other types of actuators that enable precise movement and positioning.

Key Specifications and Features:

- Servo Motor: Typically used in applications requiring precise angle adjustments and torque.
- DC Motor with Encoder: Used for precise rotational control, often used in wheels or gears.
- Motor Driver: Controls the power sent to motors, ensuring smooth operation and control.

Advantages:

- Provide precise control over mechanical movement.
- Can be easily integrated with microcontrollers and drivers for automation tasks.

Disadvantages:

- Motors can draw significant current, affecting power consumption and system performance.
- Motors with encoders require proper calibration to ensure accurate readings.

12. Servo Motor max 40kg max torque

Overview:

This high-torque servo motor is designed for heavy-duty tasks, providing a maximum torque of 40kg. It is suitable for robotic arms, large-scale automation, and other applications that demand powerful actuation.

Key Specifications and Features:

- Torque: 40 kg-cm
- Power Supply: Typically 6V to 12V
- Rotation Range: 180°
- Applications: Robotics, mechanical systems requiring precise and powerful movement.

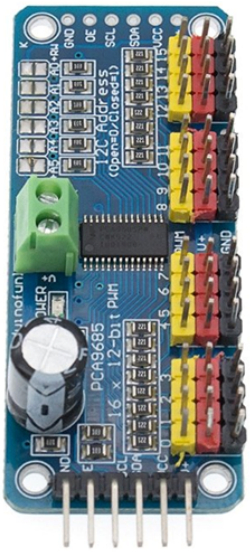
Advantages:

- Capable of handling heavy loads with high precision.
- Ideal for tasks such as robotic arms and larger robotic systems.

Disadvantages:

- Larger and heavier than standard servo motors.
- Consumes more power due to the high torque.

13. Servo Motor Driver PCA9685PW



Overview:

The PCA9685PW is a motor driver used to control multiple servo motors, allowing precise control for up to 16 servos simultaneously. This driver is often used in robotics, automation, and other projects requiring multiple actuators.

Key Specifications and Features:

- Number of Channels: 16
- Interface: I2C communication
- Control: PWM (Pulse Width Modulation) for precise motor control
- Voltage: Typically operates with 5V power supply.

Advantages:

- Allows for the control of multiple servos with minimal wiring.
- I2C interface enables easy communication with microcontrollers like Raspberry Pi.

Disadvantages:

- Limited by the number of channels (up to 16).
- May require additional power regulation for larger systems with many motors.

14. DC Motor with Encoder 620 RPM

Overview:

This DC motor with encoder provides precise control of rotational speed and direction, making it ideal for applications like robotics, conveyor systems, or any system requiring accurate movement feedback.

Key Specifications and Features:

- Speed: 500 RPM
- Voltage: Typically, 12V
- Encoder: Provides feedback for position and speed control
- Applications: Robotics, industrial automation, CNC machines.

Advantages:

- Encoder feedback allows for precise control over motor speed and direction.
- Reliable and efficient for various automation tasks.

Disadvantages:

- Requires additional circuitry to process encoder signals and control the motor.
- Higher current draw may result in power supply demands.

15. IBT-4 Arduino DC Motor Driver

Overview:

The IBT-4 is a motor driver used to control high-power DC motors, offering robust performance in driving large motors that require higher voltages and currents, suitable for robotics and vehicle applications.

Key Specifications and Features:

- Motor Voltage Range: 5V to 36V
- Current: Up to 43A peak
- Interface: Typically used with Arduino or other microcontrollers.
- Control Type: PWM for speed and direction control.

Advantages:

- Can control large motors, ideal for high-torque applications.
- Easy to integrate with Arduino and other microcontroller platforms.

Disadvantages:

- Can generate heat under heavy load, requiring adequate cooling.
- Requires careful handling due to high current capabilities.

Description of Vehicle Movement System

The robot uses a combination of DC motors for forward motion and servo motors for steering. Motor encoders provide feedback for precise control. Need more descriptions

Designing the Mechanical Alignment

i. Conceptualization and CAD Modeling

1. The team began by visualizing the robot's mechanical layout, focusing on optimizing component placement for balance and stability and sketching the design using basic drawing and understanding aliment control system and how this importance for impacting the vehicle movement balance while steering the vehicle on low speed and high speed, and vibration controls.
2. The Team studied basic vehicle Stability Control System front wheels by learning from the research "Kinematic Control System for Car-Like Vehicles" file:/C:/Users/alkha/Downloads/Kinematic_Control_System_for_Car-Like_Vehicles.pdf and "Kinematics and Control for Wheeled Robots" https:/ucr-robotics.readthedocs.io/en/latest/tbot/moRbt.html
3. **CAD Software:**
   a. Advanced CAD tools such as **Fusion 360** were used to create 3D models of the robot parts.
   b. These models included detailed representations of all components, including the chassis, wheels, servo motors, and sensors.
4. **Key Factors in Alignment**:
   a. Weight Distribution and Balance:

i. Components like the battery and Raspberry Pi were centrally placed to lower the center of gravity and improve stability, also with our basics scaling for wait balance we used small digital scale, we scaled all parts first and all the components together after assembling, the net weight is 1,387 gm

b. **Motor Placement**:

i. DC motors and wheels were aligned symmetrically to ensure consistent torque delivery and straight-line movement, The connection between the back wheel by steel piler (SIZE)

c. **Servo Integration**:

i. The steering mechanism was carefully aligned to allow smooth rotation within the operational range of the servo motor, and applying most of theory into practical using mentioned reference

ii. Iterative Design and Prototyping

1. The initial CAD design underwent multiple iterations to address potential alignment issues. Example where to mount and joints place should tack place and where to screws wholes should go

2. Prototypes of the alignment were 3D-printed to verify the fit and functionality of components. Adjustments were made based on physical testing results.

iii. Precision Measurements

1. Dimensions were measured with high accuracy to ensure all parts fit perfectly, the from wheels from left to right screws were holding the wheel its 8 centimeters with center point of 4 centimeters, from the back its 12 centimeters
2. Tolerances were accounted for to allow for slight adjustments during assembly while maintaining structural integrity.

iv. **Custom Mount Design**

a. **Dedicated Mounting Bracket**: A custom bracket was designed in the CAD Fusion 360 software to securely hold the servo motor in place. This bracket:

1. Was tailored to fit the specific dimensions of the 40kg max torque servo motor.
2. Included screw holes and slots for precise positioning and easy adjustment.

b. **Material Selection**:

i. The mount was 3D-printed using durable materials such as PLA+, ensuring it could withstand the forces exerted during operation.

v. **Integration with the Chassis**

1. The servo mount was integrated into the chassis design:
   a. Positioned close to the robot's steering mechanism for direct linkage with minimal lag.
   b. Fixed using bolts and locknuts to prevent movement or vibration during operation.

vi. **Steering Linkage Alignment**

1. **Direct Coupling**:
   a. The servo horn was connected to the steering linkage, designed to convert the servo's rotational motion into precise wheel angles.
2. **Adjustable Linkage Rods**:
   a. The linkage rods were adjustable, allowing fine-tuning of steering angles to achieve maximum efficiency.
3. **Testing and Calibration**:
   a. The steering system was tested repeatedly to ensure smooth operation and accurate alignment.

vii. **Wiring and Power Management**

1. The servo motor was connected to the **PCA9685 Servo Motor Driver**, which provided precise control of servo positions.
2. Power was managed through the robot's centralized power system, with voltage regulation from the **Step-Down Converter** to ensure stable operation.

Ensuring Long-Term Durability

viii. **Reinforcements**:

1. Additional supports were added around the servo mount to absorb shocks and vibrations during movement.

ix. Regular Testing:

1. The servo was tested under different loads and conditions to verify its durability and responsiveness**.**

x. **Lubrication and Maintenance**:

1. The steering linkage was periodically checked and lubricated to maintain smooth movement.

Key Benefits of the Design

xi. **Alignment Accuracy**: The 3D-printed models ensured precise alignment, minimizing mechanical stress and improving performance.

1. **Ease of Assembly**: The modular design allowed the servo motor to be mounted and replaced quickly without disassembling the entire chassis.
2. **Performance Efficiency**: The integration of the servo with the steering system provided high torque and precise control, enabling accurate navigation in competitive tasks.

Implementing this strategy by leveraging advanced 3D modeling and precise engineering; the team successfully created a robust mechanical alignment

and servo mounting system that contributed significantly to the robot's overall functionality and reliability.

Motor Selection and Specifications

The team's selection and implementation of motors and motor drivers were guided by the requirements for efficient movement, precise control, and reliability under competitive conditions. Here's a detailed explanation of how the team approached this aspect:

xii. **DC Motor with Encoder: 500 RPM, High-Torque Motor**

xiii. **Selection Criteria:**

1. **Torque Requirements**: The team opted for high-torque motors to ensure the robot could handle inclines, overcome obstacles, and carry the weight of all mounted components without compromising speed.
2. **Speed Optimization**: A motor with a **500 RPM rating** was chosen, striking a balance between sufficient speed for navigation and controlled motion for obstacle avoidance and precision tasks.
3. **Encoder Integration**: Motors with built-in encoders were selected to provide feedback on rotation and speed, essential for:
    a. Ensuring accurate movement and stopping.
    b. Synchronizing wheel rotations for straight-line motion and consistent turns.

xiv. **Implementation**

1. **Placement**: The DC motors were mounted symmetrically on either side of the chassis to ensure even power distribution and stability.

2. **Power and Control**: The motors were connected to the **IBT-4 DC Motor Driver**, which could handle high current loads and provided smooth, responsive control.
3. **Feedback Utilization**:
    a. The encoder signals were processed by the Raspberry Pi to calculate wheel rotation and adjust motor speeds dynamically.
    b. Encoder data was also used to improve navigation accuracy during challenges.

Servo Motor: 40kg Max Torque for Steering Control

xv. Selection Criteria:

1. **Torque Capacity**: A servo motor with a maximum torque of **40kg** was selected to handle the demands of steering the robot's wheels, especially under high loads or during sharp turns.
2. **Precision**: Servos are inherently designed for precise angle control, making them ideal for steering applications where accuracy is critical.

xvi. Implementation

1. **Custom Mounting**: The servo motor was securely mounted using a custom 3D-printed bracket that ensured stability while allowing for easy adjustments during testing.
2. **Steering Linkage**:
    a. The servo horn was connected to a linkage system designed to translate the servo's rotational motion into precise angular movement of the wheels.
    b. Adjustable linkage rods allowed for fine-tuning steering angles to optimize performance.
3. **Control**:

a. The servo motor was controlled via the **PCA9685 Servo Motor Driver**, which provided accurate PWM signals for fine control over the steering mechanism.
b. Real-time adjustments were made based on inputs from the Raspberry Pi and sensor feedback.

Motor Drivers

IBT-4 DC Motor Driver

Current Handling:

The IBT-4 motor driver was chosen for its ability to handle the high current demands of the DC motors without overheating or performance loss.

Bidirectional Control:

Enabled precise forward and reverse control of the motors, essential for tasks like parking and obstacle avoidance.

Integration:

The IBT-4 driver was connected to the Raspberry Pi, which provided control signals based on programmed logic and sensor feedback.

PCA9685 Servo Motor Driver

Multi-Channel Capability: Allowed control of multiple servos simultaneously, making it ideal for applications with complex motion systems.

Precision: Provided stable and accurate PWM signals, ensuring smooth and precise servo movement for steering.

Ease of Use: Its compatibility with the Raspberry Pi ensured seamless integration and reduced development time.

Testing and Refinement

a. **Initial Testing**:

1. The team tested each motor individually to validate performance, torque, and responsiveness.
2. Encoder functionality was verified to ensure accurate feedback for navigation.

b. **Integration Testing**:
    1. Both DC motors and the servo were tested together to ensure smooth coordination between movement and steering.

c. **Adjustments**:
    1. PID controllers were implemented in the code to fine-tune motor speeds and ensure precise stopping and turning.
    2. The steering linkage was adjusted to minimize dead zones and maximize turning efficiency.

d. Achievements
    1. The combination of high-torque DC motors and a powerful servo motor ensured the robot could navigate complex environments with precision and reliability.
    1. The motor drivers provided stable and efficient control, contributing to the overall success of the robot in meeting competition requirements.

After consulting our mentor, we carefully selected and implementing these components, we created a robust and responsive motion system, perfectly suited to the challenges of the WRO Future Engineering competition.

Chassis Design and Mounting of Components

The team's approach to chassis design and component mounting was grounded in precision engineering, leveraging modern tools and techniques to create a structure that was lightweight, compact, and stable. Here's a

comprehensive explanation of how the team addressed this crucial aspect of the robot:

1. Chassis Design

a. Lightweight and Compact Structure

- **Material Selection**:
  - The chassis was constructed using lightweight yet durable materials like aluminum alloy or high-grade plastics such as ABS.
  - This choice reduced the overall weight of the robot, enhancing its maneuverability without compromising strength.
- **Compact Dimensions**:
  - The chassis was designed to house all components efficiently within a small footprint, ensuring the robot could navigate tight spaces and meet competition size constraints.
  - Special attention was paid to minimizing excess space while maintaining accessibility for maintenance and adjustments.

CAD Software Utilization

- **Software Tools**:
  - Advanced CAD software, such as Fusion 360 or SolidWorks, was used to design the chassis.
  - The team created detailed 3D models that included precise dimensions, component placement, and mounting holes.
- **Iterative Design Process**:
  - Multiple iterations of the chassis were developed and tested virtually to ensure compatibility with all components and to optimize weight distribution.

- Stress simulations were conducted in the CAD software to identify and reinforce potential weak points.

c. Structural Integrity

- **Reinforcement**:
  - Key load-bearing areas, such as motor mounts and component brackets, were reinforced to handle operational stresses.
  - Cross-beams or support plates were incorporated into the design to prevent flexing or deformation during movement.

- **Cable Management**:
  - Built-in cable routing channels were included in the design to keep wiring neat and prevent interference with moving parts.

Mounting of Components

1. Balanced Weight Distribution

7. **Centralized Components**:
   a. Heavy components, such as the battery and motor drivers, were placed near the center of the chassis to lower the center of gravity and improve stability.
   b. This arrangement minimized the risk of tipping during turns or while navigating uneven terrain.
8. **Symmetrical Layout**:
   a. Components were distributed symmetrically to ensure balanced weight on all wheels, enabling smooth and consistent movement.
9. **Layered Design**:
   a. The chassis was divided into multiple levels to separate components:

i. **Top Layer**: Electronics like the Raspberry Pi, camera module, and display were mounted here for easy access.
ii. **Middle Layer**: Sensors and wiring were housed in this layer for protection and organization.
iii. **Bottom Layer**: Motors, battery, and heavy components were mounted here to maximize stability.

2. Custom Mounting Solutions

- **3D-Printed Mounts**:
  - Custom brackets and holders for components such as the Raspberry Pi, sensors, and motor controllers were designed and 3D-printed.
  - These mounts ensured a secure fit while accommodating specific component dimensions.
- **Modularity**:
  - The mounting system was designed to be modular, allowing individual components to be easily removed and replaced without disassembling the entire robot.
- **Shock Absorption**:
  - Components like the Raspberry Pi were mounted on vibration-dampening pads to protect sensitive electronics from shocks and vibrations during operation.

Fastening Mechanisms

- **Bolts and Locknuts**:
  - Key components were fastened using bolts and locknuts to ensure a firm attachment that would not loosen over time.
- **Quick-Release Clips**:

    - For frequently accessed parts, quick-release clips were used to speed up assembly and maintenance.
- **Velcro and Straps**:
    - The battery and other large components were secured using Velcro straps, allowing for quick removal while keeping them firmly in place during operation.

3. Final Testing and Adjustments

a. Physical Prototyping

- After virtual simulations, the chassis was 3D-printed or CNC-machined for physical testing.
- Components were mounted, and the robot was subjected to various stress tests to ensure the chassis maintained structural integrity.

b. Stability Tests

8. Weight distribution was fine-tuned by adjusting component placement.
9. Stability during sharp turns and while climbing inclines was tested to verify that the robot's center of gravity remained low and balanced.

4. Key Outcomes

- **Lightweight Efficiency**: The use of lightweight materials ensured the robot could achieve high speeds without overloading the motors.
- **Compact and Stable**: The compact design enabled smooth navigation in confined spaces, while the balanced weight distribution provided excellent stability.

- **Modular and Accessible**: The modular component mounting system made the robot easy to maintain and upgrade, enhancing its long-term reliability.

By combining advanced CAD design, thoughtful weight distribution, and innovative mounting techniques, the team created a robust chassis that met all functional and competitive requirements.

Engineering Principles Applied

- The team's approach to motor and gear selection was rooted in fundamental engineering principles, with calculations for speed, torque, and power serving as the foundation for designing an efficient and high-performing robotic system. Here's how these principles were applied:

1. Speed Calculations

- **a. Determining Optimal Speed**
    - **Competition Requirements**: The robot needed to navigate the challenge course quickly while maintaining precision. The team analyzed the maximum speed the robot could achieve without compromising obstacle detection and steering accuracy.
    - **Linear Speed (v)**:
        - Formula:
            - $v = r \times \omega$
        - Where:
            - r = Radius of the wheels
            - ω = Angular velocity of the wheels (derived from motor RPM and gear ratio)

    - The desired linear speed was calculated to ensure the robot could complete tasks efficiently while maintaining control.
- **b. Gear Ratio Impact**
    - The team selected a gear ratio that reduced the motor's high RPM to a manageable wheel speed, ensuring smooth navigation without overloading the motors.

2. Torque Calculations

- **a. Analyzing Torque Requirements**
    - **Load Considerations**: The torque required was calculated based on the robot's weight, wheel radius, and the forces it would encounter (e.g., friction, inclines).
        - **Torque (τ\tauτ)**:
            - Formula: τ=F×r\tau = F \times rτ=F×r
        - Where:
            - FFF = Force required to move the robot
            - rrr = Radius of the wheels
    - The team ensured the selected motors provided sufficient torque to overcome resistive forces while maintaining smooth operation.
- **b. High-Torque Motor Selection**
    - The team selected DC motors with encoders capable of delivering high torque (500 RPM motors) to ensure:
    - The robot could carry its payload without stalling.
    - Smooth starts and stops, especially when navigating obstacles or inclines.
-

3. Power Calculations

- **a. Motor Power Requirements**
    - **Power Output (P)**:
        - Formula: P=τ×ωP = \tau \times \omegaP=τ×ω
        - Where:
            - τ\tauτ = Torque
            - ω\omegaω = Angular velocity
        - Calculations ensured the selected motors provided adequate power to maintain the desired speed and torque under load.
- **b. Power Supply Considerations**
    - The **Nihewo 6500mAh 90C LiPo battery** was chosen to provide sufficient power for the motors and other electronic components.
    - The team ensured the battery voltage matched the motor driver's requirements, with additional regulation provided by step-up and step-down converters.

4. Gear Selection

- **a. Balancing Speed and Torque**
    - The team selected a gear ratio that optimized the balance between speed and torque:
    - Lower gear ratios favored higher speed but reduced torque.
    - Higher gear ratios provided more torque for carrying payloads or climbing inclines but reduced speed.
    - For this project, a mid-range gear ratio was chosen to meet the dual demands of speed and torque efficiently.
- **b. Custom Gear Design**
    - **Bevel Gears**: Custom 3D-printed bevel gears (2–3 cm diameter) were used for the drivetrain, ensuring durability and precision.

- The gear teeth were designed to minimize wear and provide smooth power transfer from the motor to the wheels.

5. Iterative Testing and Refinement

- **a. Initial Simulations**
  - The team used CAD and simulation software to model the robot's performance under various loads and conditions.
  - Simulated torque and speed values were compared against theoretical calculations to verify accuracy.
- **b. Physical Testing**
  - Once the robot was assembled, the motors and drivetrain were tested in real-world scenarios:
    - Navigating obstacles.
    - Climbing inclines.
    - Performing sharp turns and sudden stops.
  - Adjustments were made to the gear ratio and wheel size based on performance data.

6. Key Achievements

- **a. Optimized Performance**
  - The robot achieved a balance of speed and torque, enabling it to navigate challenges quickly while handling obstacles and inclines effortlessly.
- **b. Efficient Power Usage**
  - Calculations ensured that the motors operated within their optimal power range, maximizing efficiency and minimizing heat generation.
- **c. Robust and Reliable Operation**

- By aligning motor and gear selection with calculated requirements, the team created a system that was both powerful and durable, capable of sustaining long periods of operation without failure.

7.Obstacle Management

The team employed a structured and logical approach to navigate obstacles, combining advanced computer vision techniques, real-time sensor feedback, and well-optimized decision-making processes. Here's how the team tackled obstacle management:

1. Strategy for Navigating Challenges

a. Detection with OpenCV

- The **Raspberry Pi Camera Module 3** served as the primary vision sensor for obstacle detection.
- Using the **OpenCV library**, the robot processed real-time images to:
  - Identify the colors and shapes of objects on the course.
  - Differentiate between **red** and **green** blocks to determine the appropriate navigation response.

b. Navigation Logic

- **Color-Based Turns**:
  - **Red Blocks**: Detected red blocks triggered a **right turn** to avoid collisions.
  - **Green Blocks**: Detected green blocks prompted a **left turn** to navigate around the obstacle.
- **Line Detection**:
  - The robot identified lines on the course using OpenCV, ensuring it stayed on track and adjusted direction when required.

c. Real-Time Feedback Integration

- **Gyroscope**:
    - Provided orientation data to ensure the robot maintained its intended direction after maneuvering around obstacles.
    - Used to correct deviations caused by sudden turns or uneven terrain.
- **Encoder Data**:
    - Motor encoders tracked wheel rotations, ensuring precise distance measurement and accurate turns.

d. Testing and Refinement

- Extensive testing was conducted to identify edge cases, such as overlapping red and green blocks or low lighting conditions.
- Adjustments were made to the vision processing algorithms to improve detection accuracy under varying conditions.

2. Flowcharts for Movement and Decision Logic

a. Purpose of Flowcharts

- Flowcharts were developed to visually represent the robot's decision-making processes, ensuring clarity and consistency in logic implementation.
- These diagrams provided a step-by-step guide for:
    - Navigating obstacles.
    - Deciding the next move based on sensor inputs.
    - Performing precise parking maneuvers.

b. Key Elements of the Flowchart

- **Input Nodes**:
    - Camera data (block color and position).
    - Gyroscope and encoder feedback (orientation and distance).

- **Decision Nodes**:
    - Conditional checks for block color:
        - If red, execute a right turn.
        - If green, execute a left turn.
    - Line detection:
        - Adjust movement to stay on course.
- **Output Nodes**:
    - Motor commands for forward motion, turning, or stopping.

c. Iterative Development

- The flowcharts were refined over multiple iterations to account for edge cases and ensure robustness.
- The diagrams were shared among team members to facilitate clear communication and collaborative debugging.

3. Pseudo-Code and Source Code with Comments

a. Pseudo-Code

The team wrote detailed pseudo-code before actual implementation to outline the logic in a structured format.

Code Features

- **Modularity**:
    - Functions were defined for specific tasks like detect_color, turn_right, turn_left, and adjust_direction.
- **Comments**:
    - Each function and key logic block included detailed comments explaining the purpose and behavior, aiding debugging and future development.
- **Error Handling**:

- Built-in checks for cases where no obstacles were detected or sensors provided ambiguous readings.

4. Testing and Iterative Refinement

a. Testing Scenarios

- The robot was tested in various environments to evaluate its obstacle management capabilities, including:
  - Courses with multiple obstacles of different colors.
  - Poor lighting or shadows affecting camera input.
  - Sudden changes in the robot's path (e.g., unexpected curves).

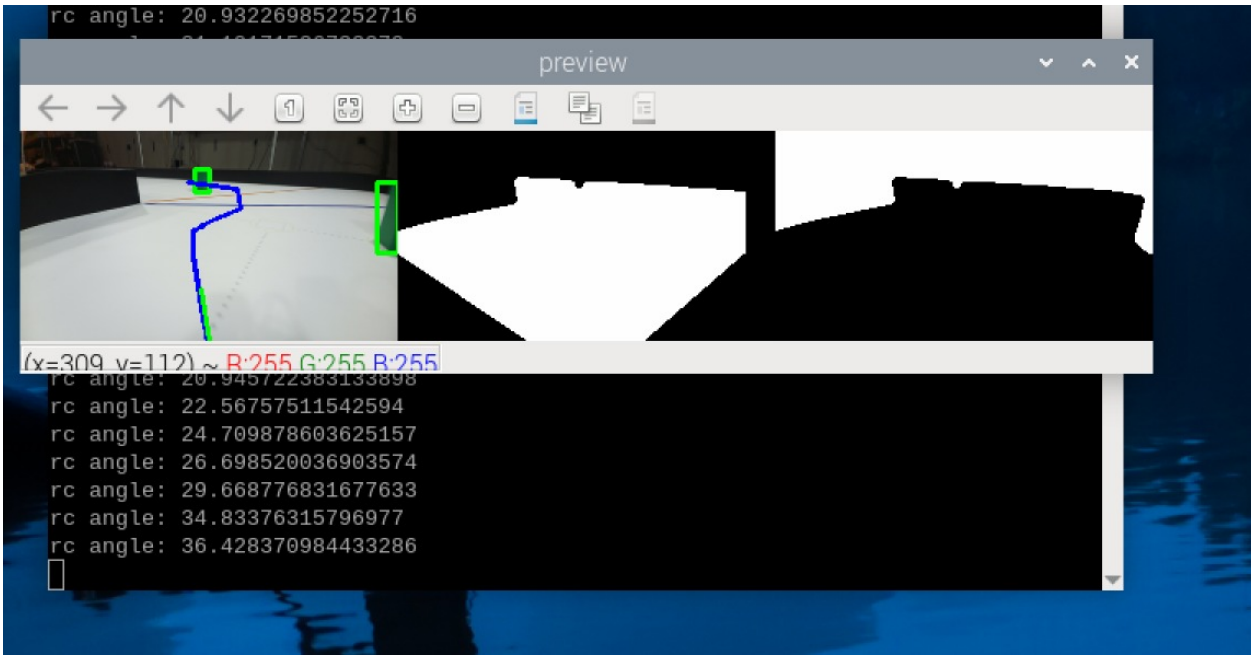
b. Feedback and Adjustments

- Vision processing thresholds were adjusted to improve color detection accuracy.
- Gyroscope sensitivity and encoder calibration were fine-tuned to ensure smooth and precise navigation.

5. Outcomes

- **Accurate Obstacle Avoidance**: The robot reliably identified red and green blocks and executed the correct navigation response.
- **Seamless Integration**: Real-time sensor feedback ensured smooth and consistent movement around obstacles.
- **Robust Logic**: The flowcharts and modular code structure allowed for quick debugging and enhancements.

By combining advanced computer vision, real-time sensor data, and a clear logical framework, the team achieved a robust obstacle management system capable of handling complex challenges efficiently.


rc angle: 20.932269852252716
preview
(x=309, y=112) ~ R:255 G:255 B:255
rc angle: 20.945722383133898
rc angle: 22.56757511542594
rc angle: 24.709878603625157
rc angle: 26.698520036903574
rc angle: 29.668776831677633
rc angle: 34.83376315796977
rc angle: 36.428370984433286

8.Design and Prototyping

The design and prototyping phase of the robot's development followed an iterative and structured approach, with a focus on refining individual subsystems and integrating them into a cohesive and functional design. The process began with conceptualizing the robot's overall structure and performance requirements, which were then broken down into smaller subsystems for focused development and testing. This iterative methodology allowed the team to identify potential issues early on and implement improvements in subsequent prototypes.

One of the most critical aspects of the design process was the careful selection of components. The team prioritized high-performance and reliable parts, such as the Raspberry Pi 5, which served as the central processing unit, and the Raspberry Pi Camera Module 3, which provided the essential vision capabilities required for tasks like obstacle detection and navigation. These components were chosen not only for their advanced features but also for their proven track records in computational efficiency and compatibility with the project's requirements. Their inclusion ensured that the robot could handle intensive tasks such as image processing and real-time decision-making.

Prototyping brought its own set of challenges, particularly in managing the complexity of the wiring. The initial designs, while functional, had an unorganized layout that led to difficulties in maintenance and increased the risk of short circuits. To address this, the team refined the wiring scheme by introducing structured layouts, modular connection points, and effective cable management strategies. These refinements greatly improved the robot's reliability and made future iterations easier to assemble and troubleshoot.

The evolution of the robot's design was marked by a progression from basic functional prototypes to a polished, competition-ready machine. Early versions were focused primarily on validating core functionalities such as motor control, sensor integration, and obstacle detection. These prototypes provided

invaluable insights into the system's behavior, guiding improvements in both hardware and software. As the project advanced, the team refined the chassis design for better stability, optimized the placement of components for balanced weight distribution, and enhanced the robot's overall aesthetics and durability. The final design reflected a culmination of these efforts, achieving a harmonious blend of performance, reliability, and user-friendly features.

- This thorough and iterative approach to design and prototyping allowed the team to develop a robot capable of meeting the demands of the competition while ensuring ease of maintenance and adaptability for future enhancements.

- **CAD Files and Assembly Instructions**
    - The team leveraged advanced CAD (Computer-Aided Design) tools to create detailed schematics and assembly instructions for the robot, ensuring precision, structural integrity, and ease of maintenance. Here's how the team worked on this aspect:

1. Development of CAD Designs

a. Design Tools and Software

- The team used industry-standard CAD software such as **Fusion 360** or **SolidWorks** to design the robot's components and overall structure.
- The software allowed for:
    - Precise modeling of individual parts.
    - Simulation of component interactions and stress analysis.
    - Realistic 3D visualization of the complete robot.

b. Detailed Schematics

- **Component Models**: Each major component, including the chassis, motor mounts, brackets, and wiring channels, was modeled individually to exact specifications.
- **Dimensional Accuracy**: All parts were dimensioned precisely to ensure compatibility during assembly.

- **Layered Structure**:
  - The design featured multiple layers to separate components like electronics, sensors, and motors for easier access and reduced interference.
  - **Exploded Views**: Exploded diagrams were created to visualize how the components fit together, aiding in understanding the assembly process.

c. Iterative Design Process

- The CAD designs underwent multiple iterations based on testing and feedback to:
- Improve fit and functionality.
- Address issues like weight distribution, interference between components, or structural weaknesses.

- Simulations in the CAD software were used to predict performance under various conditions and refine the design.

2. Structural Integrity and Stability

- **a. Reinforcement Features**
  - Key areas, such as motor mounts and load-bearing joints, were reinforced with additional material to handle stresses during operation.
  - Cross-beams and brackets were added in critical locations to prevent flexing or deformation.

- **b. Weight Distribution**
    - The CAD designs prioritized placing heavy components, such as the battery and motors, near the center of gravity to enhance stability.
    - Lightweight materials were incorporated into less critical areas to reduce overall weight without compromising strength.

3. Assembly Instructions

- **a. Modular Design**
    - The robot was designed with modularity in mind, allowing individual components or subsystems to be assembled, removed, or replaced independently.
    - Each module (e.g., drivetrain, electronics, steering mechanism) was detailed separately in the assembly instructions.
- **b. Step-by-Step Guide**
    - The team created step-by-step assembly instructions, including:
    - **Tools Required**: A list of tools, such as screwdrivers, Allen keys, and wrenches, needed for assembly.
    - **Fastening Instructions**: Details on how to secure components using bolts, nuts, or clips.
    - **Cable Routing**: Diagrams showing the path for wiring to prevent tangling and interference with moving parts.
- **c. Labeling and Markings**
    - Components in the CAD files were labeled clearly, matching physical parts to the instructions.
    - Markings on the chassis or mounts indicated alignment points for accurate assembly.

9. Coding

Code Explanation 1:1

This Python code implements an autonomous robot control system, integrating hardware such as a Raspberry Pi, camera, motors, sensors, and servos to navigate a track and avoid obstacles. Below is a detailed explanation of the key components and logic within the code:

1. Libraries and Initialization

Imported Libraries

- cv2 and numpy: For image processing and numerical operations.
- Picamera2: Used to control the Raspberry Pi Camera Module for capturing frames.
- Custom Modules (process_frames, motor_controller, servo_controller, BnoSensor_controller): These likely define helper functions to process images, control motors, servos, and sensors.
- collections and datetime: For maintaining smoothed angle buffers and logging.
- os and time: For file operations and handling delays.

Camera Configuration

The Picamera2 is initialized with a specific resolution:

- cam_res: Defines the camera resolution.
- video_config: Configures video mode and formats for capturing frames.

2. Steering and Sensor Logic

Logging Steering Angles

- The function log_steering_angle() records smoothed steering angles with timestamps into a text file. This log is useful for debugging and performance analysis.

Gyroscope Integration

- Functions _get_raw_rc_angle() and get_rc_angle() interact with the gyroscope to fetch and normalize yaw angles.
- These angles help the robot maintain its orientation and align with the track.

3. Stuck Detection Logic

Stuck Detection Mechanism

The stuck_checker class:

- Compares consecutive frames using the absolute difference method.
- Checks if the robot is "stuck" by evaluating similarity over time:
    - If frames remain similar beyond a threshold and duration, the robot assumes it's stuck.
    - Appropriate recovery actions are initiated, such as reversing or changing steering angles.

Similarity Measurement

- Frames are converted to grayscale for comparison.
- Structural differences are computed, and a similarity ratio is derived. If the ratio exceeds a predefined threshold, the robot considers itself stuck.

4. Main Logic

Hardware Initialization

- Motors: The create_dc_motor() initializes the motors for movement.
- Servo: Controlled using create_servo() for steering with specific angle configurations.
- Gyroscope: Initialized using creat_bno_sensor() for orientation feedback.

Key Variables

- offset_fix_factor: Adjusts steering offset for track imperfections.
- steer_angle_buffer: Maintains a history of steering angles to smooth sudden variations.

Image Processing and Frame Capture

- Frames are captured using picam2.capture_array().
- The process_frame() function processes these frames to compute:
    - Offset Steering: Steering corrections based on detected track deviations.
    - Wall Points: Helps track lap progression.

5. Obstacle Avoidance and Navigation

Steering Logic

- The steering angle is computed based on the offset detected by the camera.
- Smoothed angles (averaged from steer_angle_buffer) are set to the servo using servo.set_angle().
- log_steering_angle() records each steering angle for later analysis.

Obstacle Avoidance

- If the robot detects it is stuck (using check_if_stuck()), it:
    - Reverses direction by flipping the steering angle.
    - Activates recovery logic for a specified duration.

Motor Control

- motor.set_speed(run_speed) adjusts the speed dynamically based on detected conditions (e.g., obstacles or stuck scenarios).

6. Lap Counting Logic

Lap Detection

- The robot monitors the wall point relative to its initial position to track lap completion.
- After every few laps (e.g., 3 laps), the robot halts and exits the loop.

Gyroscope Assistance

- Lap counting ensures the robot is aligned properly using gyroscope feedback to verify angles.

7. Recovery and Cleanup

Safe Exit

- On exiting the loop or in case of errors, the code:
    - Stops the camera and motors.
    - Resets the servo to its center position.
    - Releases resources like windows and hardware locks.

Overall Workflow

1. Initialization: Set up motors, servo, camera, and sensors.
2. Frame Capture: Continuously capture frames from the camera.
3. Obstacle Avoidance:
    - Process each frame to calculate offset steering.
    - Adjust steering and speed dynamically.
    - Detect and recover from stuck situations.
4. Lap Counting: Monitor and count laps using wall points and gyroscope data.
5. Termination: Clean up and safely shut down hardware.

Key Features and Achievements

- Robust Steering Control: Smooth steering adjustments via smoothed angles and servo commands.

- Obstacle Detection: Stuck detection ensures recovery and continued navigation.
- Performance Logging: Logs steering angles for debugging and analysis.
- Lap Completion: Uses wall point tracking and gyroscope angles to manage progress.

Code Explanation 1:2

This Python script implements a TinyML-based object detection system designed to recognize obstacles using a pre-trained Edge Impulse model. It utilizes a Raspberry Pi Camera and Edge Impulse's library to perform real-time inference and detect obstacles. Here's a detailed explanation of the script:

1. Library Imports

The script imports essential libraries for handling:

- cv2 and numpy: For image processing and mathematical operations.
- edge_impulse_linux.image: To run the TinyML model using Edge Impulse's tools.
- Picamera2: For interfacing with the Raspberry Pi Camera.

2. Obstacle Detection: get_obstical_cordinates Function

This function uses a TinyML model to detect obstacles in a given frame.

Inputs:

- frame: An image frame captured from the camera.

Outputs:

- An array containing the label, x-coordinate, and y-coordinate of each detected obstacle.

Steps:

1. **Image Preprocessing:**
    - Converts the frame from BGR (OpenCV default) to RGB format for compatibility with the model.
    - Resizes and crops the image using Edge Impulse's auto studio settings (get_features_from_image_auto_studio_setings).
2. **Model Inference:**
    - The pre-trained TinyML model classifies objects within the frame, returning bounding boxes and confidence scores.
3. **Filtering Results:**
    - Bounding boxes with confidence values below 0.993 are ignored to ensure detection accuracy.
4. **Mapping to Real Coordinates:**
    - Converts bounding box coordinates and dimensions from the cropped image back to the original frame size.
    - Labels are converted into class indices for consistency (e.g., green_block = 0, red_block = 1, magenta_block = 2).
5. **Output:**
    - Appends the obstacle type and its center point (x, y) in the real image frame to the results array.

3. Main Function

The main function initializes the camera, loads the model, and continuously processes video frames for obstacle detection.

Camera Setup:

- The Raspberry Pi Camera is configured with a reduced resolution to optimize processing speed (RESIZE_FACTOR).

Model Initialization:

- The Edge Impulse model file (.eim) is loaded using the ImageImpulseRunner.

Continuous Detection Loop:

1. **Frame Capture:**
    - Frames are captured from the camera in real-time.
2. **Preprocessing:**
    - Each frame is converted to RGB and resized/cropped to match the model's input dimensions.
3. **Inference:**
    - The model processes the frame and identifies bounding boxes for detected objects.
4. **Display Results:**
    - Detected objects are drawn as rectangles on the frame with labels and confidence scores.
    - The processed frame is displayed in a debug window.

Termination:

- Pressing the q key exits the loop, stopping the camera and model runner.

4. Key Features

1. **Real-Time Obstacle Detection:**
    - Leverages Edge Impulse's TinyML capabilities to perform efficient and lightweight inference on a Raspberry Pi.
2. **High Accuracy:**

- Filters detections based on confidence values to minimize false positives.

3. **Scalable:**
    - Designed to process up to five frames per detection cycle for enhanced reliability.
4. **Visual Debugging:**
    - Displays processed frames with annotated bounding boxes for real-time visualization of detections.

5. Example Use Case

This script is ideal for a robot that needs to navigate a course by identifying and responding to obstacles like colored blocks. It provides the necessary coordinates and labels to the robot's main control logic for decision-making.

Enhancements

- Integration: Combine get_obstical_cordinates with motor control logic to enable autonomous obstacle avoidance.
- Performance Tuning: Experiment with the RESIZE_FACTOR and inference settings for improved speed.
- Additional Features: Include functionality to log detections or implement edge-case handling for overlapping objects.

Code Explanation 1:3

This Python script provides a comprehensive real-time computer vision system for detecting paths and obstacles, specifically tailored for applications like

autonomous navigation. Here's a detailed explanation of its components and functionality:

1. Core Components

1. Libraries Used:
    - cv2 and numpy: For image processing and numerical operations.
    - json and os: To handle file operations for loading and saving HSV color ranges.
    - Picamera2: To interface with the Raspberry Pi Camera.
2. HSV Color Ranges:
    - The color_ranges dictionary defines HSV (Hue, Saturation, Value) thresholds for detecting specific colors (e.g., black, green, red). These thresholds are used to segment the image into regions of interest for obstacle and path detection.

2. Loading and Saving HSV Values

- load_hsv_from_ json Function:
    - Checks for the presence of a JSON file to load predefined HSV ranges for color segmentation.
    - Updates the color_ranges dictionary dynamically, enabling adjustments without altering the script.

3. Path and Obstacle Detection

Color Detection:

- detect_color_contours Function:
    - Detects contours of a specific color within a frame by applying:
        1. Color segmentation using cv2.inRange.
        2. Gaussian blurring and thresholding for noise reduction.

3. Contour finding to detect regions matching the color.

Path Generation:

- generate_path Function:
    - Identifies the centerline of the detected track and generates a list of points representing the path.
    - These points are calculated as midpoints of white pixels in each row of the track.

Path Smoothing:

- filter_path_points Function:
    - Applies a moving average to smooth the detected path.
    - Fits a quadratic curve to the points and filters outliers based on a deviation threshold.

Contour Analysis:

- find_bottommost_contour Function:
    - Identifies the contour closest to the bottom of the image, which is critical for detecting obstacles near the robot.

4. Frame Processing

The process_frame function orchestrates the detection logic:

1. Image Preprocessing:
    - Converts the frame to grayscale and HSV formats.
    - Applies Gaussian blurring and thresholding to isolate the track and obstacles.
2. Region of Interest (ROI):
    - Defines specific areas of the frame for detecting obstacles and paths based on their expected positions.
3. Obstacle Detection:

- Identifies contours for colors like red and green, which represent obstacles.
- Updates obstacle positions and flags based on detected regions.

4. Track Filtering:
    - Processes the largest contour of the track to refine the ROI and improve path detection.
5. Path Calculation:
    - Generates a path using the filtered track and calculates an offset_steer value to guide navigation.
6. Visualization:
    - Draws contours, paths, and other relevant markers on the frame for debugging and visualization.

5. Main Function

The main function handles the overall workflow:

1. Camera Initialization:
    - Configures the Raspberry Pi Camera with a reduced resolution for optimized performance.
2. Real-Time Frame Processing:
    - Captures frames continuously and processes each frame using process_frame.
    - Displays the processed frames with overlays for obstacles, paths, and track regions.
3. Dynamic Updates:
    - Continuously adjusts regions of interest based on the detected contours and path points.
4. Exit Handling:
    - Allows the user to terminate the script by pressing q.

6. Key Features

1. Dynamic HSV Configuration:
    - HSV ranges can be saved and loaded from a JSON file, enabling quick adjustments for different environments.
2. Obstacle and Path Detection:
    - Identifies obstacles (red and green blocks) and tracks paths dynamically using color segmentation and contour analysis.
3. Smooth Navigation:
    - Uses path smoothing and deviation filtering to ensure reliable navigation even in noisy environments.
4. Real-Time Visualization:
    - Provides a live preview of the processed frames, showing detected obstacles, paths, and tracks.
5. Adaptable ROI:
    - Dynamically adjusts the ROI based on detected contours, ensuring robustness in varied scenarios.

7. Applications

- Autonomous robot navigation.
- Obstacle avoidance systems.
- Real-time path planning and visualization for vehicles or drones.

Potential Enhancements

1. Performance Optimization:
    - Use a lower resolution for faster processing or integrate parallel processing for real-time efficiency.
2. Additional Features:

- Implement obstacle classification to prioritize navigation strategies.
- Add logging for detected paths and obstacles for post-analysis.

3. Environment Adaptation:
    - Include automatic calibration for HSV ranges based on lighting conditions

Explanation of the Servo Control Script

This script provides a Python-based implementation for controlling servos using the Adafruit PCA9685 PWM/Servo Driver via an I2C interface. It defines a ServoController class and a helper function, create_servo, to easily manage servo motors for robotic or mechanical applications.

Key Components

1. Libraries and Initialization

- **adafruit_pca9685.PCA9685**: A library to control the PCA9685 servo driver module.
- **busio**: Used to initialize the I2C bus for communication.
- **board.SCL and board.SDA**: Represent the I2C clock and data lines, respectively.

2. ServoController Class

This class encapsulates all functionalities for controlling a servo motor:

Attributes:

- i2c: Initializes the I2C bus.
- pca: Creates a PCA9685 object to manage the servo channels.
- servo: Represents a specific channel on the PCA9685.

- min_pulse and max_pulse: Define the pulse width range for the servo, usually between 1000 and 2000 microseconds.
- reverse_factor: Allows the reversal of servo angle directions if required.
- center_fix: A correction factor for centering the servo, defaulting to 0.

Class Methods

set_angle(angle)

- Sets the servo to a specific angle between **-90° and 90°**.
- **Logic**:
    - Maps the angle to the corresponding pulse width using the formula:
      pulse width=min_pulse+(max_pulse−min_pulse)×(reverse_factor×angle+90180)\text{pulse width} = \text{min_pulse} + (\text{max_pulse} - \text{min_pulse}) \times \left(\frac{\text{reverse_factor} \times \text{angle} + 90}{180}\right)pulse width=min_pulse+(max_pulse−min_pulse)×(180reverse_factor×angle+90)
    - Converts the pulse width to a duty cycle suitable for the PCA9685.
- **Error Handling**:
    - Raises a ValueError if the angle is out of the allowable range.

sweep(start, end, step, delay)

- Moves the servo through a range of angles (default: -90° to 90°) in specified increments (step).
- **Arguments**:
    - start: Starting angle of the sweep.
    - end: Ending angle of the sweep.
    - step: Incremental angle for each step.

- delay: Time delay (in seconds) between each step.

center()

- Centers the servo by setting the angle to **0°**.

unlock()

- Unlocks the I2C bus, allowing other processes or devices to use it.

3. Helper Function: create_servo

- Simplifies the instantiation of the ServoController class.
- **Arguments**:
  - channel: Specifies the PCA9685 channel for the servo.
  - min_pulse and max_pulse: Define the pulse width range.
  - reverse_angle: Boolean to reverse the servo direction.
  - max_angle: Maximum allowable angle (default: 90°).

Key Features

1. **Precision Control**:
   - The script provides precise mapping of angles to pulse widths, ensuring accurate servo movements.
2. **Reversible Angles**:
   - The reverse_factor enables compatibility with servos installed in inverted orientations.
3. **Sweeping Capability:**
   - The sweep method automates servo movement over a range of angles, useful for scanning or calibration tasks.
4. **Modularity:**

- The create_servo function abstracts servo initialization, making the code reusable and adaptable for different applications.

Example Use Case

python

Copy code

```
if __name__ == "__main__":
    # Create a servo on channel 0 with default settings
    servo = create_servo(channel=0)

    # Center the servo
    servo.center()
    time.sleep(1)

    # Perform a sweep from -90° to 90°
    servo.sweep(start=-90, end=90, step=10, delay=0.5)

    # Set the servo to a specific angle
    servo.set_angle(-45)
    time.sleep(1)
    servo.set_angle(45)

    # Release the I2C bus
    servo.unlock()
```

Applications

- Robotics: For controlling robotic arms, wheels, or mechanisms requiring angular movement.

- Automation: In home automation systems for positioning mechanisms like blinds or valves.
- Education: For teaching servo motor control concepts in STEM projects.

This script ensures robust and precise servo control, making it highly suitable for real-world robotics and automation tasks.

Explanation of the BNO085 Sensor Script

This script provides a comprehensive implementation for managing a BNO085 9-DOF (Degrees of Freedom) IMU (Inertial Measurement Unit) sensor. The script is designed to interface with the sensor over I2C, read and process its data, and make it accessible for robotics or navigation applications.

1. Core Components

1.1 Libraries Used

- **busio and board**: For initializing and managing the I2C interface on the Raspberry Pi.
- **adafruit_bno08x**: Adafruit's library for controlling the BNO085 sensor.
- **math and numpy**: Used for mathematical operations, including trigonometric calculations.
- **time**: For timing operations and delays.

1.2 Classes and Functions

- **GyroError**: A custom exception to handle gyroscope-related errors.
- **BNO085Sensor**: A class encapsulating all operations for the BNO085 IMU sensor.

- **creat_bno_sensor**: A factory function to initialize and return an instance of the BNO085Sensor class.

2. BNO085Sensor Class

Attributes

- **address**: I2C address of the sensor (default is 0x4B).
- **i2c**: An instance of the I2C bus.
- **bno**: The initialized sensor object from the Adafruit library.
- **gyro_angles**: Tracks the integrated gyroscope angles (roll, pitch, yaw).
- **prev_gyro_angle_z**: Stores the last gyroscope Z-axis angle for integration continuity.
- **previous_time**: Tracks time for computing angle integration.

3. Class Methods

3.1 Initialization

- **__init__**:
    - Initializes the I2C connection.
    - Configures the BNO085 sensor to enable key features like accelerometer, gyroscope, and magnetometer readings.
    - Handles errors during initialization, including scanning available I2C devices for debugging.

3.2 Sensor Configuration

- **enable_features**:
    - Enables specific features of the BNO085:
        - Accelerometer
        - Gyroscope
        - Magnetometer
        - Rotation vector (quaternion output for orientation).

- **reset_sensor**:
    - Performs a soft or hard reset on the sensor, resetting its state.

3.3 Data Processing

- **quat_to_euler**:
    - Converts quaternion data into Euler angles (roll, pitch, yaw) for easier interpretation of orientation.
- **update_angles_from_gyro**:
    - Integrates gyroscope readings over time to compute angular displacement.
    - Uses a complementary filter to reduce noise and improve accuracy.

4. Sensor Data Retrieval

- **get_sensor_data**:
    - Reads data from the accelerometer, gyroscope, magnetometer, and quaternion sensors.
    - Converts quaternion to Euler angles for orientation tracking.
    - Returns a dictionary containing:
        - acceleration: Linear acceleration in m/s^2.
        - gyro: Angular velocity in rad/s.
        - magnetic: Magnetic field strength in μT.
        - euler: Orientation angles (roll, pitch, yaw) in degrees.
        - gyro_angles: Integrated gyroscope angles.

5. Real-Time Updates

Main Functionality

- The script continuously retrieves sensor data and prints the yaw angle (orientation around the Z-axis).

- Implements angle normalization to restrict yaw values between -180° and 180°.

6. Key Features

1. **Sensor Initialization:**
   - Automatically scans for connected I2C devices if the sensor fails to initialize.
   - Configures essential features for real-time orientation and motion tracking.
2. **Orientation Tracking:**
   - Computes accurate orientation data using gyroscope integration and quaternion-to-Euler conversion.
3. **Noise Reduction:**
   - Employs a complementary filter to combine gyroscope and accelerometer data for smoother angle calculations.
4. **Modular Design:**
   - The BNO085Sensor class encapsulates all sensor operations, making it reusable across multiple projects.

7. Example Usage

1. Basic Yaw Tracking

python

Copy code

```
if __name__ == "__main__":
    sensor = creat_bno_sensor()
    try:
        while True:
            data = sensor.get_sensor_data()
```

```
        if data:
            yaw = data['euler'][2]
            print(f"Yaw: {yaw:.2f}°")
        time.sleep(0.1)
    except KeyboardInterrupt:
        print("Exiting...")
        sensor.close()
```

2. Application in Robotics

- The yaw angle can be used for:
    - **Heading Control**: Maintaining or adjusting the robot's direction.
    - **Path Following**: Ensuring the robot stays aligned with a predefined path.

8. Error Handling

- Handles initialization errors, including missing sensors or incorrect I2C addresses.
- Prints the list of available I2C devices for debugging.

9. Applications

- Robotics: For navigation and orientation.
- Drones: For stabilization and flight control.
- IoT Devices: For motion and orientation tracking.

This script provides a robust foundation for integrating the BNO085 sensor into projects requiring precise motion and orientation tracking.

10. Performance Testing

As the team leader, overseeing the performance testing phase was a pivotal step in ensuring our robot met the rigorous standards of the competition. This

phase was carried out systematically, focusing on core metrics, iterative improvements, and final validation of the robot's capabilities.

Testing Procedures and Metrics

Speed Evaluation

The robot's speed was tested to ensure it could navigate efficiently without sacrificing stability. Multiple trials were conducted on varied terrains to verify consistent movement under different conditions.

Obstacle Detection Accuracy

The performance of the **Raspberry Pi Camera Module 3** and the **OpenCV-based vision system** was evaluated by placing obstacles of different colors and shapes in various lighting conditions. The robot's ability to differentiate between red and green blocks and make corresponding navigational decisions was meticulously analyzed.

Parking Precision

The parking mechanism was tested repeatedly to ensure the robot could align itself accurately within a designated space. Feedback from motor encoders, the gyroscope, and the servo motor steering system was used to refine precision and ensure consistent results.

Adjustments Based on Testing Feedback

Vision Processing Enhancements

One of the primary improvements was optimizing the vision system. The camera's resolution was adjusted to capture clearer images, and the **frames per second (FPS)** rate was increased to process real-time data more efficiently. These changes significantly boosted obstacle detection accuracy, particularly at higher speeds or under complex conditions.

Refinement of Obstacle Avoidance Logic

Through iterative debugging, the robot's decision-making algorithms were

refined to address edge cases. Scenarios such as overlapping obstacles or partially visible blocks were tested extensively, and adjustments to the code ensured the robot could navigate these challenges reliably.

Performance Results and Key Achievements

High Accuracy and Reliability

The robot successfully navigated all challenge scenarios, showcasing a high degree of accuracy and responsiveness. Its ability to detect obstacles, make split-second decisions, and execute precise movements was a standout achievement.

Seamless Coordination

The integration of hardware and software systems proved flawless, with motor controls, vision processing, and sensor feedback working in harmony to achieve optimal performance.

Competitive Readiness

The thorough testing and subsequent refinements positioned the robot as a strong competitor, demonstrating reliability, adaptability, and precision in all competition-required tasks.

11. Acknowledgments

أكاديمية طويق
Tuwaiq Academy

I am a robotics trainer with three years of experience, specializing in various robots. I have achieved first-place rankings in several competitions, including being named "Best Engineer." I was the champion of the **"Future Science Challenge"** organized by the Hamdan Bin Rashid Al Maktoum Foundation, and I won a gold medal at ITEX and the "Best Engineering Design" award at VEX. I am passionate about robotics and aim to inspire the next generation of engineers.

GitHub Link

Scan to access

Github

YouTube Link

Obstacle Challenge

Open Challenge