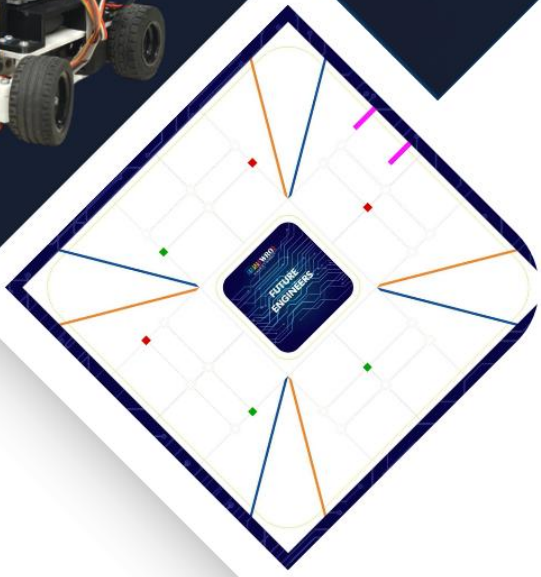
YB ROBOT CLUB

YB-SUNFLOWER

```
void loop() {
  float baseDesiredDistance = 25;
  while (analogRead(BUTTON) > 500) {
    getIMU();
    motor(50);
    Color_detection();
    float desiredDistance = baseDesiredDistance;  // Start with the base
    if (TURN == 'L') {
      desiredDistance = 16;
    } else if (TURN == 'R') {
      desiredDistance = 17.5;
    }
    float distanceError = getDistance() - desiredDistance;
    float deadband = 2.0;
    if (abs(distanceError) < deadband) {
      distanceError = 0.0;
    }
    float directionFactor = (TURN == 'R') ? -1.0 : 1.0;
    float adjustedYaw = pvYaw + (distanceError * directionFactor);
    float pidOutput = compassPID.Run(adjustedYaw);
    steering_servo(pidOutput);
    ultra_servo(pvYaw, TURN);
    // Serial.println(getDistance());

    if (count >= 12) {
      long timer01 = millis();
      while (millis() - timer01 < 800) {
        motor(100);
        getIMU();
        Color_detection();
        float desiredDistance = baseDesiredDistance;  // Start with the base
        if (TURN == 'L') {
```



WRO Future Engineer 2025 Documentation

WRO®
WORLD ROBOT OLYMPIAD ™

FUTURE
ENGINEERS

A SELF-DRIVING
CAR CHALLENGE

AGE GROUPS:
14-22

WRO® 2025
SELF-DRIVING CARS

WRO international premium partner

aramco

WRO international gold partners

fischertechnik

ROBOROBO
LEARNING FUTURE

Abstract

Objective:

This document outlines our strategy for the WRO Future Engineer 2024 competition, focusing on the design and program development of a vehicle capable of navigating obstacles and handling diverse challenges. Key improvements include advanced motion control, energy management, and adaptable programming, enabling responsive and efficient performance.

Methods:

To achieve precise maneuverability, we integrated advanced motion control with customized programming. Energy management strategies optimize power consumption, while a systematic inspection ensures vehicle readiness. Our decision-making approach combines sensor input with real-time camera analysis for greater adaptability.

Key Engineering Factors:

This project highlights essential strategies for obstacle navigation, with vehicle design strategies and adaptive coding techniques that provide real-time responses to competition challenges. Developed algorithms and source code support flexible, reliable performance in varied conditions.

Conclusion:

This document reflects our comprehensive engineering and programming approaches, demonstrating a commitment to excellence in the WRO Future Engineer competition through innovative design and real-time adaptability.

Content

1.About Us

This is our team

Our team name, YB-SUNFLOWER, carries a meaningful story. The word “Sunflower” comes from one of our seniors, Tawan (ตะวัน). In Thai, “sunflower” is ทานตะวัน, which is directly connected to his name. Since 2023, he has inspired us to join the Future Engineers Competition, always supporting us and giving guidance from outside the team. To honor the role he played in motivating us, we included “Sunflower” in our team name. This year is even more special, because for the first time, he is actually competing together with us as part of the team.

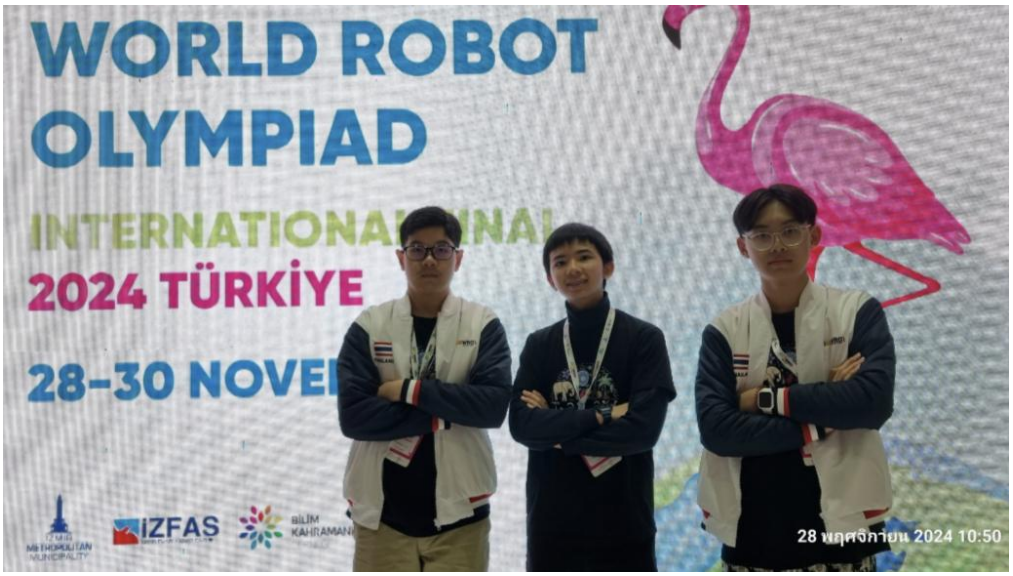
Team History

We are **YB-SUNFLOWER**, team from Bangkok, Thailand. We are a part of YB-Robot club from Yothinburana School with our mentor, Punnapon Tanasnitikul. This is our club website https://ybrobot.club/. The club was established in 2009 with the first name being “YB Dream Team”. Last year, we also attend in WRO 2024 that took place in Türkiye.

This is the first WRO international award in 2010.

This is the second WRO international award in 2011.

This is the latest WRO international award in 2024 (13th place).



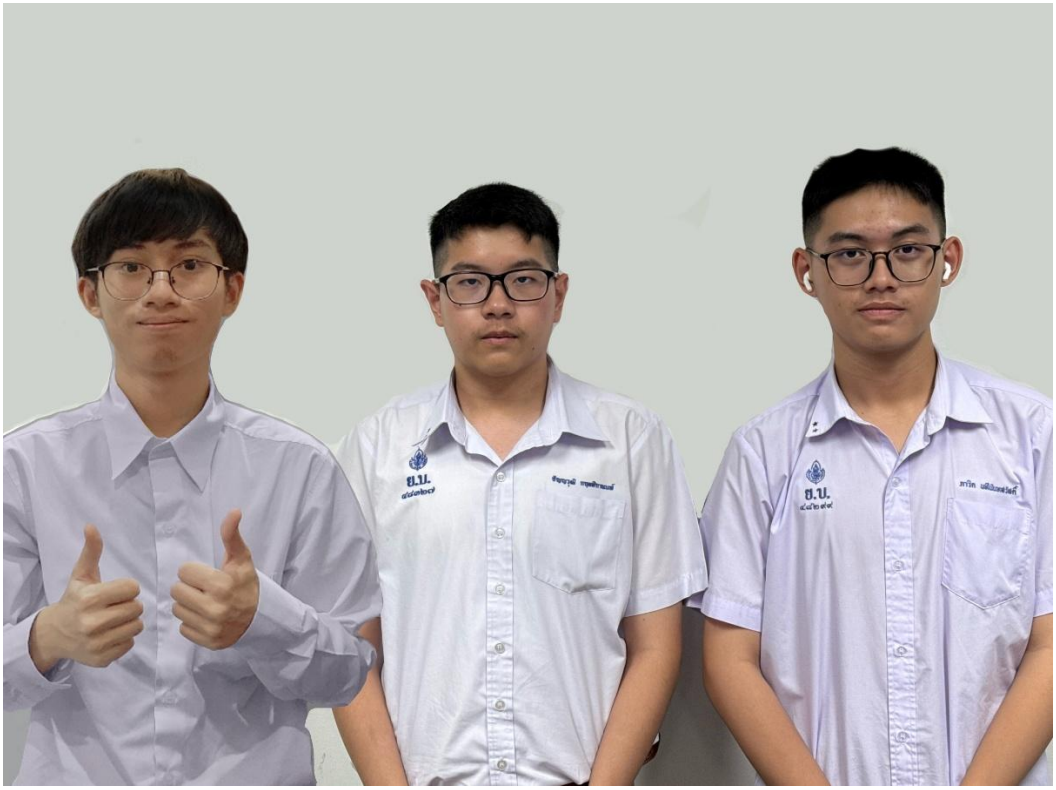
Currently in 2025, we are a part of YB Robot Team. Our senior brought this team name to global recognition, so there is a responsibility since our senior perform excellently. We are hoping to keep this name on the winning streak and make them proud.

Our Team

The team consists of 2 intimate friends and 1 of our senior, 'Tawan'. Pawit Nateenantasawasd, Thanyawut Krittikanon, and Natapol Chusang. We have known each other for four years since 2021, forming a strong bond that enhances our collaborative efforts. We first met in high school, being in robot club, competing in many robot competitions together, and sharing our robotic knowledge. Much of what we know today has been learned from the experiences of our seniors in out team, who have taught us valuable lessons and helped shape who we are now.

Last 2 years, with great enthusiasm and high spirits, we joined Future Engineers twice. Reaching the international round was already a great honor, but we were soon to realize that in order to win, one needs more than just enthusiasm. Our mistakes led us take the 8th and 13th position, a respectable position indeed.

This year, we are back with greater confidence, and wiser from past experiences. We honed our strategies and prepared extensively. We fixed every mistake we have done last 2 years. Being back at this competition make us want to ensure the best outcome, and we are here to make everyone proud.



There are three members in our team, all extremely focused and dedicated to their positions, performing functions as follows:

1. Thanyawut Krittikanon – Robot Designer

Thanyawut is known in our team as **The God of Fusion**, with outstanding skills in 3D design and modeling. His expertise in creating precise and creative robot designs makes him a crucial part of our team. With his talent, he can turn our ideas into detailed models that guide the building process and bring our concepts to life.

2. Pawit Nateenantasawasd – Document Designer and Electrician

I am Pawit, the document designer of our team. My role is to make sure that everything we present — from reports to presentations — is clear, well-structured, and professional. I enjoy transforming complex ideas into something simple and easy to understand, while also giving it a creative and polished design. Besides designing documents, I also have strong coding skills in multiple languages such as C++, C, Python, Micropython, and HTML, which allow me to contribute to the technical side of our work.

3. Pongpapat Putongkam – The Mechanic and Constructor

Natapol is our master programmer, and sometimes we joke that he's not even human because of how effortlessly he can code anything. From bad-looking robot to advanced systems, he can take on any programming challenge that comes his way. His ability to quickly understand problems and turn them into working solutions makes him a key part of our team's success. With his skills, even the most complex systems feel possible to build — it's like there's no code in the world he cannot write.

Our Vision and Goal

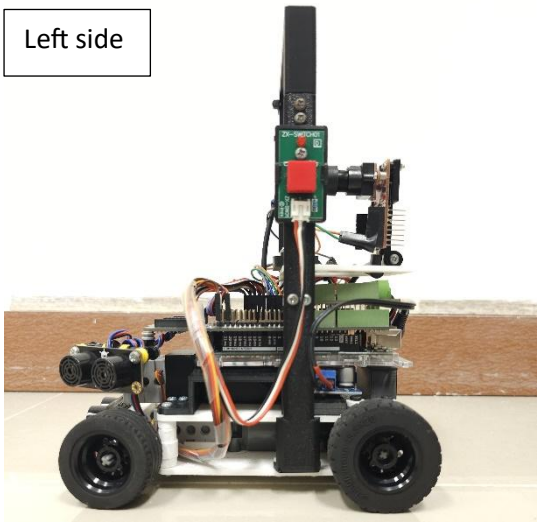
We want to create a robot that can operate similarly to a self-driving car, with the hope of integrating it with an actual automobile. This competition is the first step toward our future goal of helping our country and inventing new things. Additionally, via participating in this competition, we have developed and learned numerous skills like collaboration, imagination, and foresight. We also desired to meet new friends from other nations in order to exchange cultures. Furthermore, we intend to utilize the experience we have obtained from this tournament to better ourselves and mentor our junior at school.

" *We're a team; we work as a whole, think as though we have a hundred heads, and know each other as if we've been together for years. We've come together, gathered the hands of each member, and moved forward with one, united mind*. "

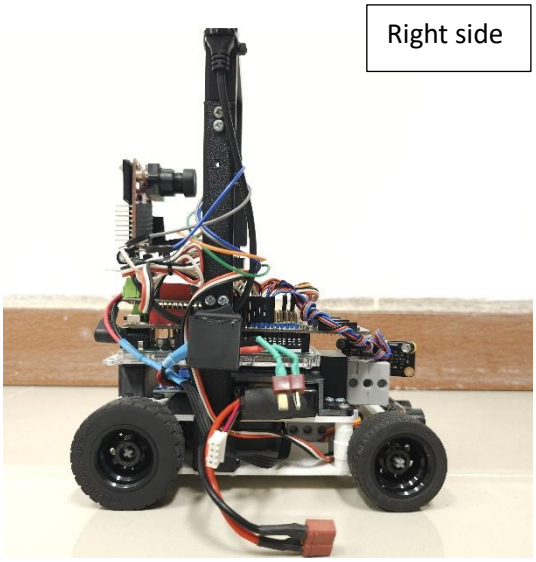
-YB_SUNFLOWER-

2. Robot Photos

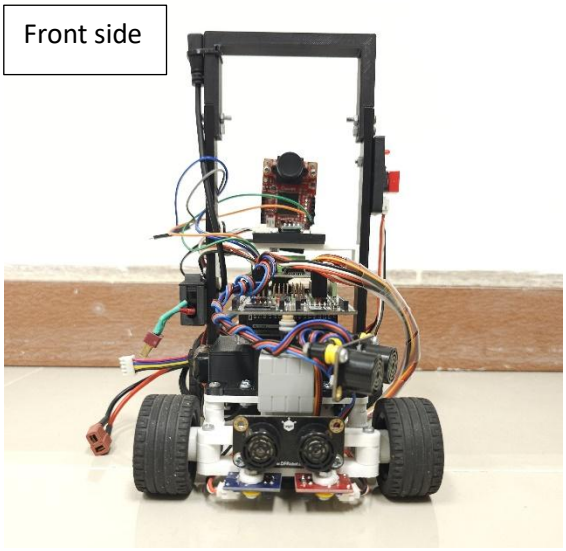
Left side



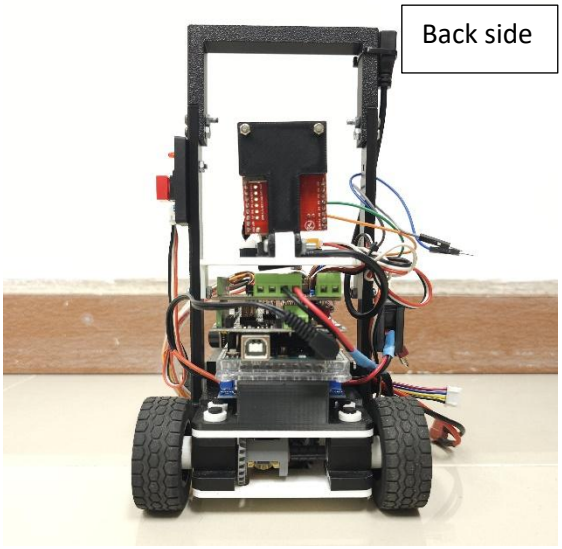
Right side



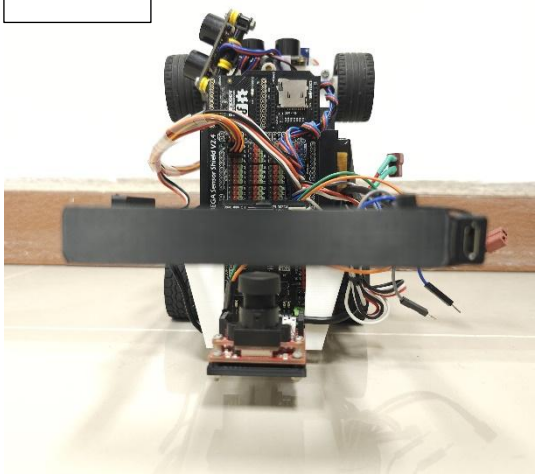
Front side



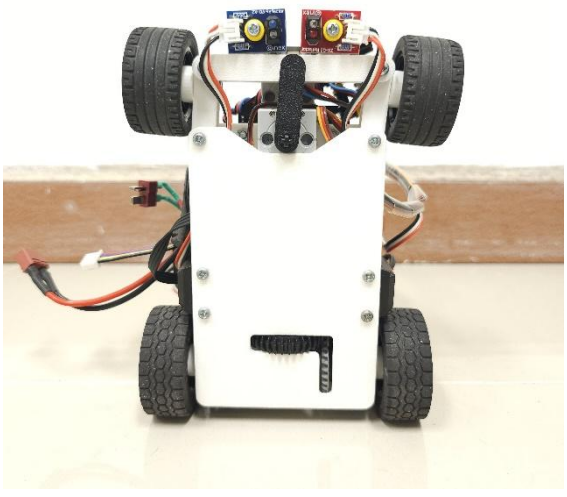
Back side



Top side

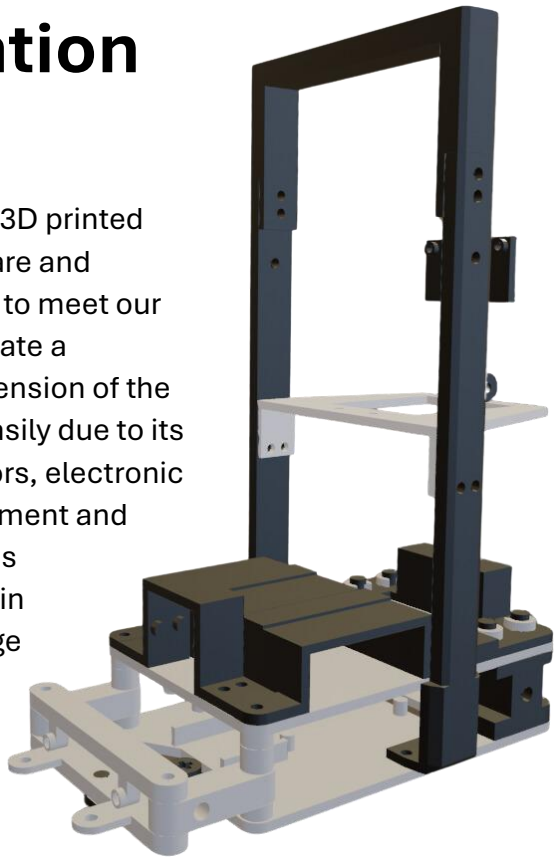


Bottom side



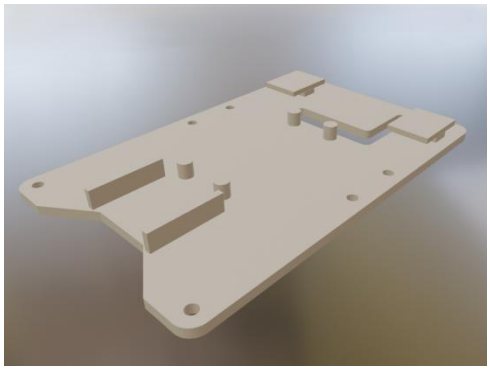
3. Engineering information

3.1 Robot's Chassis



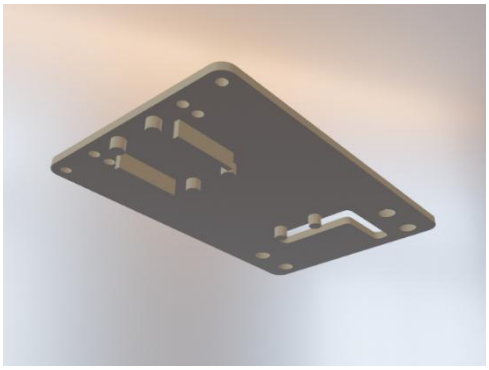
The robot chassis design consists mainly of the 3D printed parts, which provide a strong support for all the hardware and electronics devices. Each element is custom designed to meet our specific requirements and interlocks seamlessly to create a lightweight yet resilient structure. Modification and extension of the orthotics and bambulab 3D-printed chassis is made easily due to its modular construction. It is designed to house the motors, electronic circuitry, and battery while enhancing thermal management and balanced weight distribution for efficient operation. This strategy not only allows for fast modeling and changes in design, but it also makes it easy if an operational change requires that some parts should be removed or replaced with better ones as the robot matures.

Main Base



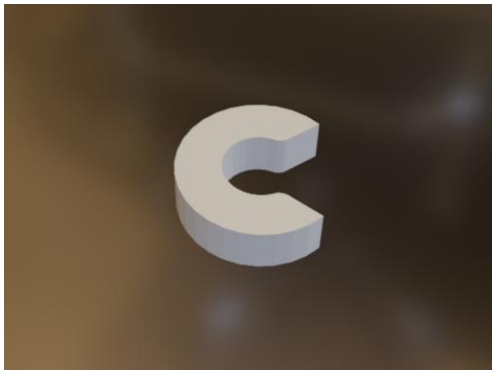
The main base provides the foundation for creating all other parts. It has **cylindrical extensions meant to attach the servo and motor** too — designed with LEGO connectors in mind, as opposed to clunky black pegs. Also detailed are **the openings to accommodate both differential gear and regular gears so they do not hit the base when trying to rotate.** One is a specially designed slot to work with this 3D-printed differential gear mount. **The front part is cut out deeper so the servo for steering fits inside and has no problems to spin freely.** This design allows to move effectively and optics in the small space of the robot.

Support Base



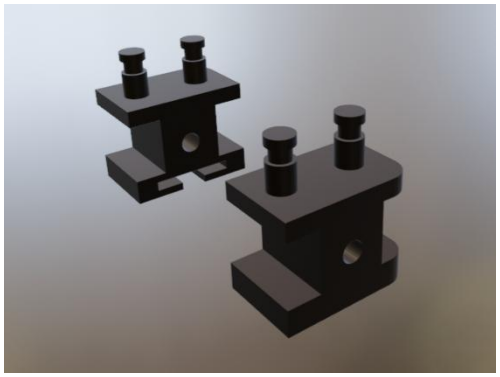
The purpose of this component is to reinforce the main base securing attachments more firmly. It serves as an additional foundation to mount objects and 3D models, ensuring stability and strength in the overall structure.

C Clip



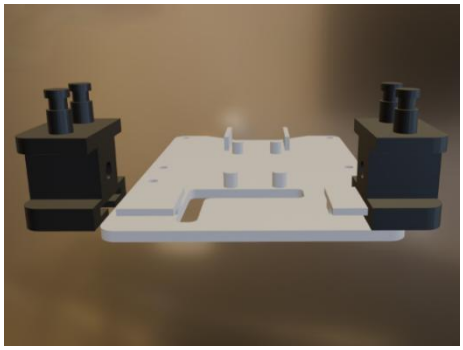
This component functions as a stabilizing element, **similar to a nut**, preventing objects from shifting or moving out of place. The open, C-shaped design allows it to be easily positioned around shafts or other components, ensuring they remain securely fixed while maintaining flexibility for adjustments when needed.

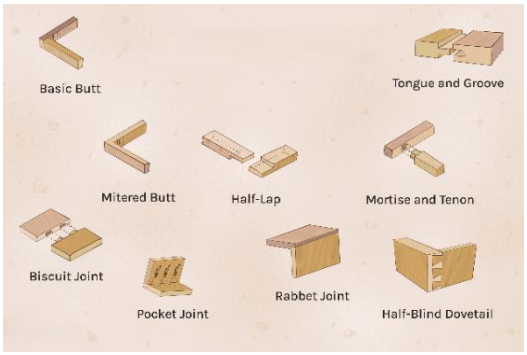
Differential Gear Mount



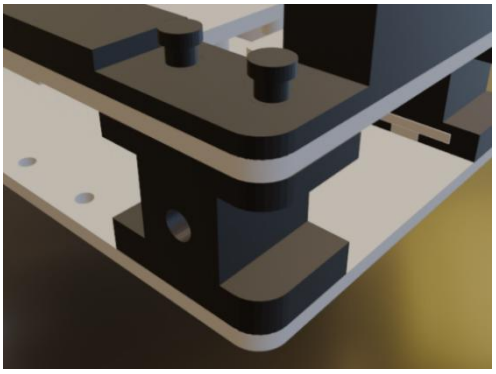
This component is very important for supporting the differential gear firmly and creating solidity to the drivetrain of our robot. Among the basic choices that we made during the work on the robot design, one of the key decisions was to **use as few bolts and nuts as possible**. This reduction in weight is achieved by **using ABS plastic filament instead of steel, which is significantly lighter.** This choice not only decreases the overall weight but also simplifies the assembly by requiring fewer additional elements and connections.

A lighter robot just provides enhanced mobility and more efficiency, which is a crucial factor given the requirements of competitively in the competition. **In order to attain a most stringent connection to the main base an interlocking joint was implemented.** This design helps in assembling by minimizing the use of conventional fasteners during the construction process.

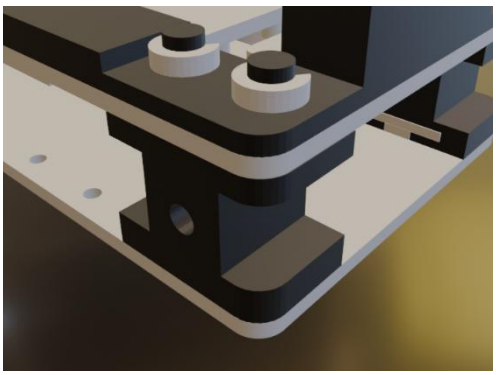




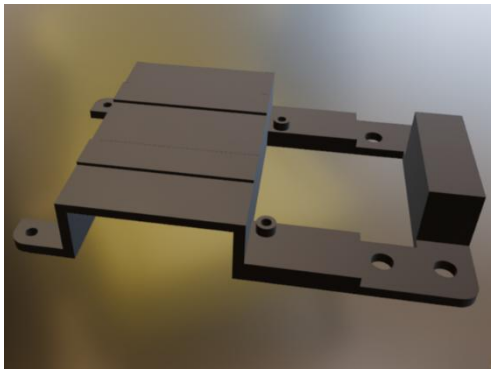
From this connection, we developed **cylindrical formings which work more like bolts**, used to place the parts in right position as well as fix them in place.



Made for added security, precision grooves feature with special C-clips that serve as a replacement for nuts, designed to fit seamlessly. These **C-clips are held firmly in place within specially designed grooves that prevent any movement.** The groove area has a smaller diameter than the other sections, which ensures a tight fit, preventing any up-and-down motion. **The only way to release the C-clip would be to pull it sideways**, making the design stable and secure.

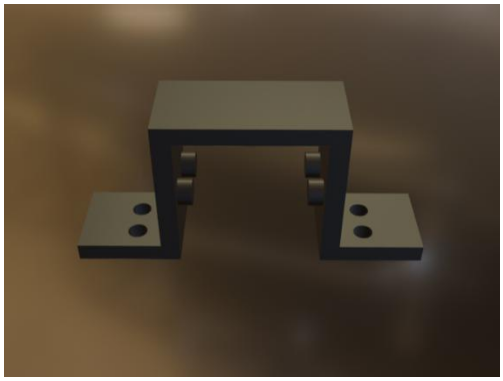


Power Tray



This is the part that holds the 11.1V battery, which is easily removable for charging. This also serves to securely mount the step-down module in place. The raised portion at the back forms a stable platform for mounting the microcontroller board. Hence, this design optimizes the layout by keeping the important electronics of the best organization in space and providing stability.

Servo Bracket



This component is intended for firm attachment of a servo motor accompanied by an ultrasonic sensor that is to be used in rotation. The design incorporates cylindrical spigots so as to eliminate the use of black pins which are used to fasten the servo motor fabricated for Lego use. With this arrangement, the sensor movement is stable, well positioned and rigid within the system that improves the overall utility of the sensor.

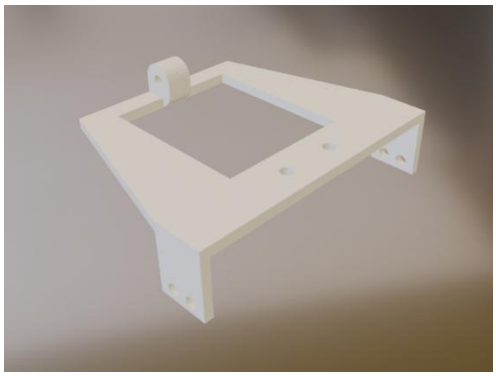
Camera Plate

This component is designed to attach to the back of the camera, providing a mounting point that enables it to be securely connected to other components.

Handle

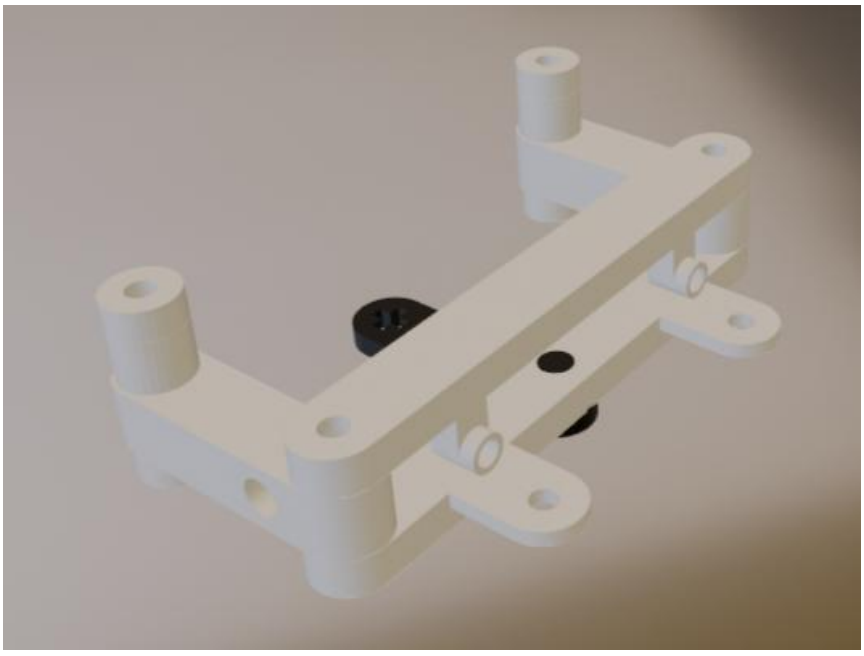
This handle is an important part as **it allows the robot to be safely lifted, especially at times of high speeds.** Should it happen that the robot malfunctions or does not act as intended, reaching down quickly to grasp it is practically impossible and could cause unintentional damage by tugging at sensitive pieces, including wires. With the handle, the robot can be raised safely and securely with minimal risk of mishandling.

Gyro-Camara Mount

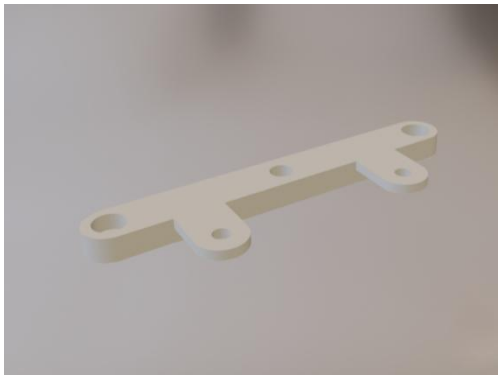


This component is mounted to the handle and serves as the mounting area for the camera, which is secured through the camera plate. **It allows the camera to tilt forward or backward, enabling flexible adjustments to capture the best angle for the robot's operations.** Additionally, it offers a stable attachment point for the gyro sensor, ensuring accurate orientation and sensor alignment. The gyro mount is positioned around the middle top of the robot to achieve the most accurate output, while the camera mount is placed at the back to ensure precise object detection.

Steering Module



- **Bottom Steering Mount**



Its primary function is to ensure that both steering arms are aligned at the same angle during turns, providing precise and synchronized steering. It connects to the steering linkage arm, which is linked to the servo motor, allowing controlled movement and adjustments to the wheel angles. Additionally, we have designed a mounting point at the front of this component for a light sensor.

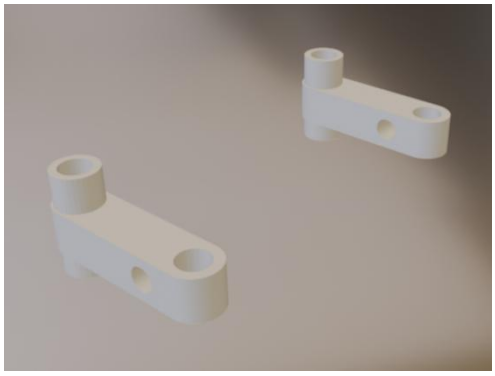
- **Steering Linkage Arm**

Its main function is to connect to the servo motor and control the steering direction of the robot.

- **Top Steering Mount**

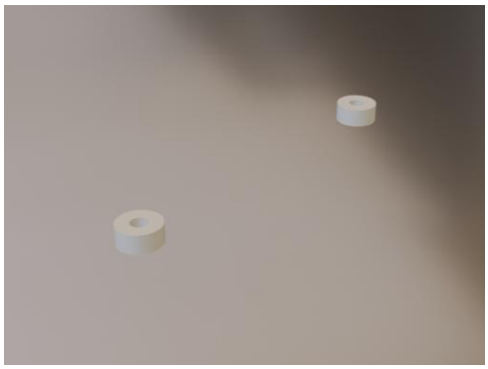
Its primary function is to reinforce the bottom steering module, providing additional strength and stability to the steering structure. Additionally, we've designed a mount for an ultrasonic sensor

- **Steering Arms**



Its main function is to keep the bottom and top steering mounts aligned with the front of the robot, ensuring stability and correct orientation. Additionally, it serves as the attachment point for the wheels. At the rear of this component, **we've designed a large hole to fit a metal standoff or PCB support post, providing extra reinforcement to prevent breakage and enhance durability.**

- **Steering Arm Supports**



Its primary function is to **provide additional support for the PCB mounting posts, which are made of metal and are insufficiently long.**

3.2 Components

3D Printer: Bambu Lab X1 Carbon by Bambu Lab

The reasons that we use Bambu Lab X1-Carbon 3D printer because this printer is a high-performance 3D printer designed for precision and efficiency. It is equipped with advanced technology and features, making it suitable for professional use as well as hobbyist projects that require high-quality prints.

Technical information

-Build Volume: 256 × 256 × 256 mm^3*

-Nozzle: 0.4 mm Hardened Steel Included

-Hotend: All-Metal

-Max Hot End Temperature: 300 ℃

-Filament Diameter: 1.75 mm

-Supported Filament: PLA, PETG, TPU, ABS, ASA, PVA, PET

-Ideal for: PA, PC, Carbon/Glass Fiber Reinforced Polymer

-Build Plate Surface: Bambu Textured PEI Plate or Bambu Cool Plate (Pre-installed, Random, Both compatible with Micro Lidar)

-Max Build Plate Temperature: 110℃@220V, 120℃@110V

-Max Speed: 500 mm/s

-Max Acceleration: 20 m/s^2

-X1C Dimensions: 389 × 389 × 457 mm^3

-Package Size (X1C): 480 × 480 × 535 mm^3

-Net Weight (X1C): 14.13 kg

-Gross Weight (X1C): 18 kg

-Package Size (X1C Combo): 480 × 480 × 590 mm^3

-Gross Weight (X1C Combo, AMS included): 22.3 kg

-Voltage: 100-240 VAC

-Frequency: 50/60 Hz

-Power Consumption: 1000W@220V, 350W@110V

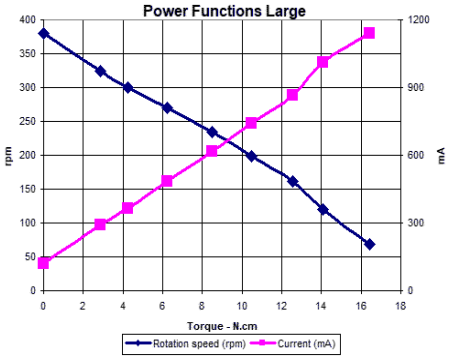
Filament: Acrylonitrile Butadiene Styrene or simply known as *ABS

ABS (acrylonitrile butadiene styrene) is probably the most common filament used in 3D printing. It is especially valuable in strong plastic parts that must remain resilient in the face of temperature swings. It is mainly used in FDM (fused deposition modeling) 3D printers. ABS is a thermoplastic polymer composed of three monomers: acrylonitrile, butadiene, and styrene. The material was first patented in the 1940s and very quickly gained popularity.

Movement Parts

1.Driving motor: Power Function Large motor by Lego

It's a simple motor, and we chose this motor because of its easy connection to our robot and its cost-effectiveness. This motor is small yet powerful and small power consumption. It is the perfect size for our robot comparing to other motor in the market which typically round on all side and need specific mounting bracket for example ZGA32RI. The motor comes with a dedicated port for Lego. So, we modified it to make it able to connect to the board.



Technical Information

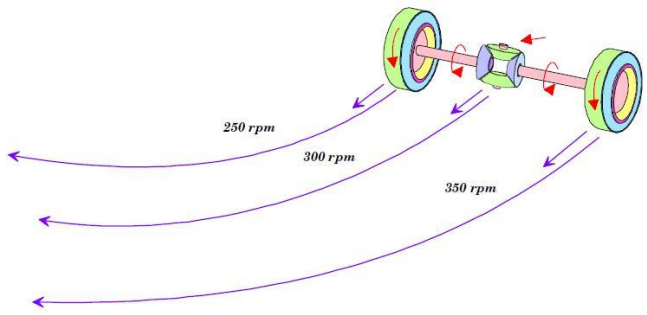
-Speed: 390rpm

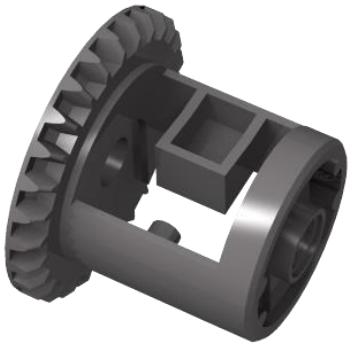
-Maximum torque: 40N/cm

-Voltage: 9v

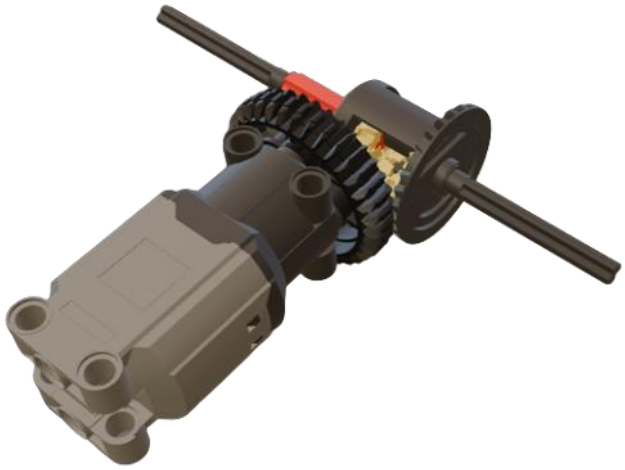
2. Differential: Technic, Gear Differential with Inner Tabs and Closed Center, 28 Bevel Teeth

This part ensure that both wheels have the power from the motor, which makes the robot drive forward. This part has gear teeth on the side, which are connected to the gear we put on the motor earlier, and then we put 3 small gears inside the differential. The differential is attached to the main body using an axle from the wheel.





Driving Motor with Differential Gear



After presenting the information and the reasons for selecting the motor and the differential gear, the next crucial step is to understand the outcomes resulting from the connection of these two components. This analysis aims to determine the speed and torque that the system can deliver, which are key factors in optimizing and enhancing the system's performance. Below are the calculation methods and the results derived from our analysis

Step 1: Calculate the gear ratio

The gear ratio is determined by the ratio of the number of teeth on the input and output gears:

Gear Ratio = $^{\text{Number of teeth on input gear}}$ / $_{\text{Number of teeth on output gear}}$ = ^{36} / $_{28}$ = 1.2857

Step 2: Calculate output RPM

The output RPM increases due to the gear ratio. To find the output RPM:

Output RPM = Input RPM × Gear Ratio = 390 × 1.2857 = 501.4 RPM

Step 3: Calculate output torque

The torque decreases inversely proportional to the gear ratio. To find the output torque:

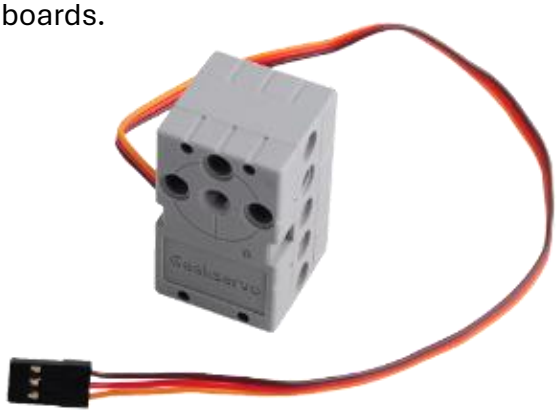
Output Torque = $^{\text{Input Torque}}$ / $_{\text{Gear Ratio}}$ = ^{40} / $_{1.2857}$ = 31.1 N·cm

Final Results:

- **Output RPM**: ~501.4 RPM
- **Output Torque**: ~31.1 N·cm

3.Servo: GEEKSERVO 2kg 360 Degrees

We use this servo for steering the robot and employ an ultrasonic sensor for rotation. This servo is compatible with LEGO, making it easy and convenient to build the robot by just putting studs in the hole on the side. We like how you can connect two axles to the dual outputs on this servo so you can power two wheels or gears or mount the servo securely inside articulated limbs and other contraptions. Additionally, the gears inside these servos will 'slip' when the blocking load is too high instead of jamming, helping avoid damage to your servos and boards.



Technical information

-Rotation: 360 Degree
-Start Voltage: 2.5V
-Working Voltage: 3.3-6.0V
-Rated Voltage: 4.8V
-Rated Current: 70mA
-Maximum torque: 1.6 0.2 kg-cm (4.8V)
-Angle rotation speed: 60 / 0.14s

4.Wheel

There are a lot of wheels to select. We chose this one because of it's size. If the wheels are too small, it reduces the speed due to the lack of rotation. But if the wheels are too big, it makes the robot slower and harder to control. With the combination of the motor and the wheels, the robot can maintain the speed we can control.

4.1 Front Wheel: Lego Tire 43.2 x 22 ZR and Wheel 30.4mm D. x 20mm with No Pin Holes and Reinforced Rim

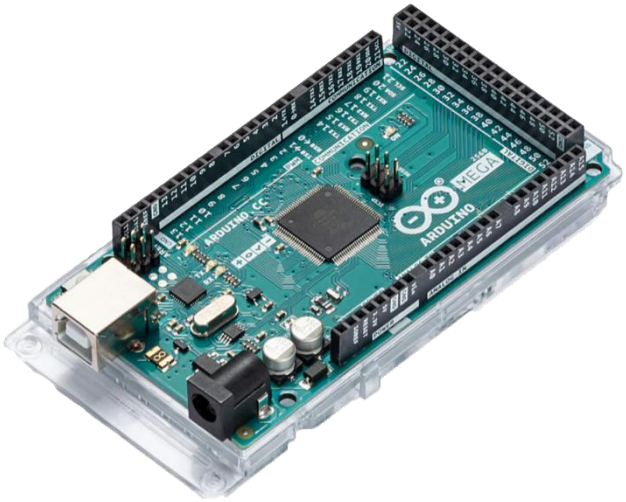
4.2 Rear Wheel: Lego Tire 49.6 x 28 VR and Wheel 36.8mm D. x 26mm VR with Axle Hole

Controller

1.Microcontroller Board: Arduino Mega 2560 R3

This part is like a brain of our body. Its job is to store all the programs of our robot from the computer, every component in the robot comes through here. We chose this board because of its connection port; it contains tons of ports that we want such as 3 UART port. We used Arduino Uno last year, but the problem is there's not enough port for OpenMV and GY-25. But there's some disadvantage in this board. Because this board has a lot of connection port, it comes with weight and size. It's almost 2 times longer than the UNO. And that makes the robot long and heavy

Technical information



-Microcontroller: ATmega2560

-Operating Voltage: 5V

-Input Voltage (recommended): 7-12V

-Input Voltage (limit): 6-20V

-Digital I/O Pins: 54 (of which 15 provide PWM output)

-Analog Input Pins: 16

-DC Current per I/O Pin: 20 mA

-DC Current for 3.3V: Pin 50 mA

-Flash Memory: 256 KB of which 8 KB used by bootloader

-SRAM: 8 KB

-EEPROM: 4 KB

-Clock Speed: 16 MHz

-LED_BUILTIN: 13

-Length: 101.52 mm

-Width: 53.3 mm

-Weight: 37 g

2.Sensor Shield: Gravity IO Sensor Shield for Arduino Mega Due

This part is an extension of the board. It is where ultrasonic, light sensors, button sensors, camera, compass, and servos go. It has a lot of connection pins which can be used for each component. But with that it also come with a very long design. Make it hard to design where to place it on robot.

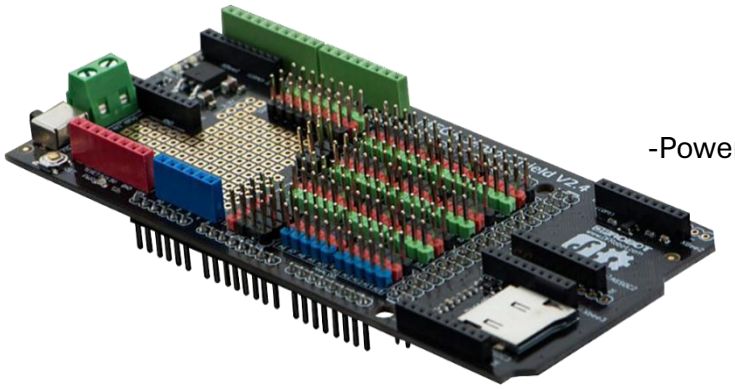
Technical information

-Compatibility: Most Arduino shields

-Compatible Boards: Arduino Mega boards, DFRobot megaADK, Arduino megaADK

-Extended TTL Connection Pins: Four Serial ports

-Prototyping Area: DIP prototyping area for additional modules or components



-Xbee Slots: 3

-microSD Slot: 1

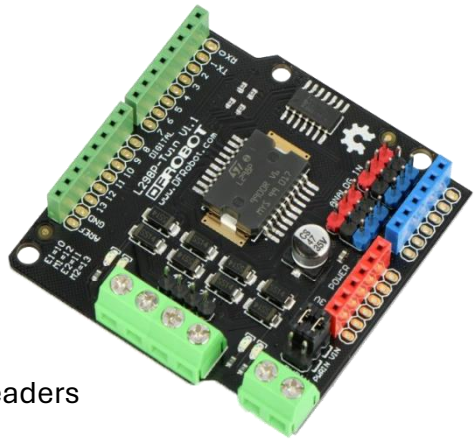
-Power Switch: Between Arduino Mega or external power

-Size: 125 x 57 mm (4.92 x 2.24")

3.Motor Shield: Gravity 2x2A Motor Shield for Arduino Twin

This part is also an extension of the board. It makes the connection between the board and motor easier. We connect the pin with the top of sensor shield.

Technical information



-Motor Driven Voltage: 4.8V to 35V

-Output Current: Up to 2A/channel

-Total Power Dissipation: 25W (T=75℃)

-Driven Structure: Dual full-bridge driver

-Driven Power Port: External power terminal, or VIN from Arduino

-Driven Output Port: 2 channel screw terminals, or male PIN headers

-Control Port: 4 TTL compatible digital signals (Digital 10-13)

-Operation Temperature: -25℃ to 130℃

Shield Size: 56x57mm (2.20x2.24")

Power Management and Inspection

1. Camera: OpenMV H7 R1

This component is very important for avoiding obstacle. It can detect red and green obstacle from distance to avoid crashing into it. The OpenMV also comes with its own microcontroller. Making the robot locate and think faster when see the obstacle. The OpenMV also comes with GLCD screen at the back of it to display what the camera sees. This camera can be coded with MicroPython. Additionally, this camera wires are connected with the sensor shield.

Installing dependencies

To set this device up in Edge Impulse, you will need to install the following software:

1. https://docs.edgeimpulse.com/docs/tools/edge-impulse-cli/cli-installation (Edge Impulse CLI.)

2. https://openmv.io/pages/download (OpenMV IDE.)

Technical information

-Processor: STM32H743VI ARM Cortex M7

Clock Speed: 480 MHz

SRAM: 1 MB

Flash Memory: 2 MB

I/O Voltage: 3.3V (5V tolerant)

Number of I/O Pins: 10

-ADC: 12-bit

-DAC: 12-bit

-Default Sensor: OV7725

-Resolution: 640x480

-Modes: 8-bit Grayscale at 75 FPS, 16-bit RGB565 (75 FPS above 320x240, 150 FPS below 320x240)

-Lens: 2.8mm, M12 mount

-Additional Modules: Global Shutter Camera, FLIR Lepton Adapter

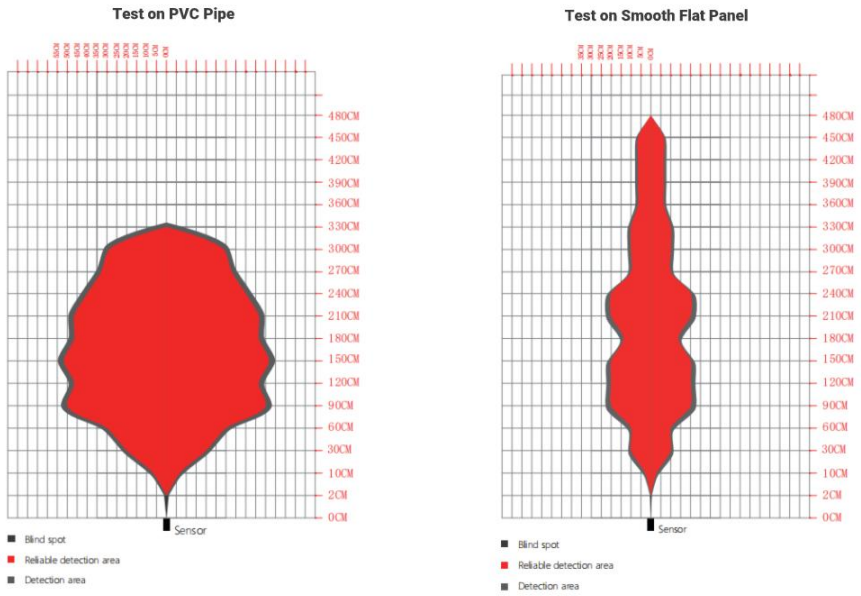
-Battery Connector: Compatible with 3.7V LiPo batteries

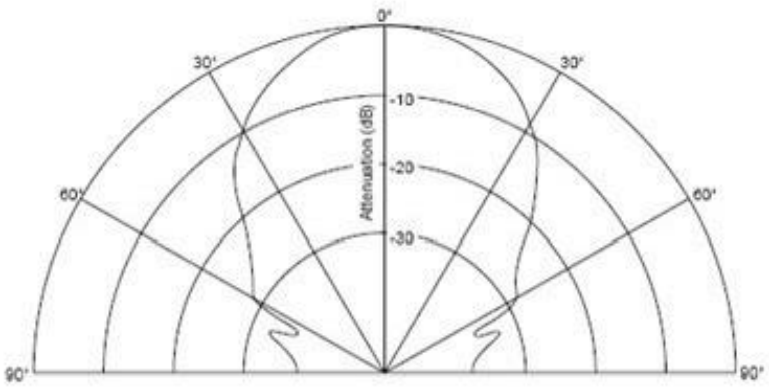
2.Ultrasonic sensor: Gravity URM 09

An Ultrasonic sensor is a device that can measure the distance to an object by using sound waves. It measures distance by sending out a sound wave at a specific frequency and listening for that sound wave to bounce back.

Since it is known that sound travels through air at about 344 m/s (1129 ft/s), you can take the time for the sound wave to return and multiply it by 344 meters (or 1129 feet) to find the total round-trip distance of the sound wave. Round-trip means that the sound wave traveled 2 times the distance to the object before it was detected by the sensor; it includes the 'trip' from the sonar sensor to the object AND the 'trip' from the object to the Ultrasonic sensor (after the sound wave bounced off the object). To find the distance to the object, simply divide the round-trip distance in half.

DFRobot URM09 is an ultrasonic sensor specially designed for fast ranging and obstacle avoidance applications. Its measuring frequency can reach up to 30Hz. The sensor adopts built-in temperature compensation and analog output. Meanwhile, it can provide accurate distance measurement within 500 cm. The sensor is compatible with Arduino, Raspberry Pi, or other main controllers.





We use the Ultrasonic Sensor (SEN0307) to measure the distance between the robot and the walls. This sensor utilizes an analog voltage output and provides accurate distance measurements within the range of 2-500 cm, with a precision of 1 cm and an accuracy of ±1%. It is highly suitable for this competition and is compatible with boards that have 3.3V or 5V logic levels.

Technical information

-Supply Voltage: 3.3~5.5V DC
-Operating Current: 20mA
-Operating Temperature Range: -10℃～＋70℃
-Measurement Range: 2cm～500cm (can be set)
-Resolution: 1cm
-Frequency: 50Hz Max

3.Light Sensors

The color sensors play an important role in both rounds, as we use them for line detection. There are 2 lines with different colors in the corner of the race field, which is why we use 2 different colors of the color sensor, blue and red. The blue color sensor is used for detecting both colors, while the red color sensor is only used for the blue line.

3.1 ZX-03B By INEX

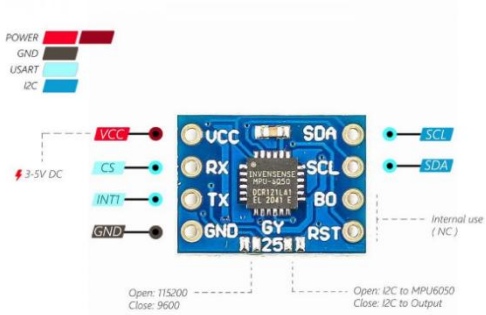
3.2 ZX-03R By INEX

4.Gyro: Gy-25

A gyro is a component that enables a robot to determine its orientation and turn in the appropriate direction. We chose this gyro sensor specifically because of how effective it is. It also comes in a very small size to attach to our robot.

Technical information

-Chip: MPU-6050
-Power voltage: 3 - 5V
-Communication mode: Serial communication (baud 9600,115200), IIC communication
-Model Size: 15.5mm*11.5mm
-Pitch: 2.54mm
-Direct Data: YAW ROLL PITCH
-Heading angle (YAW): ± 180 °
-Roll angle (ROLL): ± 180 °
-Pitch angle (PITCH): ± 180 °



GY-25 Pin-out

5.Touch Sensor: ZX-Switch 01

This switch gives us an easier way to start the robot. Since the controller board came with switches like OK, Switch A, Switch B, and ON/OFF, it is hard to push, so we found this switch that could be attached to the frame outside the board.

6.On/Off Switch: SPST ON/OFF Switch 2 Pin Rocker Switch DC 125/250V

This switch is for cutting the power from the battery to the robot. The regulation states that before starting the robot, the power must be cut off. That's when this switch came in. To use this switch, we solder red wire (Positive pole) to the switch on 1 side for input. Then another solder red wire for output on the opposite side. You can put the black wire (Negative pole) straight into the step down. When the switch is turned on, the power from the battery will direct into the stepdown and then the robot. Additionally, we have designed a protective case for this switch to ensure safety and durability.

7.Step-down: HW-316 V6.0

This is a step-down DC-DC module. It comes with a status indicator light, a display screen that shows the voltage meter, and self-calibration of the voltage meter. The electrical voltage has an error of 0.05 V, with a measuring range of 0–40 V. We need this step down to show us how long until we need to recharge the battery.

Technical information

Input voltage: DC 4.0 ~ 38V
Output voltage: DC 1.25V ~ 36V continuously adjustable
Output current: max 5A,
Output power: up to 75W
Voltmeter error: ± 0.05V
Measure range: 0 ~ 40V
Conversion efficiency: up to 96%
Load regulation: S (I) ≤ 0.8%
Voltage Regulation: S (u) ≤ 0.8%

8.Battery: 11.1 voltage 3 cell Lipo-Battery

We upgraded from a 7.4V battery to an 11.1V battery due to reliability concerns and the limited capacity of the 7.4V option. The 11.1V battery enhances the robot's performance by providing higher voltage, which reduces current draw and minimizes power losses during operation. This upgrade ensures stable and continuous functionality without significant voltage drops, even under high current loads, a critical factor during intense practice sessions and competitions.

To ensure compatibility with our motor's maximum voltage rating of 9V, we incorporated a step-down voltage regulator. This module efficiently converts the 11.1V input to a stable 9V output, maintaining the motor's safety while maximizing the extended runtime afforded by the 11.1V battery. This solution not only improves reliability but also instills confidence in the system's durability and performance under demanding conditions.

Technical information

-3 cells Voltage: 11.1V

-Capacity: 1300mAh 20C

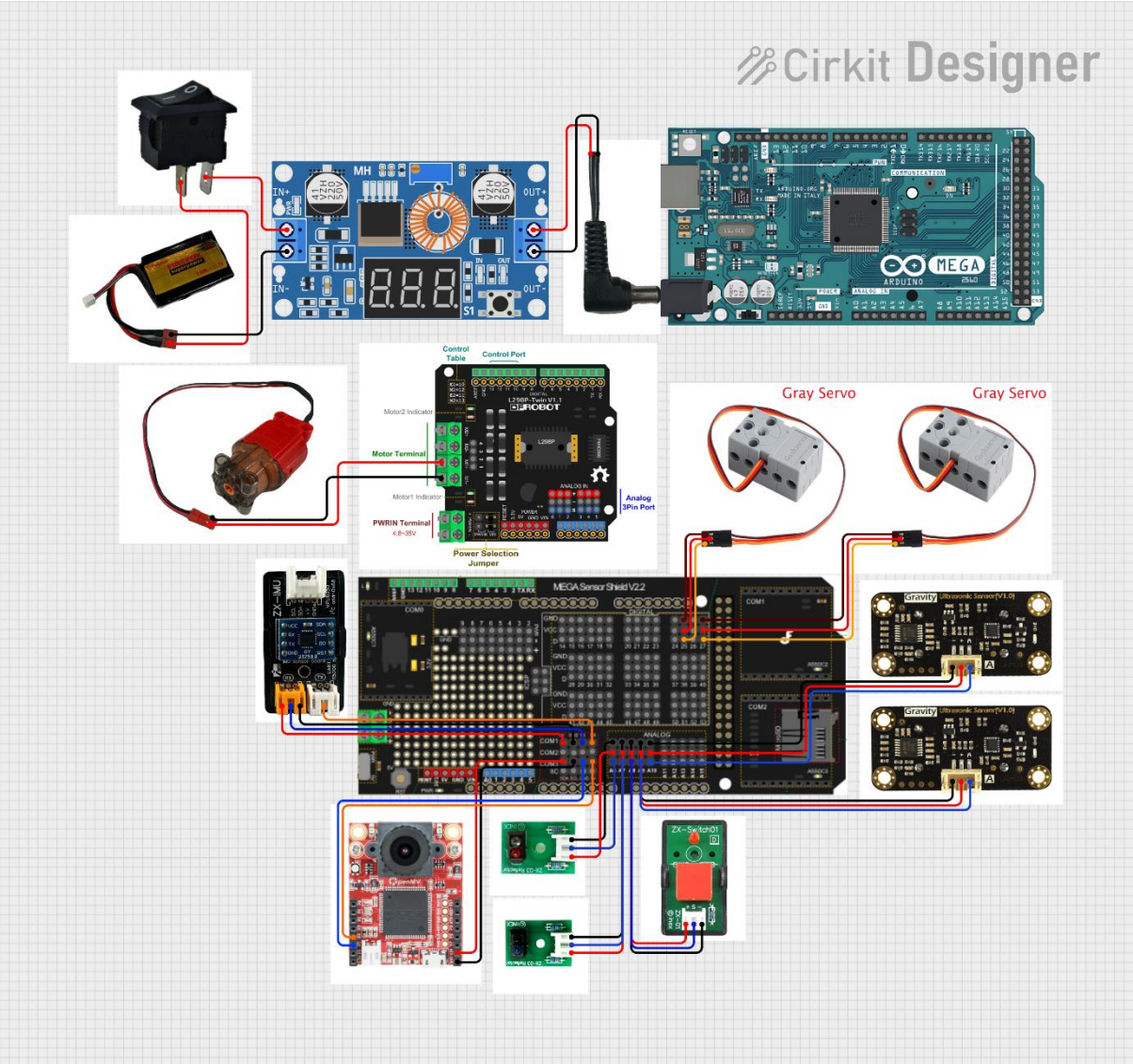
-Charging Current: Up to 5 times the capacity (5C)

-Connectors: Dean type, easily disconnect able

3.3Wiring Diagram and Power Distribution Diagram

Wiring Diagram

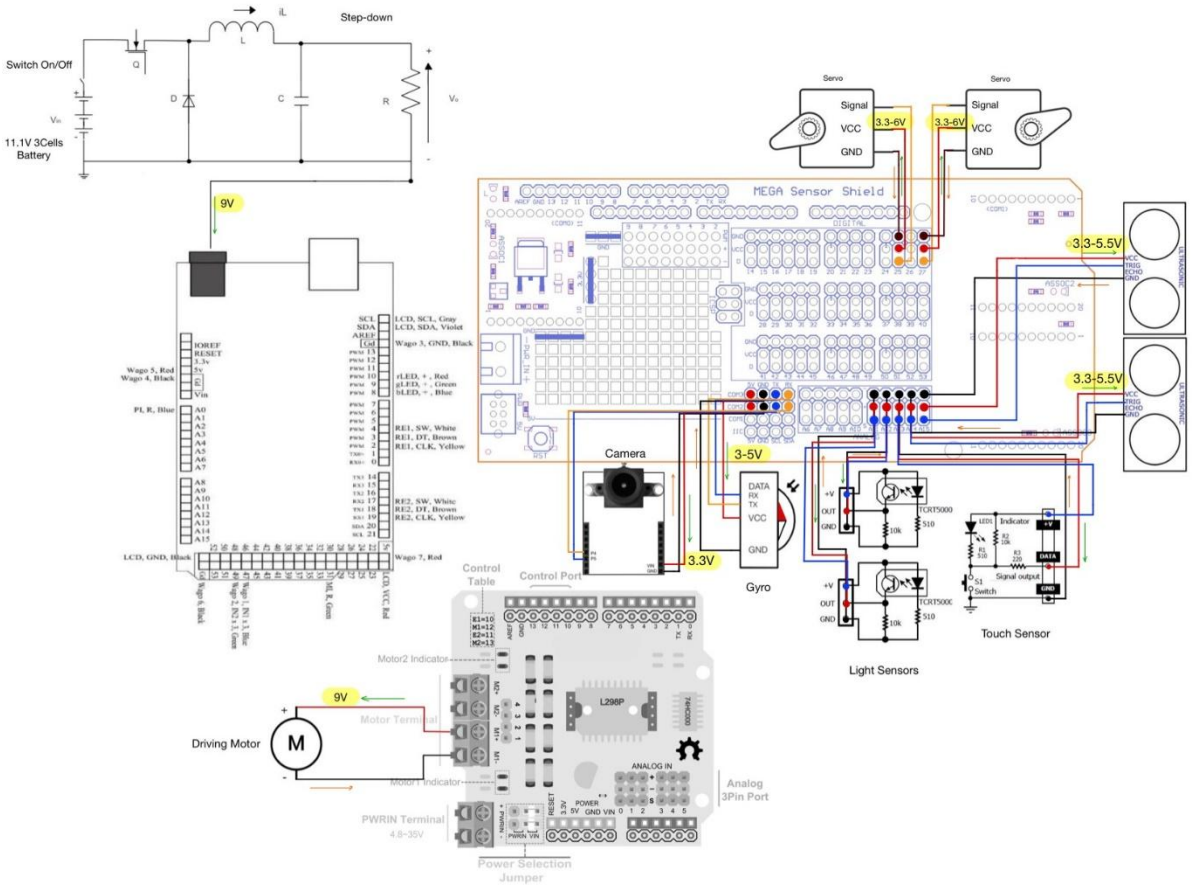
- **It shows you the connection of various components to the microcontroller.**



(Scan the QR Code below to view the image in higher resolution)

Power Distribution Diagram

- **It shows the ports of various components connected to the microcontroller, as well as the voltage levels supplied to the microcontroller and its connected components.**



(Scan the QR Code below to view the image in higher resolution)

4. Obstacle Management

How the robot works (Youtube Link)

To help you better understand what I'm explaining, you can view the video we created from this link https://youtu.be/qQTfzTyW7DM or scan the QR code below.

4.1 Open Challenge round

In this round, our robot must complete three (3) laps on the track with random placements of the inside track walls within 3 minutes.

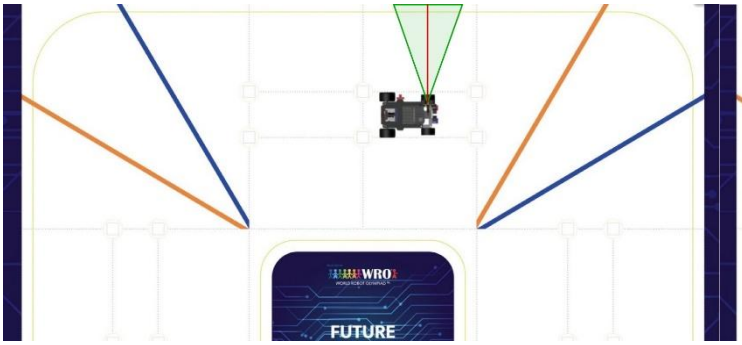
Open Challenge round (Youtube Link)

This video shows our robot completing the first round (Open Challenge), you can view the video we created from this link https://youtu.be/8S626QcRaPA or scan the QR code below.

The strategy

In this Open Challenge round, the robot will use the ultrasonic to measure the distance of the robot from the outer wall. The robot will use the color sensor to check the line color. If the first color is red, the robot will turn right, and if it's blue, the robot will turn left (because if the robot has to turn right, it will detect the red line first, and if it has to turn left, it will detect the blue line first, using different color sensors for detecting each line color). Every time, the robot will check how many times it has crossed the first line. It will count 12 lines (1 round = 4 lines), and after that, it will walk with the timer that we set until the time we set runs out.

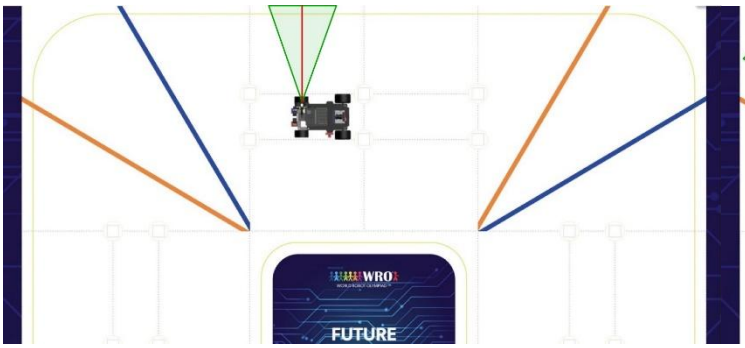
If the first line is red



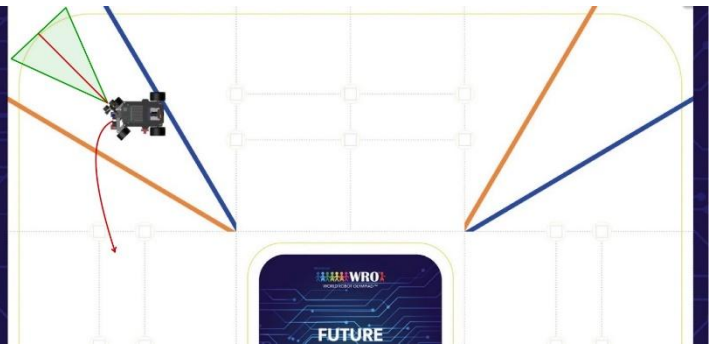
Start

When detect the first line

If the first line is blue



Start



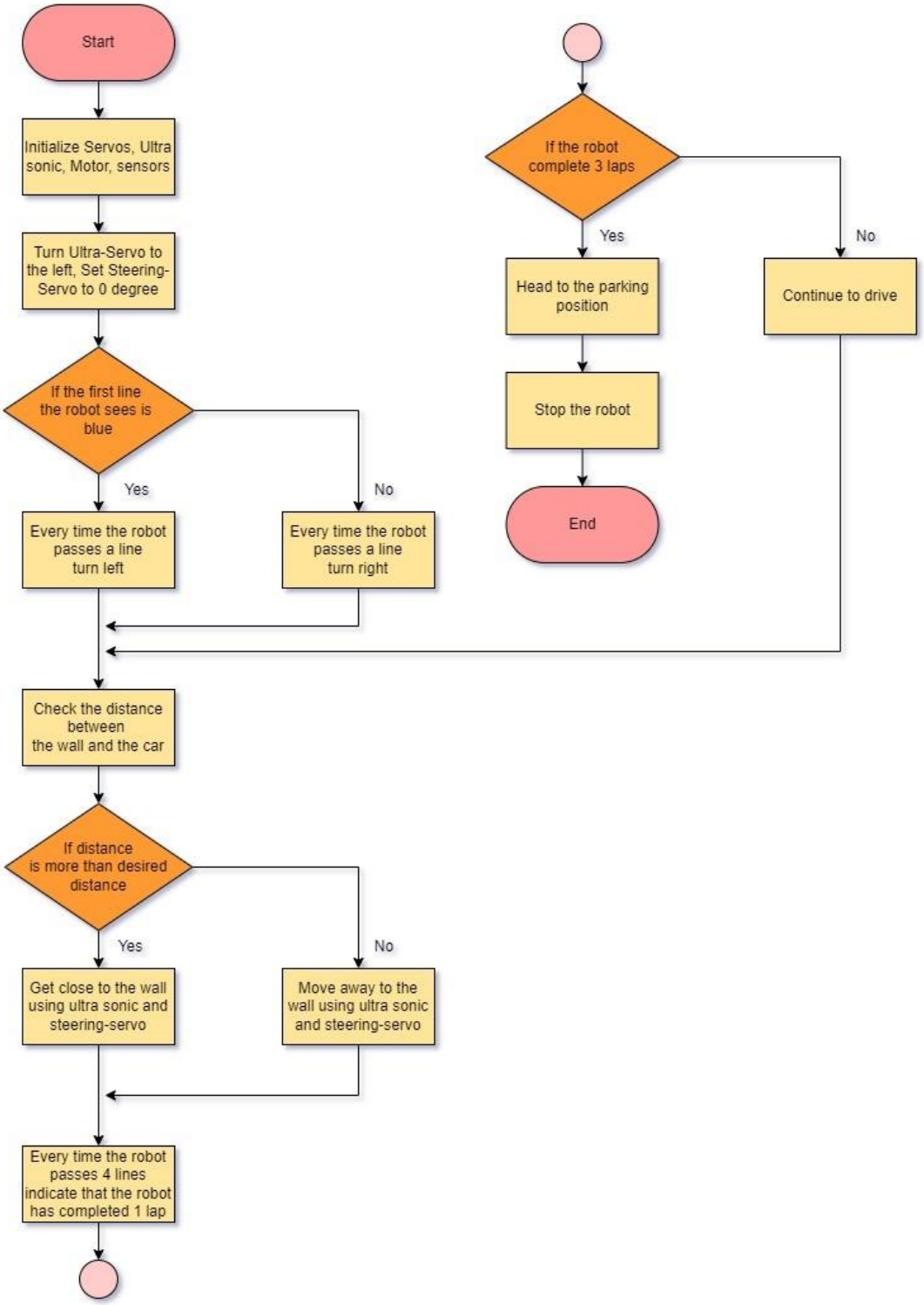
When detect the first line

The robot will use the distance of the robot from the wall and gyro sensor degree to calculate into steering degree (turning degree) to maintain the distance between the wall with PID (Proportional Integral Derivative)

$$U(t) = k_p e(t) + k_i \int e(t)\,dt + k_\rho \frac{de}{dt}$$

(PID Formular)

Flowchart [Open Challenge round]



Source Code

- **Section 1 [Open Challenge round]**

```
#include <Mapf.h>
#include <PID_v2.h>
#include <Wire.h>
#include <Servo.h>

Servo myservo;
Servo myservo2;

const int E1Pin = 10;
const int M1Pin = 12;

/**inner definition**/
typedef struct {
  byte enPin;
  byte directionPin;
} MotorContrl;

const int M1 = 0;
const int MotorNum = 1;

const MotorContrl MotorPin[] = { { E1Pin, M1Pin } };

const int Forward = LOW;
const int Backward = HIGH;


//Button
int BUTTON = A8;

//PID
PID_v2 compassPID(0.7, 0.0001, 0.05, PID::Direct);

//Ultra
int const ULTRA_PIN = A9;
int const ULTRA_PIN_II = A10;

//INEX Gyro
float pvYaw;
uint8_t rxCnt = 0, rxBuf[8];

//  Light Sensors
int const RED_SEN = A6;
int const BLUE_SEN = A7;

// Servo
int const STEER_SRV = 27;  //16
int const ULTRA_SRV = 25;  //23

//Others
char TURN = 'U';
long halt_detect_line_timer;
int Line_Number = 0;
int plus_degree = 0;
int count;
```

This code sets up a robot with servos, motors, sensors (light, ultrasonic, and gyro), and a PID controller for compass-based movement. It configures motor pins, light sensor pins, and controls turn direction and line detection timing to guide the robot's navigation.

- **Section 2 [Open Challenge round]**

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial1.begin(9600);
  compassPID.Start(0, 0, 0);
  compassPID.SetOutputLimits(-180, 180);
  compassPID.SetSampleTime(10);
  pinMode(STEER_SRV, OUTPUT);
  pinMode(ULTRA_SRV, OUTPUT);
  pinMode(ULTRA_PIN, INPUT);
  pinMode(RED_SEN, INPUT);
  pinMode(BLUE_SEN, INPUT);
  pinMode(BUTTON, INPUT);
  initMotor();
  while (!Serial)
    ;
  myservo.attach(ULTRA_SRV, 500, 2400);
  myservo2.attach(STEER_SRV, 500, 2500);
  steering_servo(0);
  ultra_servo(0, 'L');
  // check_leds();
  while (analogRead(BUTTON) > 500)
    ;
  zeroYaw();
}
```

The setup function initializes serial communication, PID control, motor, servo pins, and sensors. It calibrates the compass with zeroYaw() , waits for a button press, and prepares the system for operation by attaching servos and starting the PID control.

- **Section 3 [Open Challenge round]**

```
void loop() {
  float baseDesiredDistance = 25;
  while (analogRead(BUTTON) > 500) {
    getIMU();
    motor(50);
    Color_detection();
    float desiredDistance = baseDesiredDistance;  // Start with the base value
    if (TURN == 'L') {
      desiredDistance = 16;
    } else if (TURN == 'R') {
      desiredDistance = 17.5;
    }
    float distanceError = getDistance() - desiredDistance;
    float deadband = 2.0;
    if (abs(distanceError) < deadband) {
      distanceError = 0.0;
    }
    float directionFactor = (TURN == 'R') ? -1.0 : 1.0;
    float adjustedYaw = pvYaw - (distanceError * directionFactor);
    float pidOutput = compassPID.Run(adjustedYaw);
    steering_servo(pidOutput);
    ultra_servo(pvYaw, TURN);
    // Serial.println(getDistance());

    if (count >= 12) {
      long timer01 = millis();
      while (millis() - timer01 < 800) {
        motor(100);
        getIMU();
        Color_detection();
        float desiredDistance = baseDesiredDistance;  // Start with the base value
        if (TURN == 'L') {
          desiredDistance = 12;
        } else if (TURN == 'R') {
          desiredDistance = 14.5;
        }
        float distanceError = getDistance() - desiredDistance;
        float deadband = 2.0;
        if (abs(distanceError) < deadband) {
          distanceError = 0.0;
        }
        float directionFactor = (TURN == 'L') ? -1.0 : 1.0;
        float adjustedYaw = pvYaw + (distanceError * directionFactor);
        float pidOutput = compassPID.Run(adjustedYaw);
        steering_servo(pidOutput);
        ultra_servo(pvYaw, TURN);
      }
      motor(0);
      while (true) {
      }
    }
  }
  motor(0);
  while (analogRead(BUTTON) <= 500) {
  }
  while (analogRead(BUTTON) > 500)
    ;
  while (analogRead(BUTTON) <= 500)
    ;
}
```

The loop function controls the robot's movement by reading sensor values, adjusting motor speed, and steering using PID feedback. It checks the button status to start/stop, adjusts the robot's direction based on distance errors, and uses servos to steer. When a specific count is reached, it stops.

4.2 Obstacle Challenge round

In this round, our robot must complete three laps on a track marked with randomly placed green and red traffic signs.

- Red Obstacle: Keep to the right side of the lane.
- Green Obstacle: Keep to the left side of the lane.

The robot must not move any traffic signs. After finishing the three laps, the robot must find a parking lot and perform parallel parking.

Obstacle Challenge round (Youtube Link)

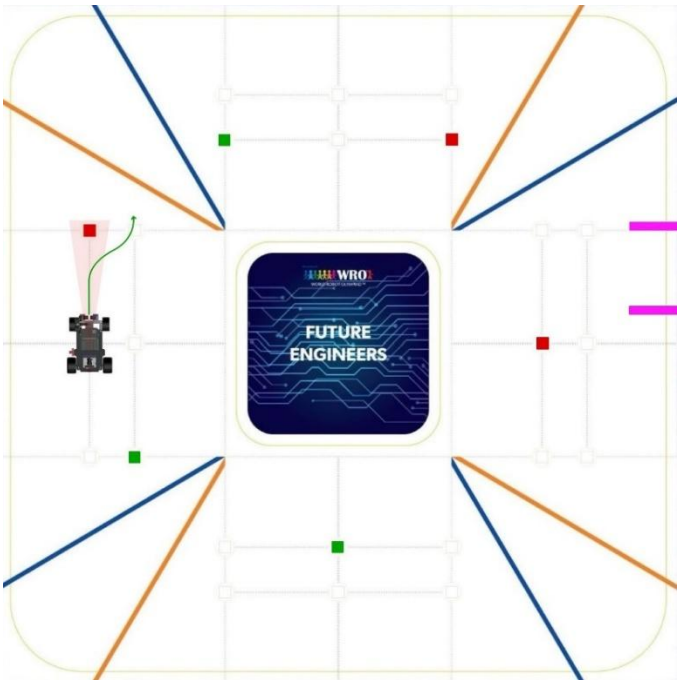
This video shows our robot completing the second round (Obstacle Challenge), you can view the video we created from this link https://youtu.be/9oGPSgff0DQ or scan the QR code below.

The strategy

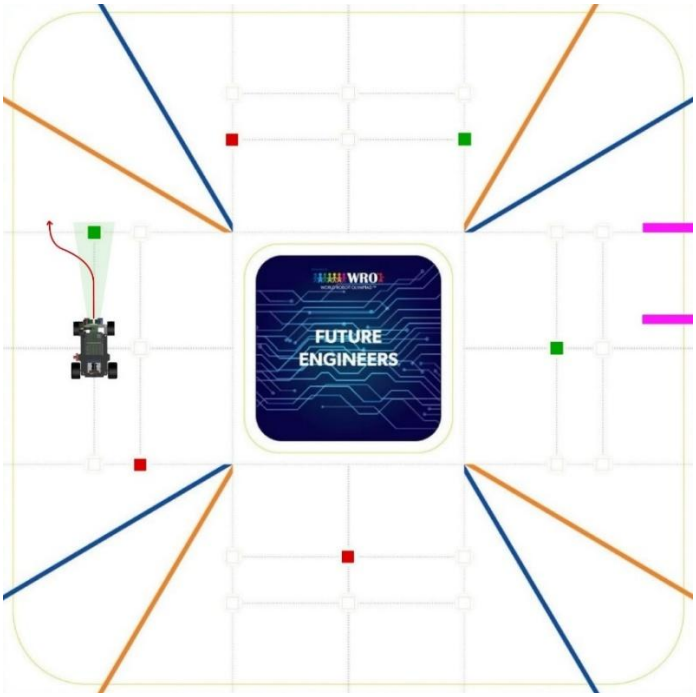
In the WRO 2025 Obstacle Challenge round, the robot uses a combination of ultrasonic sensors, a color sensor, a gyro, and an OpenMV Camera to navigate the course, detect and avoid obstacles, and maintain a safe distance from walls. The OpenMV Camera identifies obstacles and their colors, turning right for red obstacles and left for green ones, while the gyro ensures smooth and precise turns.

After completing the second round, the robot uses the OpenMV Camera to search for the purple parking area by detecting its color and comparing its size to the red and green pillars to determine its location. Once the third round is complete, the robot proceeds to park in the identified purple parking area, accurately positioning itself based on the location recorded by the OpenMV Camera.

If the robot sees red obstacle



If the robot sees green obstacle

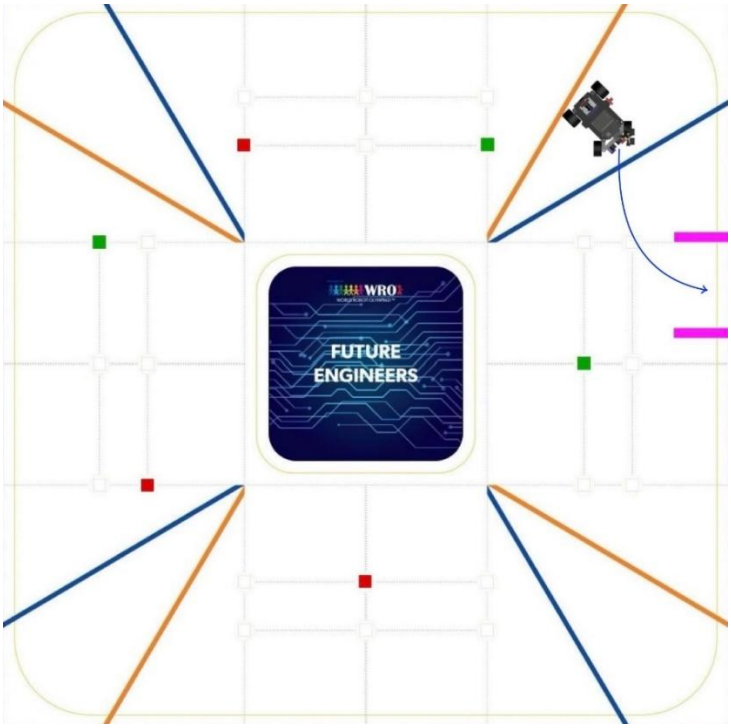


The robot still uses the PID to walk, but we added the avoidance degree to avoid the obstacles.

After completing the second round

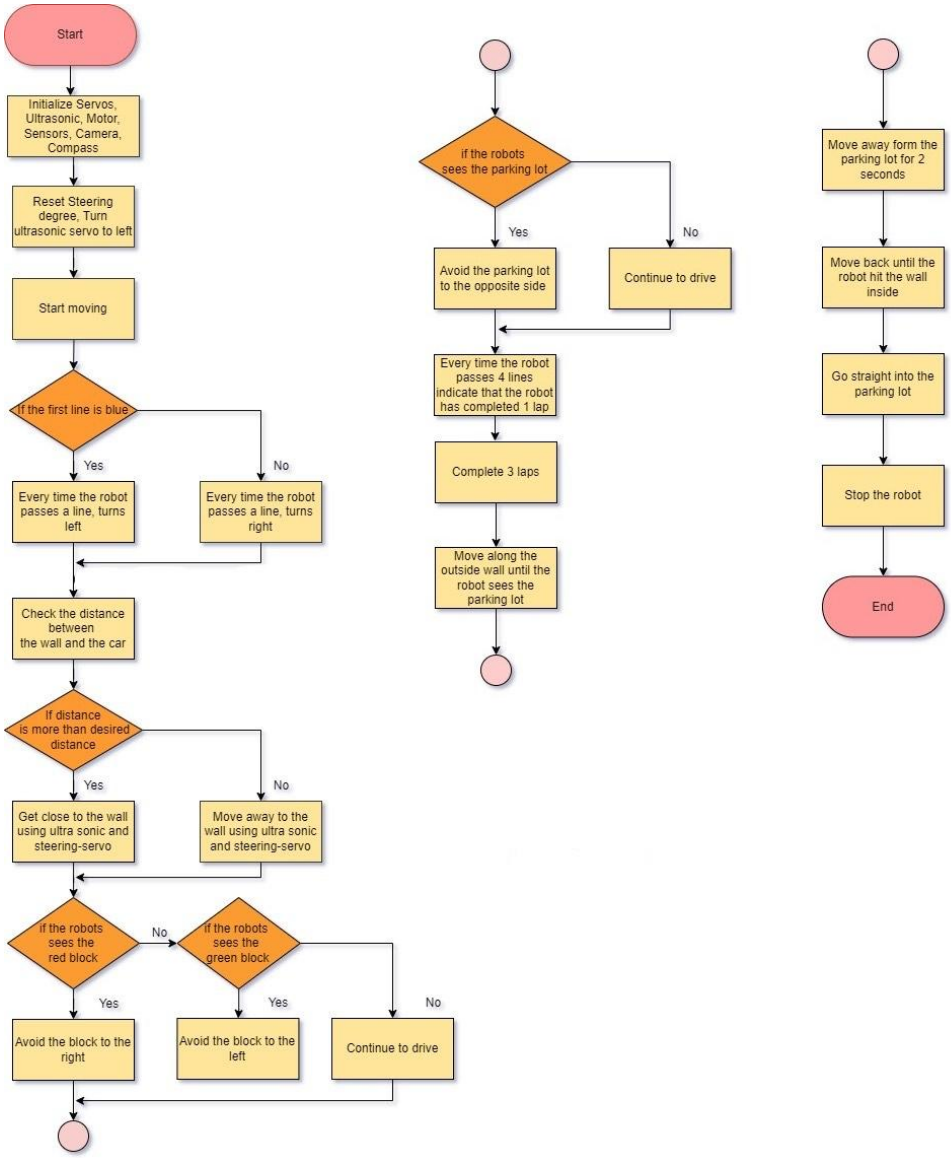
The robot uses the OpenMV Camera to search for the purple parking area, detecting its color directly and comparing its size to the red and green pillars to determine its position.

Park in parking area



The robot will drive to park in the purple parking area that was detected, using the stored position to accurately align itself.

Flowchart [Obstacle Challenge round]



(Scan the QR Code below to view the image in higher resolution)

Source Code

- **Section 1 [Obstacle Challenge round]**

```
#include <Mapf.h>
#include <PID_v2.h>
#include <Servo.h>
#include "CameraHandler.h"
```

We declare essential libraries for robot control: Mapf.h for Mapping the constrained distance from one range to another, PID_v2.h for smooth movement control, Servo.h for servo motor positioning, and CameraHandler.h for processing camera data. These libraries enable the robot to navigate, adjust movement, and interpret visual information effectively.

- **Section 2 [Obstacle Challenge round]**

```
CameraHandler camera;
BlobData blob;
BlobData purple_blob1;
BlobData purple_blob2;
```

We initialize a CameraHandler object called a camera to manage the camera's functions. It also creates three BlobData instances: blob for storing red and green pillar information, and purple_blob1 and purple_blob2 specifically for tracking two separate purple blobs. These variables enable the robot to detect, distinguish, and interact with multiple objects in its environment, particularly purple-colored ones.

- **Section 3 [Obstacle Challenge round]**

```
Servo myservo;
Servo myservo2;
```

We declare two Servo objects, myservo and myservo2, allowing control of two individual servo motors.

- **Section 4 [Obstacle Challenge round]**

```
const int E1Pin = 10;
const int M1Pin = 12;

typedef struct {
  byte enPin;
  byte directionPin;
} MotorContrl;

const int M1 = 0;
const int MotorNum = 1;

const MotorContrl MotorPin[] = { E1Pin, M1Pin };

const int Forward = LOW;
const int Backward = HIGH;
```

We set up motor control using E1Pin and M1Pin for power and direction. The MotorContrl structure and MotorPin array organize these pins, while Forward and Backward constants control motor rotation, making direction easy to manage.

- **Section 5 [Obstacle Challenge round]**

```
int const RED_SEN = 6;
int const BLUE_SEN = 7;
int const BUTTON = 8;
int const ULTRA_PIN = 9;
int const ULTRA_PIN_II = 10;
int const STEER_SRV = 27;
int const ULTRA_SRV = 25;
```

We connect Red sensor to port 6, Blue sensor to port 7, Button to port 8, Ultrasonic that measure distance between robot and the wall to port 9, Ultrasonic in front of the robot to port 8+10, Servo for steering port 27, and the last one, Servo for turning ultrasonic port 25.

- **Section 6 [Obstacle Challenge round]**

```
float pvYaw;
uint8_t rxCnt = 0, rxBuf[8];
```

We define pvYaw as a float to store the robot's yaw (orientation) angle. rxCnt is an 8-bit integer to count received data, and rxBuf is an 8-byte array to hold incoming data.

- **Section 7 [Obstacle Challenge round]**

```
long halt_detect_line_timer = 0;
long halt_detect_parking = 0;
long MV_timer = 0;

float found_parkAngle = 0;
float absYaw;
float uturnYaw;
float avoidance_degree;

int last_found_signature;
int plus_degree = 0;
int count_line = 0;
int parking_step = 0;
int parkingsection = -1;
int side = 1;
int hi = 0;
int angle = 115;

bool startpark = false;
bool parking = false;
bool next = false;
bool foundpark = false;
bool uturn = false;

char currentBlock = 'N';
char previousBlock = 'N';
char lastblock;
char lastfound = 'U';
char TURN = 'U';
char ULTRA_DIR = 'R';
```

This code above, we define several variables used for various control and tracking functions. long is used for defining time variable. float for variable that has decimal. int for variable that is integer. bool for variable that its output is true and false. char is for variable that is used to store data as a single character.

- **Section 8 [Obstacle Challenge round]**

```
void setup() {
  initialize_everything();
  while (analogRead(BUTTON) > 500)
    ;
  zeroYaw();
}
```

In void setup we initialize every part of our robot (function mentioned in another page) and then wait until the button is pressed. After that, reset the compass.

- **Section 9 [Obstacle Challenge round]**

```
void loop() {
  // Calculate camera errors
  camera.handleIncomingData();
  BlobData tempBlob = camera.getBlobData();
  if (tempBlob.signature == 1) {
    // RED
    last_found_signature = 1;
    blob = tempBlob;

  } else if (tempBlob.signature == 2) {
    // GREEN
    last_found_signature = 2;
    blob = tempBlob;

  } else if (tempBlob.signature == 3) {
    purple_blob1 = tempBlob;
  } else if (tempBlob.signature == 4) {
    purple_blob2 = tempBlob;
  }
```

This code processes data from the camera to identify and sort detected blobs by color. It first updates the camera data, then retrieves the current blob as tempBlob. The code checks the color of tempBlob based on its "signature": if it's red (1), it sets last_found_signature to 1 and stores tempBlob as blob. If it's green (2), it does the same but sets last_found_signature to 2. Purple blobs are handled separately, with purple_blob1 storing blobs marked as 3 and purple_blob2 storing blobs marked as 4.

- **Section 10 [Obstacle Challenge round]**

```
float avoidance_degree = 0;
if (tempBlob.signature == 3 && tempBlob.width / 3.9 > blob.width) {
  avoidance_degree = calculate_avoidance(tempBlob.signature, tempBlob.width, tempBlob.x, tempBlob.y) * -2;
} else {
  avoidance_degree = calculate_avoidance(blob.signature, blob.width, blob.x, blob.y);
}
```

In this code, avoidance_degree is set to 0 at the start. The code then checks if tempBlob has a signature of 3 (meaning it's a purple blob) and if it's wider than the main blob divided by 3.9. If both are true, it calculates an avoidance angle using tempBlob's data and multiplies it by -2 to create a stronger reaction in the opposite direction. If the conditions aren't met, it calculates a normal avoidance angle using red and green blob's data instead.

- **Section 11 [Obstacle Challenge round]**

```
int desiredDistance = parking_step == 0 ? (camera.isBlockFound() ? 20 : 40) : 15;
float distanceError = getDistance() - desiredDistance;
float frontDistance = getDistanceII();

float deadband = 2.0;
if (abs(distanceError) < deadband) {
  distanceError = 0.0;
}
float directionFactor = (ULTRA_DIR == 'R') ? -1.0 : 1.0;
float adjustedYaw = pvYaw - clamp(distanceError * directionFactor, -20, 20);
float pidOutput = compassPID.Run(adjustedYaw);
```

From code above, we declare variables called desireDistance to set the perfect distance between the robot and wall. If the parking step is not equal to 0 (it exits the main loop and going to perform parking) and it sees a block, set the desireDistance to 20 if not set to 40, and if it is in the main loop set it to 15. distanceError is to make the robot know that is it too far from desireDistance. deadband is to neglect small error. If distanceError is less than 2 then ignore them. Because we make the ultrasonic move left and right due to the block nearest to the robot, we need to add directionFactor to let the robot keep distance from the wall normally. The adjustedYaw is calculated by adjusting pvYaw using the distanceError and directionFactor, clamped between -20 and 20. Finally, the pidOutput is calculated using the compassPID controller to adjust the robot's steering.

- **Section 12 [Obstacle Challenge round]**

```
// TEST PARKING

if (count_line >= 8 && count_line < 12 && tempBlob.signature == 3 && parkingsection == -1) {
  parkingsection = count_line % 4;
} else if (count_line > 12) {
  if (parkingsection == 0) {
    parkingsection = 4;
  }
}
```

We tell the robot to find the parking lot while it's in the third lap. After it found the parking lot, divide the variable count_line by 4 then you get the remainder. That's the section of the field that the robot's in at the time. Store it in parkingsection.
If parkingsection equal to 0, change that to 4 to ensure that the robot park smoothly.

- **Section 13 [Obstacle Challenge round]**

```
if (parking_step == 1) {
    ULTRA_DIR = TURN == 'R' ? 'L' : 'R';
  }
```

While the robot searching for parking lot, change side that ultrasonic heads to. So, the ultrasonic scan outside wall.

- **Section 14 [Obstacle Challenge round]**

```
int parking_degree = ((purple_blob1.x + purple_blob2.x) / 2 - 160) * -0.5;
  int final_degree = camera.isBlockFound() ? mapf(min(max(getDistance(), 15), desiredDistance), 15, desiredDistance, pidOutput,
```

parking_degree is the variable that calculate degree of steering wheel for the robot to go into parking lot safely. final_degree is the variable that calculate the degree of steering wheel throughout the round. If it sees a block, avoid them. If it doesn't see anything, just continue using distance from the wall and compass.

- **Section 15 [Obstacle Challenge round]**

```
getIMU();
 color_detection();
```

This is the beginning, first, get compass value. Then, start detecting the first color of the line on the field after the robot start.

- **Section 16 [Obstacle Challenge round]**

```
switch (parking_step) {
  case 1:

    if (!startpark) {
      halt_detect_parking = millis();
      startpark = true;
    }
    if (tempBlob.signature != 3 && purple_blob1.width < 70) {
      steering_servo(pidOutput);
      ultra_servo(pvYaw, ULTRA_DIR);
      motor(40);
    } else {
      parking_step = 2;
      startpark = false;
    }
    break;
  case 2:
    halt_detect_line_timer = millis();
    if (!startpark) {
      halt_detect_parking = millis();
      startpark = true;
    }
    if (millis() - halt_detect_parking < 1400) {
      steering_servo(final_degree);
      ultra_servo(0, ULTRA_DIR);
      motor(40);
    } else {
      parking_step = 3;
      startpark = false;
      if (TURN == 'L') {
        plus_degree += 90;
      } else {
        plus_degree -= 90;
      }
    }

    break;

  case 3:

    if (!startpark) {
      halt_detect_parking = millis();
      startpark = true;
    }
    if (millis() - halt_detect_parking < 2000) {
      steering_servo(pvYaw);
      ultra_servo(0, ULTRA_DIR);
      motor(-40);
    } else {
      parking_step = 4;
      startpark = false;
    }

    break;

  case 4:
    if (frontDistance > 10) {
      steering_servo(mapf(clamp(frontDistance, 10, 20), 20, 10, parking_degree, 0));
      motor(mapf(clamp(frontDistance, 10, 20), 20, 10, 35, 30));
    } else {
      motor(30);
      delay(500);
      motor(0);
      while (true)
        ;
    }
    break;

  case 5:
    if (count_line >= 13 && count_line < 12 + parkingsection) {
      motor_and_steer(pidOutput);
      ultra_servo(pvYaw, TURN);
    } else {
      parking_step = 1;
    }
    break;

  default:
    motor_and_steer(final_degree);
    ultra_servo(pvYaw, ULTRA_DIR);
    if (count_line > 12 && parkingsection != -1) {
      parking_step = 5;
    }
    break;
}
```

This switch statement controls the robot's parking process based on the value of parking_step, dividing it into multiple stages: Case 1: The robot begins by detecting parking conditions. If the purple blob (signature 3) isn't detected or is too small (purple_blob1.width < 70), it adjusts the servo and ultrasonic direction, moving forward. When the condition changes, it moves to step 2. Case 2: The robot aligns itself. A timer (halt_detect_parking) tracks the duration, and the robot steers using final_degree while moving forward for 1400 milliseconds. Then, it advances to step 3, adjusting the turning angle (plus_degree) based on the direction that the robot turns. Case 3: The robot starts reversing for 2000 milliseconds, centering the steering and ultrasonic servos. Afterward, it proceeds to step 4. Case 4: The robot parks by steering based on frontDistance. If the distance is greater than 10, it adjusts steering and speed using a mapped value. Once close enough, it stops, waits briefly. Case 5: The robot ensures it aligns properly in the parking section. It adjusts motors and servos while tracking the line count. Once done, it resets to step 1 for further operation. But if no specific step applies, the robot moving as usual. If a parking section is found and the line count exceeds 12, it transitions to step 5.

Function

This is all the function of our program

Initialize Everything

```
void initialize_everything() {
  Serial.begin(19200);
  Serial1.begin(19200);
  Serial2.begin(19200);
  Serial3.begin(19200);

  compassPID.Start(0, 0, 0);
  compassPID.SetOutputLimits(-180, 180);
  compassPID.SetSampleTime(10);

  pinMode(STEER_SRV, OUTPUT);
  pinMode(ULTRA_SRV, OUTPUT);
  pinMode(ULTRA_PIN, INPUT);
  pinMode(RED_SEN, INPUT);
  pinMode(BLUE_SEN, INPUT);
  pinMode(BUTTON, INPUT);

  initMotor();
  while (!Serial)
    ;
  myservo.attach(ULTRA_SRV, 500, 2400);
  myservo2.attach(STEER_SRV, 500, 2500);
  steering_servo(0);
  ultra_servo(0, 'U');
}
```

The initialize_everything function sets up the robot by initializing serial ports, starting the compass PID controller with limits of -180° to 180°, and configuring pins for servos, sensors, and a button. It calls initMotor for motor setup, attaches servos with defined ranges, and resets them to default positions, ensuring the robot is ready to operate.

Degrees to radians

```
float degreesToRadians(double degrees) {
  return degrees * PI / 180.0;
}

float radiansToDegree(double raidans) {
  return raidans / PI * 180.0;
}
```

These functions convert angles between degrees and radians.

Avoidance Calculation(based on size and position)

```
float _cal_avoidance(char mode, int targetWidth, int objectWidth, int blockCenterX, int blockCenterY) {
  float focalLength = 2.8;
  float cameraFOV = 70;

  float distance = (targetWidth * focalLength * 100) / objectWidth;

  float deltaX = blockCenterX - (320 / 2);
  float deltaY = blockCenterY - (240 / 2);

  float detected_degree = -deltaX * cameraFOV / 320.0;

  float blockPositionX = distance * sin(degreesToRadians(detected_degree));
  float blockPositionY = distance * cos(degreesToRadians(detected_degree)) - 10;

  ULTRA_DIR = mode;

  if (mode == 'L') {
    return max(radiansToDegree(atan2(blockPositionX + (targetWidth / 2 + 10), blockPositionY)), 5) * 1.1;
  } else if (mode == 'R') {
    return min(radiansToDegree(atan2(blockPositionX - (targetWidth / 2 + 10), blockPositionY)), -5) * 0.9;
  } else {
    return 0;
  }
}
```

The _cal_avoidance function calculates the robot's avoidance angle based on the object's size and position. We use the camera's focal length and field of view to estimate the object's distance, then calculates the X and Y positions. Depending on whether the robot needs to turn left or right, we use trigonometry to compute the avoidance angle.

Avoidance Calculation(based on signature)

```
float calculate_avoidance(int signature, int objectWidth, int blockCenterX, int blockCenterY) {
  int avoidance_degree = 0;

  if (signature == 2) {
    avoidance_degree = _cal_avoidance('L', 5, objectWidth, blockCenterX, blockCenterY);
  } else if (signature == 1) {
    avoidance_degree = _cal_avoidance('R', 5, objectWidth, blockCenterX, blockCenterY);
  } else if (signature == 3 || signature == 4) {
    avoidance_degree = _cal_avoidance(TURN, 20, objectWidth, blockCenterX, blockCenterY);
  }
  return avoidance_degree;
}
```

The calculate_avoidance function determines the robot's avoidance direction based on the object's signature. For green (signature 2), it calculates a left turn; for red (signature 1), a right turn. For purple (signatures 3 and 4), it uses the robot's turn direction, if it turns right it avoids to the right. If it turns left, it avoids to the left.

Wrap value

```
int wrapValue(int value, int minValue, int maxValue) {
  int range = maxValue - minValue + 1;
  if (value < minValue) {
    value += range * ((minValue - value) / range + 1);
  }
  return minValue + (value - minValue) % range;
}
```

The wrapValue function adjusts a value to keep it within a specified range. If the value is below the minimum, it wraps around to the top of the range. Similarly, if the value exceeds the maximum, it wraps back around to the bottom. This ensures the value always stays within the minValue and maxValue range.

Initiate motor

```
void initMotor() {
  int i;
  for (i = 0; i < MotorNum; i++) {
    digitalWrite(MotorPin[i].enPin, LOW);

    pinMode(MotorPin[i].enPin, OUTPUT);
    pinMode(MotorPin[i].directionPin, OUTPUT);
  }
}
```

The initMotor function initializes the motors by setting up their pins. It loops through all motors and configures their enable and direction pins as output, while also setting their enable pins to LOW.

Set motor direction

```
void setMotorDirection(int motorNumber, int direction) {
  digitalWrite(MotorPin[motorNumber].directionPin, direction);
}
```

The setMotorDirection function sets the direction of a specific motor by adjusting its direction pin to either forward or backward.

Set motor speed

```
inline void setMotorSpeed(int motorNumber, int speed) {
  analogWrite(MotorPin[motorNumber].enPin, 255.0 * (speed / 100.0));
}
```

The setMotorSpeed function adjusts the motor's speed by using analogWrite to control the enable pin's duty cycle. It converts the given speed (from 0 to 100) to a value between 0 and 255, with 255 representing full speed.

Motor

```
void motor(int speed) {
  if (speed > 0) {
    setMotorDirection(M1, Forward);
    setMotorSpeed(M1, speed);
  } else {
    setMotorDirection(M1, Backward);
    setMotorSpeed(M1, speed);
  }
}
```

The motor function controls the motor's movement. If the speed is positive, it sets the motor to move forward; if negative, it sets the motor to move backward.

Color detection

```
void color_detection() {
  int blue_value = analogRead(BLUE_SEN);
  if (TURN == 'U') {
    int red_value = analogRead(RED_SEN);
    if (blue_value < 600 || red_value < 600) {
      int lowest_red_sen = red_value;
      long timer_line = millis();
      while (millis() - timer_line < 100) {
        int red_value = analogRead(RED_SEN);
        if (red_value < lowest_red_sen) {
          lowest_red_sen = red_value;
        }
      }
      if (lowest_red_sen > 600) {
        TURN = 'L';
        plus_degree += 90;
      } else {
        TURN = 'R';
        plus_degree -= 90;
      }
      halt_detect_line_timer = millis();
      count_line++;
    }
  } else {
    if (millis() - halt_detect_line_timer > 1800) {
      if (blue_value < 600) {
        if (TURN == 'R') {
          plus_degree -= 90;
        } else {
          plus_degree += 90;
        }
        halt_detect_line_timer = millis();
        count_line++;
      }
    }
  }
}
```

This function works on the direction that the robot goes during the round. First, if red sensor value is more than 600 and blue's is less than 600, it means that it found red line, so it has to turn right. If both of the sensor's value is less than 600, it means that it found blue line and has to turn left. The robot stores this data and next time it sees a line after 1.8 seconds, it turns the same direction.

Steering servo

```
void steering_servo(int degree) {
  myservo2.write((90 + max(min(degree, 50), -50)) / 2);
}
```

The function controls the steering servo's position. It takes a degree value, clamps it within the range of -50 to 50, and then adjusts it to fit the servo's expected range by adding 90 and dividing by 2.

Ultrasonic Servo

```
void ultra_servo(int degree, char mode_steer) {
  int middle_degree = 0;
  if (mode_steer == 'F') {
    middle_degree = 150;
  } else if (mode_steer == 'R') {
    middle_degree = 225;
  } else if (mode_steer == 'L' || mode_steer == 'U') {
    middle_degree = 80;
  } else {
  }
  myservo.write(mapf(max(min(middle_degree + degree, 225), 45), 0, 270, 0, 180));
}
```

The ultra_servo function controls the ultra servo based on the given degree and steering mode. It first sets a middle degree based on the mode_steer parameter: 'F' sets it to 150, 'R' to 225, and 'L' or 'U' to 80. Then, it adjusts the servo position by adding the input degree to this middle degree.

Get distance

```
float getDistance() {
  float raw_distance = mapf(analogRead(ULTRA_PIN), 0, 1023, 0, 500);
  if (TURN == 'L') {
    raw_distance += 0;
  } else if (TURN == 'R') {
    raw_distance -= 0;
  }
  return min(raw_distance, 50);
}

float getDistanceII() {
  float raw_distance = mapf(analogRead(ULTRA_PIN_II), 0, 1023, 0, 500);
  if (TURN == 'L') {
    raw_distance += 0;
  } else if (TURN == 'R') {
    raw_distance -= 0;
  }
  return min(raw_distance, 50);
}
```

The getDistance and getDistanceII functions measure the distance using ultrasonic sensors. Both functions use analogRead to get a sensor value, then map it to a distance range (0 to 500). The final distance is capped at a maximum of 50 units using min.

Get IMU

```
bool getIMU() {
  while (Serial1.available()) {
    rxBuf[rxCnt] = Serial1.read();
    if (rxCnt == 0 && rxBuf[0] != 0xAA) return;
    rxCnt++;
    if (rxCnt == 8) {
      rxCnt = 0;
      if (rxBuf[0] == 0xAA && rxBuf[7] == 0x55) {
        pvYaw = (int16_t)(rxBuf[1] << 8 | rxBuf[2]) / 100.f;
        pvYaw = wrapValue(pvYaw + plus_degree, -179, 180);
        return true;
      }
    }
  }
  return false;
}
```

The getIMU function reads data from the Serial1 interface, expecting a specific format for communication with the IMU. It stores incoming bytes in the rxBuf array. The yaw value is then adjusted by adding plus_degree and wrapped within the range of -179 to 180 degrees using the wrapValue function.

Zero Yaw

```
void zeroYaw() {
  Serial1.begin(115200);
  delay(100);
  Serial1.write(0XA5);
  delay(10);
  Serial1.write(0X54);
  delay(100);
  Serial1.write(0XA5);
  delay(10);
  Serial1.write(0X55);
  delay(100);
  Serial1.write(0XA5);
  delay(10);
  Serial1.write(0X52);
  delay(100);
}
```

The zeroYaw function is used to reset or calibrate the yaw value of the IMU (Inertial Measurement Unit).

Motor and Steering

```
void motor_and_steer(int degree) {
  degree = clamp(degree, -52, 52);
  steering_servo(degree);
  motor((map(abs(degree), 0, 30, 49, 49)));
}
```

The motor_and_steer function adjusts the robot's steering by clamping the degree between -52 and 52, then sets the servo position. It also controls the motor speed, setting it to 49 based on the absolute value of the degree.

Clamp

```
float clamp(float value, float minVal, float maxVal) {
  if (value < minVal) return minVal;
  if (value > maxVal) return maxVal;
  return value;
}
```

The clamp function limits a given value to stay within the range specified by minVal and maxVal. If the value is smaller than minVal, it returns minVal; if it's larger than maxVal, it returns maxVal; otherwise, it returns the original value.

Check LEDs

```
void check_leds() {
  while (true) {
    Serial.print("Geen: ");
    Serial.print(analogRead(BLUE_SEN));
    Serial.print("   Red: ");
    Serial.println(analogRead(RED_SEN));
  }
}
```

This function is for checking sensors value.

Angle Difference

```
float angleDiff(float a, float b) {
  float diff = a - b;
  while (diff > 180) diff -= 360;
  while (diff <= -180) diff += 360;
  return diff;
}
```

The angleDiff function calculates the difference between two angles a and b, ensuring the result is within the range of -180° to 180°. If the difference exceeds these limits, it wraps around by adding or subtracting 360° to keep the result within the expected range.

OpenMV

- **Section 1 [OpenMV]**

```
import time
import sensor
import display
from pyb import UART
```

The code imports libraries for time delays, sensor handling, display management, and UART communication, typically used in microcontroller projects with cameras and displays.

- **Section 2 [OpenMV]**

```
# Initialize sensor
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)  # 320x240 resolution

sensor.skip_frames(time=2000)  # Wait for settings to take effect.

# Lock auto-exposure and auto-white-balance to prevent drift across reboots.
sensor.set_auto_gain(False)  # Disable auto gain.
sensor.set_auto_whitebal(False)  # Disable auto white balance.

# Lock exposure to prevent fluctuations in lighting conditions.
sensor.set_auto_exposure(False, exposure_us=7000)  # Adjust exposure manually.


sensor.set_contrast(3)
sensor.set_brightness(0)
sensor.set_saturation(0)

sensor.skip_frames(time=1000)

#sensor.__write_reg(0x0E, 0b00000000)  # Disable night mode
#sensor.__write_reg(0x3E, 0b00000000)  # Disable BLC
```

This code configures the sensor with fixed settings: RGB565 format, QVGA resolution (320x240), manual exposure, and disabled auto-gain and white balance. It also adjusts contrast, brightness, and saturation for stable imaging.

- **Section 3 [OpenMV]**

```
# Color thresholds
GREEN_THRESHOLDS = [(22, 48, -52, -21, 17, 49)]
RED_THRESHOLDS = [(0, 52, 13, 37, -3, 25)]
PURPLE_THRESHOLDS = [(33, 70, -25, 14, 42, 76)]
```

These threshold values define color ranges for detecting green, red, and purple objects based on the HSV color model

- **Section 4 [OpenMV]**

```
# Region of Interest (ROI)
ROI = (0, 160, 320, 240)

# Initialize UART
uart = UART(3, 19200, timeout_char = 2000)
clock = time.clock()
```

This code initializes the Region of Interest (ROI) for image processing and sets up the UART communication at a baud rate of 19200 with a timeout of 2000 milliseconds. The clock is also initialized to manage timing for the processing loop.

- **Section 5 [OpenMV]**

```
def send_blob_data(blob, blob_type, color):
    img.draw_rectangle(blob.rect(), color=color)
    img.draw_cross(blob.cx(), blob.cy(), color=color)
    data = f"{blob.cx()},{blob.cy()},{blob.w()},{blob.h()},{blob_type}\n"
    uart.write(data)
    print(data)
```

This function send_blob_data processes and sends information about a detected blob over UART. It draws a rectangle and a cross on the image at the blob's coordinates, using the specified color. The blob's position (cx, cy), size (width, height), and type are formatted as a string and sent via UART.v.

- **Section 6 [OpenMV]**

```
def send_no_blob_data():
    data = "0,0,0,0,0\n"
    uart.write(data)
    print(data)
```

The send_no_blob_data function sends a default "no blob" data string over UART. It indicates that no blob was detected by setting all blob-related values (position, size, and type) to zero.

- **Section 7 [OpenMV]**

```
while True:
    clock.tick()
    img = sensor.snapshot()

    # Detect red and green blobs
    green_blobs = img.find_blobs(GREEN_THRESHOLDS, roi=ROI, area_threshold=30, pixels_threshold=30, merge=True)
    red_blobs = img.find_blobs(RED_THRESHOLDS, roi=ROI, area_threshold=30, pixels_threshold=30, merge=True)

    # Find the largest blob between red and green blobs
    largest_green = max(green_blobs, key=lambda b: b.area(), default=None)
    largest_red = max(red_blobs, key=lambda b: b.area(), default=None)

    # Determine the largest blob between red and green
    largest_blob = None
    if largest_green and largest_red:
        largest_blob = largest_green if largest_green.area() > largest_red.area() else largest_red
    elif largest_green:
        largest_blob = largest_green
    elif largest_red:
        largest_blob = largest_red

    # Send data for the largest red or green blob (if found), else send '0'
    if largest_blob:
        blob_type = 2 if largest_blob in green_blobs else 1  # Green = 2, Red = 1
        color = (0, 255, 0) if blob_type == 2 else (200, 0, 0)
        send_blob_data(largest_blob, blob_type, color)
    else:
        send_no_blob_data()  # Send 0 when no red or green blobs are found

    # Detect and send data for all purple blobs
    purple_blobs = img.find_blobs(PURPLE_THRESHOLDS, roi=ROI, area_threshold=30, pixels_threshold=30, merge=True)
    purple_blobs = sorted(purple_blobs, key=lambda b: b.cx())

    if purple_blobs:
        for i, purple_blob in enumerate(purple_blobs):
            color = (255, 0, 255) if i == 0 else (252, 244, 3)  # Alternate colors for blobs
            send_blob_data(purple_blob, 3 + i, color)  # Different blob types for each purple blob
    cropped_img = img.crop(roi=ROI)  # Crop to exclude the top 120 pixels
    # Optional: Read and print incoming UART data
    if uart.any():
        data = uart.readline()
        print("Received:", data)
```

This code captures images, detects red, green, and purple blobs, and sends their data via UART. It finds the largest green or red blob and sends its position and size, drawing a rectangle and cross on the image. If no blobs are found, a "no blob" signal is sent. Purple blobs are also detected and sent with alternating colors and types. The image is cropped to focus on the lower half, and incoming UART data is printed when received. This loop enables real-time object tracking.

5.Link and Video

5.1 GitHub

GitHub (FE_YB_Sunflower) - https://github.com/ThanyawutII/FE_YB_Sunflower/tree/main

5.2 Picture

Wiring Diagram - https://github.com/ThanyawutII/FE_YB_Sunflower/blob/main/Schemes/Wiring%20Diagram.png

Power Distribution Diagram - https://github.com/ThanyawutII/FE_YB_Sunflower/blob/main/Schemes/Wiring%20Diagram.png

Flowchart [Obstacle Challenge round] - https://github.com/ThanyawutII/FE_YB_Sunflower/blob/main/FlowChart/ObstecleChallenge.jpg

5.3 Video (YouTube)

How the robot works - https://www.youtube.com/watch?v=qQTfzTyW7DM

Open Challenge round - https://youtu.be/8S626QcRaPA

Obstacle Challenge round - https://youtu.be/9oGPSgff0DQ