Haruto-Kaori / III-Wind

Code Issues Pull requests Actions Projects Security Insights Settings

Phoenician

www.youtube.com/@setekiteam

0 stars 0 forks 1 watching Branches Activity Tags

Private repository

main 1 Branch 0 Tags Go to file Go to file Add file Code

Haruto-Kaori Update README.md 6fe8663 · 2 months ago

3D-models	Add files via upload	2 months ago
ARA Versions	Update readme.md	2 months ago
documentation	Update README.md	2 months ago
electrical-schemes	Update README.md	2 months ago
robot-photos	Update README.md	2 months ago
src	Add files via upload	2 months ago
team-photos	Update README.md	2 months ago
video	Update README.md	2 months ago
README.md	Update README.md	2 months ago

README

Seteki® Team 🤖

Looking Up

INSTAGRAM YOUTUBE GITHUB

This is the official repository of the Seteki® team that will represent Instituto Episcopal San José in the WRO category: Future Engineers 2025, Panama.

Overview of our repository folders 📜

```
📦 WRO2025 Seteki
├── 📁 3D-models           # Contains 3D design files for the robot's components
│   ├── 📁 All 3D files    # All robot versions
│   └── 📄 README.md       # 3D models explained
├── 📁 Documentation       # All about our media
│   ├── 📁 Regional[1]     # Regional 1 Git Hub Printable
│   └── 📄 README.md       # Explanation
├── 📁 ARA Versions        # Robot ARA versions
│   ├── 📁 ARA 1.0         # 1.0
│   ├── 📁 ARA 1.1         # 1.1
│   └── 📄 README.md       # Explanation
├── 📁 electrical-schemes  # Contains circuit files
│   ├── 📁 Diagrams        # All the circuit diagrams
│   └── 📄 README.md       # Explanation
├── 📁 robot-photos        # All the robot photos
│   ├── 📁 Photos          # Every Photo
│   └── 📄 README.md       # Explanation
├── 📁 src                 # All the scripts of our robot
│   ├── 📁 First round     # First round code
│   ├── 📁 Second round    # Second round code
│   └── 📄 README.md       # Explanation
├── 📁 team-photos         # Photos of seteki members.
│   ├── 📁 Normal Photo    # Seteki formal photo
│   ├── 📁 Funny Photo     # Seteki random photo xd
│   └── 📄 README.md       # Explanation
├── 📁 video               # Robot videos
│   └── 📄 README.md       # Explanation
└── 📄 README.md           # Main documentation for the project
```

- `3D-models` - includes all the 3d printed parts of the robot that will go to the regional competition, and also contain and explanation about every piece designed Here.
- `documentation` - Here you'll find all the GitHub Printables for every competition. Here.
- `ARA Versions` - Here you'll see every version of our robot. Here
- `electrical-schemes` - contains the circuit diagrams of all the electrical parts of the seteki robot (ARA 1.1). Here.
- `robot-photos` - photos of every angle of our robot for you can see it. Here
- `src` - this folder contains every code that makes our robot work, it have explanation for every part of the codes. Here
- `team-photos` - photos of the seteki team, it has the normal photo and one funny photo. Here
- `video` - the link to our youtube channel where you can see our robot in action completing both challenges. Here.
- `README.md` - Here's all our journey in the development of our robot here we explain every part of the robot making. Here.

> ⓘ **Caution**
>
> Due to GitHub problems some charts and GIFs can load a little bit slow, so we beg you to be patient while reading to not miss anything, THANKS!

Part 1 / About Us 🗣️

We are Team Seteki, proudly representing Instituto Episcopal San José in Panama's World Robot Olympiad (WRO) Future Engineers 2025 category. Our mission is to blend technology and innovation, knowing that innovation can go hand-in-hand with responsibility. The name Seteki comes from the Atelopus zeteki.

> ⓘ **Note**
>
> Explication of the name after `About us` part.

c

Team Name Explication 🥸

The name of our Team (Seteki), comes from an animal called " ***Atelopus zeteki*** ". It is a small species of anuran amphibian of the Bufonidae family.

When we chose this name, it was because we are people who are using the possible resources to develop these robots and technology, but sometimes we forget that all this comes from natural resources and we are taking advantage of all this and we do not take care of it, and as a small annex we wanted to give this message that we also have a responsibility to the world and in addition to this we also want to have our country represented in our team.

> ⚠ **Warning**
>
> This species is in danger of extinction, more information here.

Team Presentation 🤑

Jean 🎶

Age: 17

— Name: Jean Paul Sosa Cruz.

— Team Role: 3D Designer, Git Hub Documentation, Part of the electrical circuits, Robot logic.

— Hobbys: Guitar, Bass, Music, Cinema, Soccer, Robotics, Books ngl.

— Goals With The Team: To grow together, innovate and to have fun btw.

— Personal Trayectory: 2022 WRO Germany, Music Band for 7 years, 2017 Kodu Panama Oeste, 2024 WRO TUrkiye.

Jonathan ⚽

Age: 15

— Name : Jonathan Michael Sosa Cruz.

— Team Role: Electronics and circuits, The mechanic of the team, Scheme designer.

— Hobbys: Soccer, Video Games, Bass.

— Goals With the team: he just wants the competition to have good food.

— Personal Trayectory: WRO Germany 2022, Member of the music band, several soccer teams.

Anthony 🤓 👆

Age: 17

— Name: Anthony Bruce Gómez Arjona.

— Team Role: Programmer, Robot logic, Part of the electrical circuits.

— Hobbys: Physics, learning in youtube, Science, music.

— Goals with the team: Bring something totally different to this competition and try to win and to get free food.

— Personal Trayectory: 2024 honorable mention at the physics nationals, 2024 3rd Place Senacyt science fair, Member of the music band.

> ⓘ **Note**
>
> To see our funny photo you can press here

Tutor: Edgar Ruiz 🤖

Age: 26

- Name: Edgar Antonio Ruiz Caballero

- Goals with the team: Wants to innovate with the team, grow as a team.

- Personal Trayectory: - Active participant in robotics competitions such as RoboCup Jr , WRO, and others since 2015. - National Winner of RoboCup Jr 2017 and team coachrepresenting Panama in Nagoya, Japan. - Third Place National Winner in WRO 2018 as team coach for The Oxford School. - National Winner of RoboCup Jr 2018 and team coach representing Panama in Sydney, Australia in 2019. - Participant in 2021 STEAM training hosted by Grupo Editorial EDELVIVES in Madrid, Spain. - National Winner of WRO 2022 and team coach representing Panama in Dortmund, Germany. - National Winner of WRO 2024 (World Robot Olympiad) and team coach representing Panama in Izmir, Turkey.

Part 2 / Robot Chasis 🚙

In this part 2 we are going to explain everything that supports our robot, that is the base, how we made it, the reason for each part, the platforms we used to design everything related to the chassis, all the implements we used physically, and how each mechanical part of our cart or robot works.

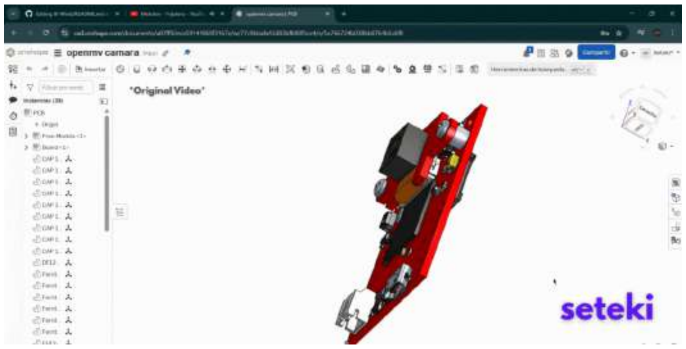
Platforms we use

For the chassis of our cart in our team we decided to adjust the chassis based on our needs, and that function is fulfilled by 3D printing, as it allows us to make the parts just to the extent in which we require them, as a first point we use two platforms for 3D modeling, OnShape and `FUSION` , although most of the robot is made in Onshape.

OnShape	Autodesk FUSION 360

OnShape:

Onshape is a cloud-based 3D CAD platform that allows real-time collaboration and requires no installation, as it runs entirely in a web browser. It supports cross-platform access from any device, making teamwork easy and efficient. Key advantages include built-in version control, secure cloud storage, and integrated tools for design and data management. With fast iteration, instant sharing, and automatic backups, Onshape streamlines modern product development.



Most of our robot is made on this platform, in which we already gave a little introduction about it, when we go breaking down each 3D part of the robot will be explained.

FUSION 360:

Fusion 360 is a cloud-based 3D CAD, CAM, and CAE software developed by Autodesk. It combines design, engineering, simulation, and manufacturing in one platform. Users can create precise 3D models, assemblies, and technical drawings. Its parametric and freeform modeling tools allow flexibility and fast changes. Fusion 360 also offers simulation features to test stress, motion, and heat. With built-in CAM, it supports CNC machining up to 5-axis. It includes PCB and electronics design, making it ideal for integrated projects. Cloud collaboration enables real-time teamwork and version control. Fusion 360 is user-friendly, cross-platform, and widely used by students, engineers, and makers.

Anthony modeling gears in fusion*
This was for practice
GitHub
seteki

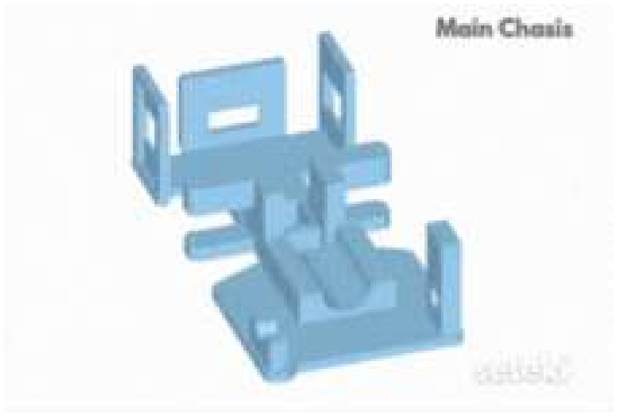
the major use we made of FUSION 360 was to model the gears that we implemented in the robot, the use of FUSION was important for the team to model, and to implement `FUTURE IMPROVEMENTS` for our robot.

Robot Main Chasis support

For the base of our robot, we decided to design it on the OnShape platform, and the main points we wanted to cover to design our main chassis are those of:

▶️ `Lightweight model`
▶️ `Friendly and integrated model at the time of physical assembly.`
▶️ `Futuristic but practical and simple model.`

> ⓘ **Note**
>
> To see each 3D model of our robot on GitHub you can click `HERE` , or go directly to the folder of `3D models` of our robot, or you can also see them graphically in the `README.md` and below each image or `GIF` of our 3D pieces open a link to see the modeling shown, Thank you :)



For the base of our robot we did it on the basis of four homogeneous parts, which would be:

▶️ `Ultrasonic Sensors`
▶️ `Steering`
▶️ `Motor and shaft`
▶️ `Top plate`

Steering Parts

Our 3D `Steering` made in onshape, was made with the plans to make it in a simple but functional way, and it is mainly based on an `Ackerman system`, but as you can clearly see in our address, we only took the principle because in itself it has quite a few differences but in essence it is the base of an `Ackerman system`.

> **Important**
>
> To more details in the explication of `Ackerman System`, PLease go Here.

The Ackermann steering system is a geometric arrangement used on vehicles with two steerable front wheels, designed so that during a turn all four wheels roll along concentrc arcs without lateral slip. Key points:

▶️ `Fundamental Objective`: Ensure that the extended axes of the inner and outer front wheels intersct at the vehicle's Instantaneous Center of Curvature (ICC), so each wheel follows its own circular path.

▶️ `Geometry`:

$$\delta_{f,in} = \tan^{-1}\left(\frac{L}{R-\frac{T}{2}}\right) \quad \delta_{f,out} = \tan^{-1}\left(\frac{L}{R+\frac{T}{2}}\right)$$

The equation shown define the ideal steering angles for the inner and outer front wheels (δ_n and δ_o) in an Ackermann geometry.

L is the wheelbase (distance between front and rear axles).

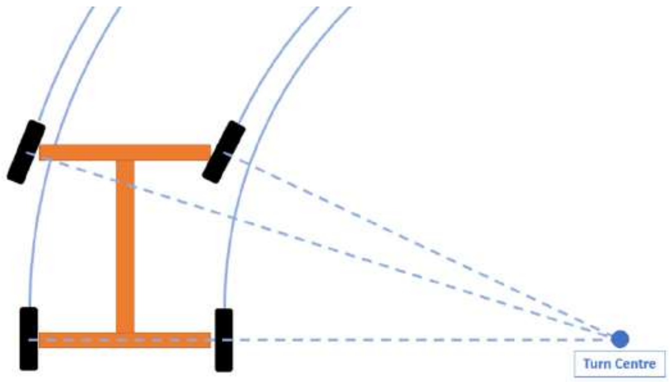
T is the front track (lateral spacing between the two front wheels).

R is the desired turning radius measured from the midpoint of the rear axle to the curve's center.

When you turn, the inner wheel must pivot more sharply than the outer one, because it follows a tighter circular path (radius = R − T/2) while the outer wheel follows a looser path (radius = R + T/2). Taking the arctangent of the wheelbase divided by each effective radius yields the exact mechanical angle each wheel must achieve so that all wheels roll without sideways slip. This precise relationship minimizes tire scrubbing, distributes cornering forces correctly, and ensures the vehicle tracks smoothly through turns.

▶️ `Mechanical Implementation` : Steering linkage ("tie rod") connects the two steering knuckles. A fixed pivot point on the chassis or steering arm drives the tie rod so that lateral movement produces the correct differential angles between inner and outer wheels.

▶️ `Ackermann Angle` : The steering arms (on the knuckles) are canted such that imaginary lines drawn through them meet at a point on the rear axle. The exact cant angle depends on track width and the chosen tie-rod attachment points.



Motor & movement parts:

The part of the motor and the movement of the rear axle was what complicated us a little more in the summer when we started with the approach of our small robot, the part of the mobilization will have `MAJOR OPTIMIZATIONS` in the future.

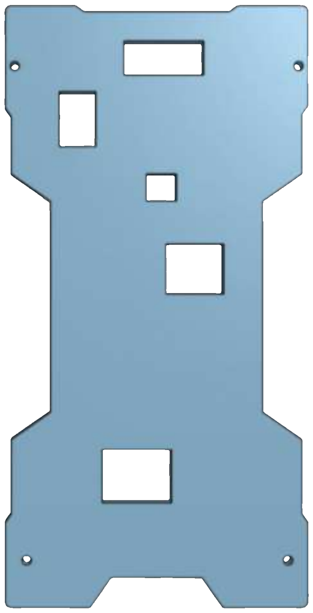
This motor part assembly it's formed by a electric motor rigidly mounted to the chassis whose vertical output shaft carries a bevel pinion that meshes at 90° with a larger bevel-gear crown affixed to a transverse drive shaft; this bevel pair redirects motor torque into the shaft, which spans the chassis and drives both rear wheels solidly linked (no differential). The drive shaft is supported by pillow-block bearing housings containing bearings or bushings that center it, resist radial loads, and minimize friction, while shaft-to-housing adapter blocks ensure precise alignment and easy removal for maintenance. Fasteners and keys lock the gears and shaft components together, preventing slippage under load; as the motor turns, torque flows through the bevel gears into the shaft, spinning the rear axle smoothly and reliably even under moderate loads.

> ⓘ **Caution**

Due to GitHub problems some charts and GIFs can load a little bit slow, so we beg you to be patient while reading to not miss anything, THANKS!

Top Plate

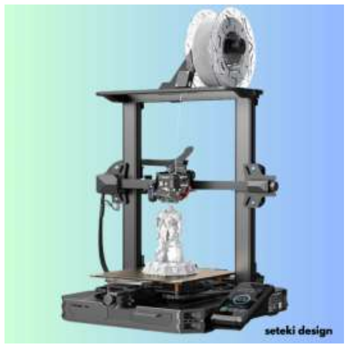
And the last part but not least the top casing which is where all the electronic components and the camera, the battery in its all the heart of our cart.



> **Note**
>
> To see each 3D model of our robot on GitHub you can click `HERE`, or go directly to the folder of `3D models` of our robot, or you can also see them graphically in the `README.md` and below each image or `GIF` of our 3D pieces open a link to see the modeling shown, yes it's again me btw :)

Equipment

In this part we are going to detail all the equipment of printers and filament with which we were able to develop the whole chassis of our cart.

	
Ender-3 V3 SE	Ender S1 PRO
Amazon link	Amazon Link

Filaments

Creality Filament PLA (Multiple Colours)	Creality Filament PETG (Multiple Colours)
Amazon	Amazon

Part 3 / Circuits and Electronics 🛰️

In the circuit part, we will describe each electrical component of the robot, a brief summary with a datasheet of each component, we will explain the diagram of the whole electrical circuit, as well as its explanation in the readme and in the folders of the folder, we will also detail where we got all the parts.

> **Tip**
>
> In the README.md you can find an cost analysys of every single component int the robot.

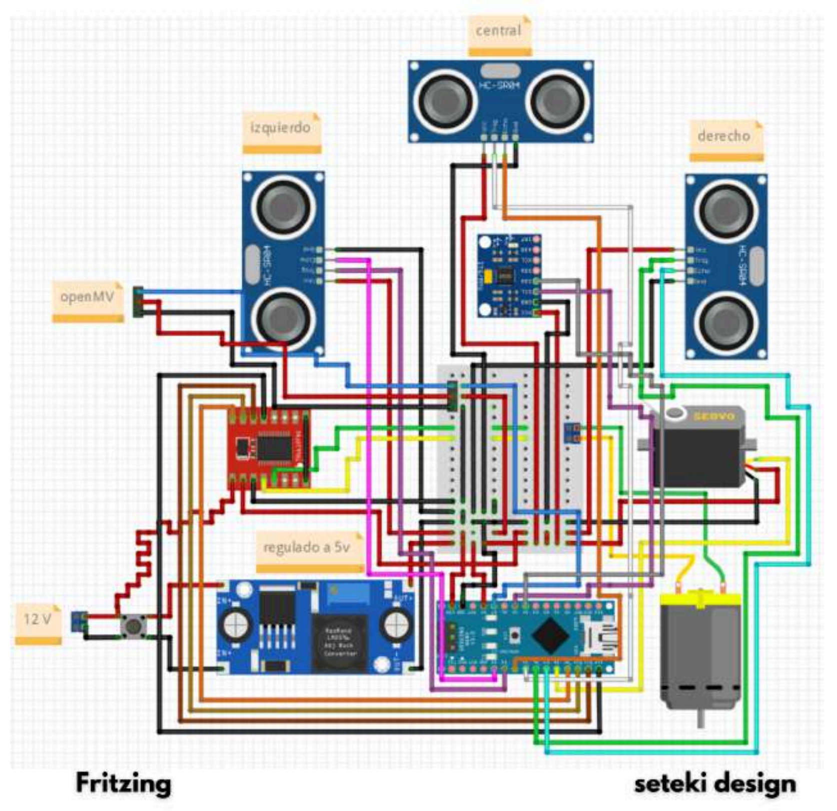
Diagrams

```
For the realisation of all the digital diagrams of our ARA 1.1 robot, we use the fritzing platform.
```

Fritzing
Download

Fritzing: Fritzing is an open-source tool that helps users design and document electronic circuits. It features a visual interface with breadboard, schematic, and PCB views. Users can create circuit diagrams, design custom PCBs, and export files for manufacturing. It's widely used in education, prototyping, and the maker community.

Seteki Diagram:



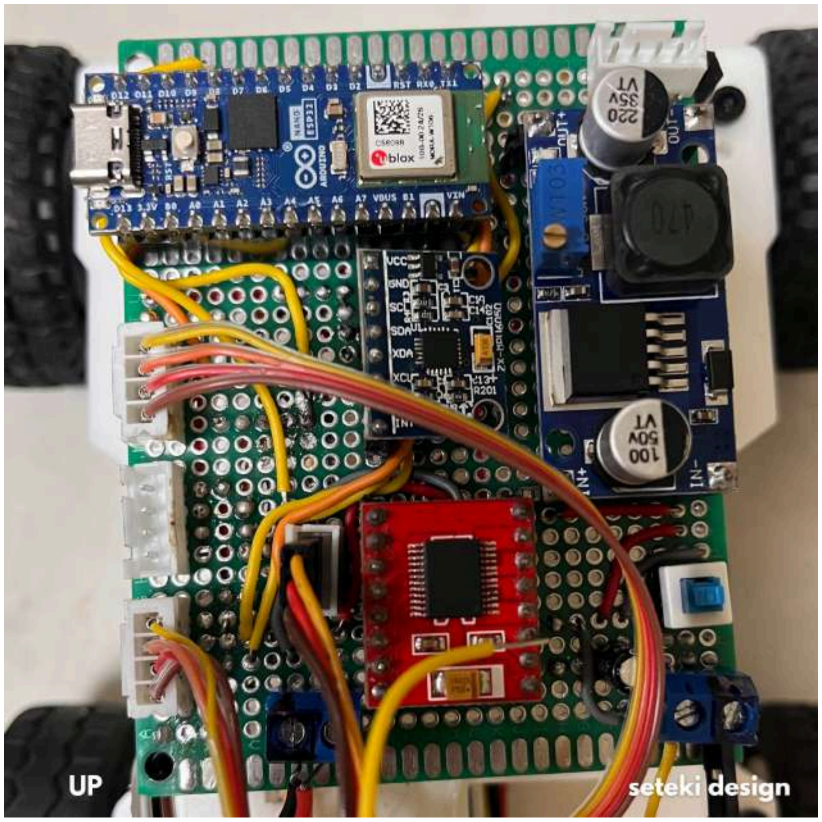
Scheme
Circuits Folder

Explanation:

- Microcontroller (Arduino Nano): Acts as the central processing unit, managing all sensor inputs and actuator outputs.
- Ultrasonic Sensors (HC-SR04): Three sensors (left, center, right) provide obstacle detection and distance measurement. Each is connected to the digital I/O pins via Trig and Echo lines, powered by +5V and GND.
- Power Supply: A 12V source is regulated to 5V using a buck converter (LM2596), ensuring safe voltage levels for the logic components. The regulated output supplies the sensors and the Arduino.
- Motor Driver: Controls a DC motor via PWM signals from the Arduino. It receives 12V for motor power and 5V logic input. The driver controls direction and speed of the rear motor.
- Servo Motor: Controlled via a PWM signal for steering purposes (Ackermann-style likely), connected to one of the digital pins.
- Gyroscope/Accelerometer (MPU-6050): Connected via I^{2}C (SDA, SCL) for IMU-based motion tracking, feeding orientation and movement data to the controller.
- OpenMV Cam: A vision system interfaced via UART and power lines. It provides object detection, color tracking, or line following. Its TX and RX are connected to the Nano for serial communication.

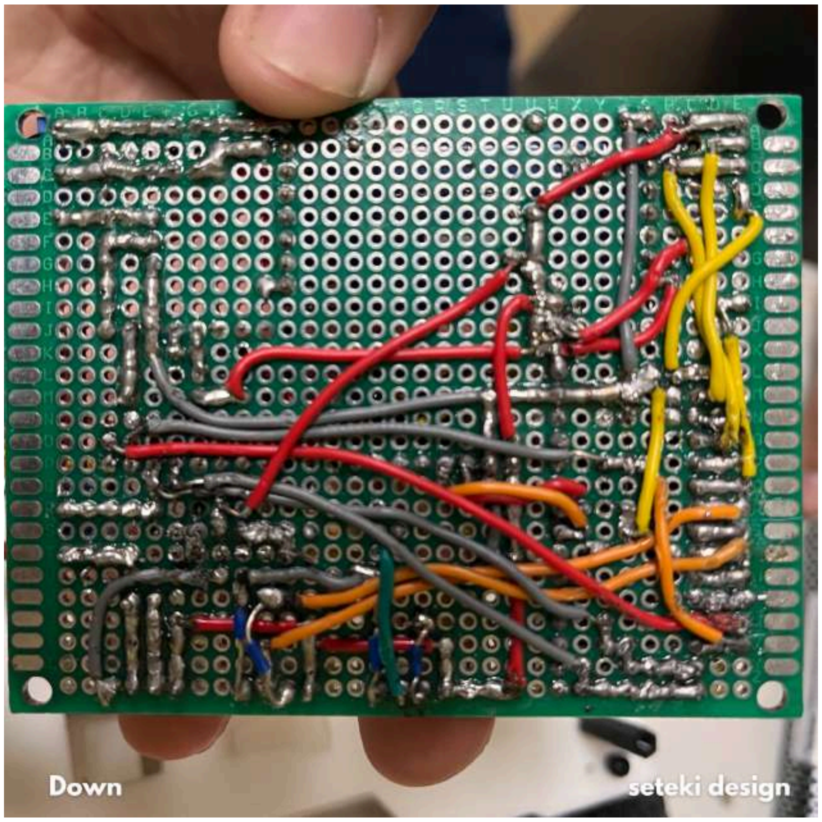
PCB Photos

```
Here will be two photos of the PCB.
```



On Top

Circuits Folder



Below

Circuits Folder

Vehicle Components

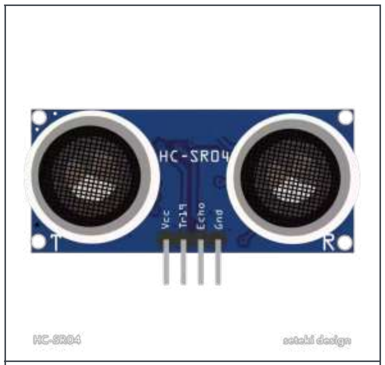
```
Every robot electrical component
```

- `Ultrasonic Sensors` [3] HC-SR04
  - Buy Here
- `Motor Driver` [1] Puente H TB6612FNG 1.2A
  - Buy Here
- `Servo motor` [1] INJORA 7KG
  - Buy Here
- `Motor` [1] Metal Gearmotor 25Dx65L mm HP 12V with 48 CPR Encoder
  - Buy Here
- `Voltage Regulator` [1] LM2596
  - Buy Here
- `Interruptor Latching` [1] Tact Switch
  - Buy Here
- `Capacitors` [2] capacitors
  - Buy Here
- `Male female conectors` [12] XH-4Y
  - Buy Here
- `Gyroscope` [1] MPU-6050 (IMU Sensor)
  - Buy Here
- `Micro-controller` [1] Arduino NANO ESP32
  - Buy Here
- `Terminal` [1] PCB Terminal
  - Buy Here
- `Prototype wires` [32] Prototype wires
  - Buy Here
- `Normal Wires` [11] Normal Wires
  - Buy Here
- `Camera` [1] Open MV H7 Plus
  - Buy Here
- `Lipo Battery` [1] Airsoft LIPPO Battery 11.1V
  - Buy Here
- `Ceramic Capacitors` [2] Ceramic Capacitors
  - Buy Here

Components explication

HC-SR04 (Ultrasonic Sensor):

The HC-SR04 ultrasonic sensor is a popular device used to measure distances using sound waves. It emits an ultrasonic pulse and calculates the time it takes for the echo to return, allowing precise distance measurements. Commonly used in robotics, obstacle detection, and automation, it provides non-contact sensing capabilities. One major advantage is its low cost and easy interfacing with microcontrollers like Arduino. It offers good accuracy within a range of 2 cm to 400 cm. Its response time is fast and reliable for real-time applications. The sensor is lightweight and energy-efficient. It's ideal for indoor environments with stable conditions. Despite being sensitive to soft or angled surfaces, its reliability makes it widely adopted in DIY and educational projects.



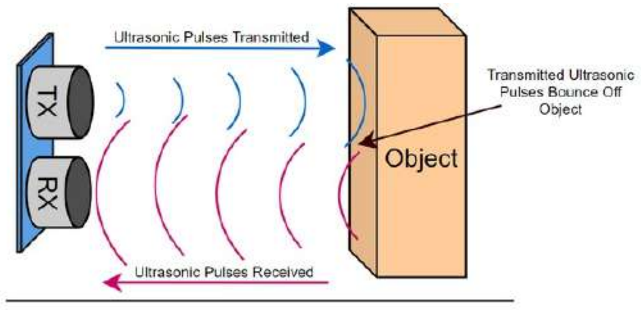
HC-SR04

HC-SR04 Conections:

- VCC → 5V on Arduino Nano
- GND → GND on Arduino Nano
- Trig → Any digital pin (e.g., D2)
- Echo → Any digital pin with interrupt support (e.g., D3)

You need to set the Trig pin HIGH for 10 microseconds to send a pulse, and then measure the time it takes for the Echo pin to go HIGH and back LOW. This time is used to calculate distance based on the speed of sound.

Mounting the two ultrasonic sensors vertically on the sides of your robot car is better because it reduces interference from the ground and improves side object detection. Vertical orientation gives a clearer field of view along the sides of the car, helping to detect walls, obstacles, or path boundaries more accurately. It also minimizes noise from reflections that occur when the sensor faces downward or too close to the floor, improving reliability in navigation.


Ultrasonic Pulses Transmitted
TX
RX
Object
Transmitted Ultrasonic Pulses Bounce Off Object
Ultrasonic Pulses Received

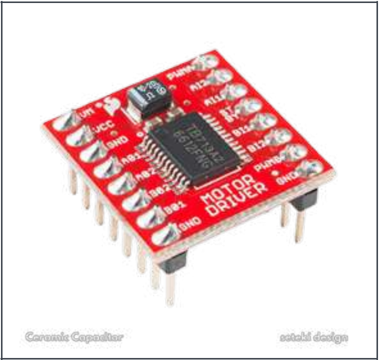
Ultrasonic Sensor: Working Principle

> **Caution**
>
> Due to GitHub problems some charts and GIFs can load a little bit slow, so we beg you to be patient while reading to not miss anything, THANKS!

> **Important**
>
> In addition to this we also added ceramic capacitors to filter the ultrasonic signals.

TB6612FNG 1.2A (Motor driver)

The TB6612FNG is a dual H-bridge motor driver designed to control two DC motors independently. It allows precise control of motor direction and speed using digital signals and PWM (Pulse Width Modulation). With a continuous current output of 1.2A per channel and a peak of 3.2A, it is suitable for small to medium-sized motors. The driver operates efficiently at voltages between 2.5V and 13.5V. It includes built-in protection features such as thermal shutdown and under-voltage lockout, enhancing safety and durability. Its compact design makes it ideal for space-constrained applications. The TB6612FNG offers better performance and energy efficiency compared to older drivers like the L298N. It generates less heat, provides faster switching, and ensures smoother motor control. This makes it a reliable choice in modern electronics and robotic systems.



Motor Driver TB6612FNG 1.2A

🟦 Power and Ground VCC → 5V on Arduino Nano (logic power)

VM → External motor power (e.g., 6V–12V battery, depending on your motors)

GND → Common ground (connect to both Arduino GND and battery GND)

🟦 Motor Outputs A01 / A02 → Connect to Motor A terminals

B01 / B02 → Connect to Motor B terminals

🟦 Control Pins PWMA → PWM pin on Arduino (e.g., D3) to control speed of Motor A

AIN1 → Digital pin (e.g., D4) to control direction

AIN2 → Digital pin (e.g., D5) to control direction

PWMB → PWM pin (e.g., D6) for Motor B speed

BIN1 → Digital pin (e.g., D7) for direction

BIN2 → Digital pin (e.g., D8) for direction

🟦 Standby STBY → Connect to 5V (or control with a digital pin like D9 to enable/disable motors)

> **Important**
>
> Remember: All grounds (Arduino, battery, and driver) must be connected together for proper operation. 💡 Use PWM capable pins for PWMA and PWMB to control speed with analogWrite().

Injora 7kg (Servo motor):

The Injora 7kg servo motor is a high-torque actuator commonly used in RC cars, robotics, and small mechanical systems. It provides up to 7 kilograms of torque, making it strong enough for steering systems and joint movement. This servo operates with precise angular control, usually within 0–180 degrees. It responds quickly and accurately to input signals, offering reliable performance in dynamic applications. Its compact size allows easy integration into small projects. The metal gear design improves durability and reduces wear over time. It runs efficiently on 4.8V–6.8V, compatible with most microcontrollers. Its strength-to-size ratio is excellent, ideal for lightweight robotic platforms. It also maintains position under load, which is key for stability. Overall, it's a solid, affordable servo with great balance between power and precision.

Injora 7kg (Servo motor)

Servo Cable Connections (usually with 3 wires): Brown or Black (GND) → GND on Arduino Nano ESP32

- Red (VCC/Power) → External 5V–6V power supply (not from the Arduino 3.3V or 5V pin)
- Orange or Yellow (Signal/PWM) → PWM-capable digital pin on Arduino Nano ESP32 (e.g., D4 or D13)

> **Important**
>
> ⚠️ Important Power Note: 💡 The Injora 7kg servo needs more current than the Arduino Nano ESP32 can safely provide. Use an external power source (like a 5V BEC or battery pack) to power the servo's VCC. Connect all GNDs together (ESP32 GND, power supply GND, and servo GND).

🟩 Wiring Summary:

- Servo Wire Connects To GND ESP32 GND
- (optionak) VCC External 5V–6V Power Signal D4 (or any PWM pin)

> **Tip**
>
> Use a capacitor (e.g., 470µF–1000µF) between VCC and GND of the servo supply to avoid voltage dips. In code, use ledcWrite() or servo libraries compatible with the ESP32's PWM channels.

Metal Gearmotor 25Dx65L mm HP 12V with 48 CPR Encoder (Motor):

The Pololu 20.4:1 Metal Gearmotor 25D×65L HP 12V with encoder is a compact, high-power DC motor ideal for robotics and automation. It delivers around 500 RPM and up to 7.4 kg·cm of stall torque, with a strong metal gearbox for durability. Its built-in 48 CPR encoder provides nearly 980 counts per gearbox revolution, allowing precise speed and position control. Operating efficiently at 12V, it supports closed-loop systems and handles moderate loads reliably.

Gearmotor 25Dx65L mm HP 12V with 48 CPR

The Pololu 20.4:1 gearmotor cannot be connected directly to the Arduino Nano because it requires more current than the board can supply. Instead, connect it to a motor driver (TB6612FNG):

- Motor terminals (M1 and M2) → connect to the output pins of the driver (e.g., A01 and A02)
- The driver control pins (IN1, IN2, PWM) → connect to Arduino Nano digital and PWM pins to control direction and speed
- VM on the driver → connect to an external power source (6V–12V, depending on your motor voltage)
- GND (driver) → connect to Arduino GND and battery GND (shared ground is essential).

LM2596 (Voltage Regulator):

The LM2596 is a step-down (buck) voltage regulator that efficiently converts higher voltages (up to 40V) down to a stable, lower voltage like 5V or 3.3V. It works using high-frequency switching and a feedback loop to maintain output voltage. Unlike linear regulators, it minimizes heat loss and energy waste. It can handle up to 2A of continuous current, making it suitable for powering microcontrollers and motors.

LM2596

LM2596 Pin/Label

- IN+ : Positive input voltage (e.g., 7V–36V from battery or adapter)
- IN– : GND of input power source
- OUT+ : 5V input pin on Arduino Nano (Vin or 5V, depending on output)
- OUT– GND pin on Arduino Nano

MPU-6050 (IMU Sensor):

The MPU-6050 is a 6-axis motion tracking sensor that combines a 3-axis accelerometer and a 3-axis gyroscope in a single compact chip. It measures both linear acceleration and rotational velocity, allowing it to detect orientation, movement, and tilt. The sensor communicates via I2C, making it easy to connect to microcontrollers like the Arduino. It features built-in digital motion processing (DMP) that helps offload complex calculations from the main processor. With high sensitivity and low noise, it provides accurate and stable readings.

MPU-6050 (Gyroscope)

- MPU-6050 Pin Arduino Nano Pin
- VCC: 5V
- GND: GND
- SCL: A5
- SDA: A4

- INT: (Optional) D2 or any digital pin (for interrupts)

> **Important**
>
> ⚠️ Important Power Note: 💡 The MPU-6050 works with both 3.3V and 5V, but always check your module version. Most breakout boards Include a voltage regulator, so connecting VCC to 5V is safe. SCL (Serial Clock Line) and SDA (Serial Data Line) are for I2C communication. You can optionally connect INT to a digital pin (e.g., D2) to use interrupt features like motion detection.

Arduino NANO ESP32:

The Arduino Nano ESP32 is a small and modern microcontroller that makes it easy to create smart and connected projects. It has built-in Wi-Fi and Bluetooth, so it can communicate wirelessly with other devices. Its compact size makes it perfect for small spaces and portable gadgets. You can program it using the Arduino platform, which is simple and beginner-friendly.

Arduino NANO ESP32

The Arduino Nano ESP32 features the Espressif ESP32-S3 microcontroller with a dual-core Xtensa LX7 processor running at 240 MHz. It includes 512 KB SRAM, 384 KB ROM, and 8 MB external flash. The board supports 2.4 GHz Wi-Fi (IEEE 802.11 b/g/n) and Bluetooth 5 (LE) connectivity. It operates at 3.3 V logic level and includes 14 digital I/O pins, 8 analog inputs, and multiple PWM channels. Communication interfaces include I2C, SPI, UART, USB 2.0 (via USB-C), and JTAG. The onboard voltage regulator supports input voltages from 5 V to 21 V via VIN. It is programmable via the Arduino IDE or MicroPython and supports native USB functionality.

Open MV H7 Plus (CAMERA):

The OpenMV H7 is a powerful embedded vision camera designed for real-time image processing and machine vision tasks. It works by capturing video and analyzing it directly on-board using a high-speed microcontroller, so there's no need for a separate computer. It can recognize objects, track faces, detect colors, read QR codes, and even run machine learning models like neural networks. The camera is programmable in Python, making it easy and accessible for developers and students. Its compact size and built-in lens make it ideal for robotics, automation, and AI applications. What makes it especially good is its ability to process images in real time with low power consumption. It bridges the gap between camera modules and full computer vision systems, offering a smart, efficient, and versatile solution for embedded vision.

Open MV H7 Plus

The OpenMV H7 is powered by an STM32H7 Cortex-M7 processor running at 480 MHz, with a double-precision FPU and DSP instructions for high-performance image processing. It includes 1 MB of SRAM and 2 MB of internal flash memory, with support for external microSD cards. The board features multiple 5V-tolerant I/O pins capable of UART, SPI, I²C, PWM, ADC, DAC, and CAN functions. It typically uses an OV7725 (640×480) or OV5640 (5MP) camera sensor, supporting up to 150 FPS at low resolutions. Image processing tasks like face tracking, QR code reading, and neural network inference run directly on board using MicroPython. It includes USB-C for programming and communication, servo headers, RGB/IR LED support, and LiPo battery input. Power consumption ranges between 110–170 mA at 3.3 V, with onboard voltage regulation. Its compact form factor (around 45×36×30 mm) makes it suitable for embedded vision systems.

LIPPO Battery (CAMERA):

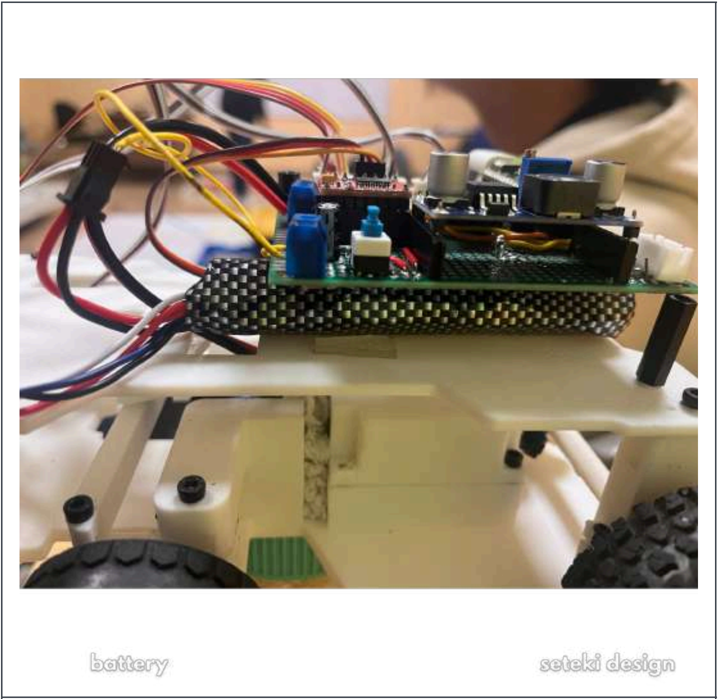
LiPo 11.1V batteries (Lithium Polymer) are rechargeable batteries made up of three cells in series, each providing 3.7V nominal voltage (3.7 V × 3 = 11.1 V). They work by moving lithium ions between positive and negative electrodes during charging and discharging. When discharging, lithium ions move from the anode to the cathode, creating an electric current that powers devices. When charging, the process is reversed. LiPo batteries are lightweight and deliver high current output, making them ideal for drones, RC vehicles, and robotics.

Battery LIPPO

Part 4 / Power - Mobility management

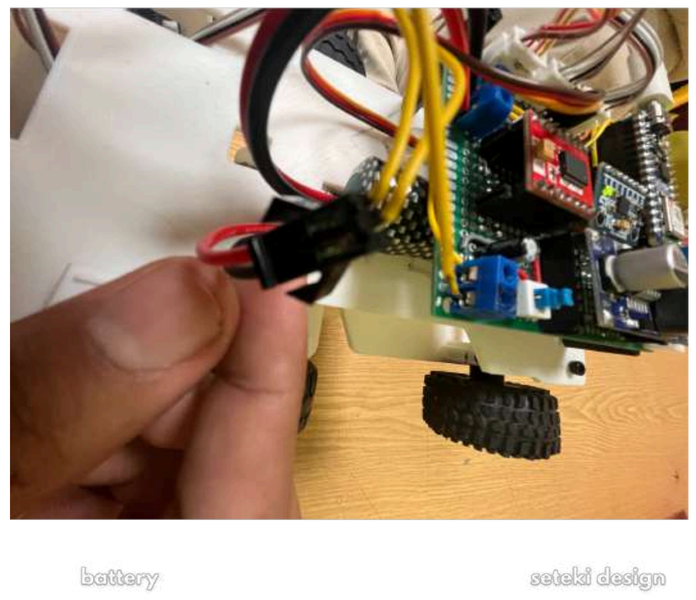
On the power side of the cart this can be seen graphically in the diagram.

The battery feeds the whole circuit with 12.1 V but clearly this would burn all our robot xd, so for that we have the LM2596 voltage regulator, in which 5V come out of this for the arduino and all the sensors, but a part of the robot voltage takes 12V to feed the motor, through a driver, so 12V of all the power supply of the robot goes to the driver.



LIPPO

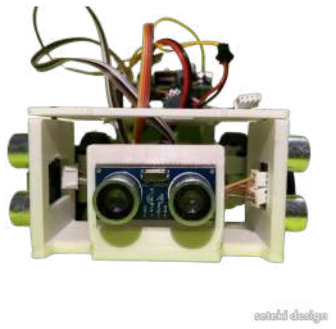
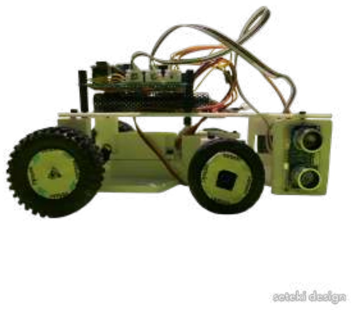
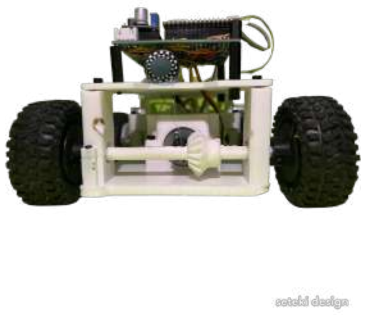
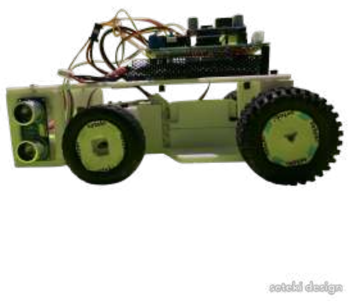
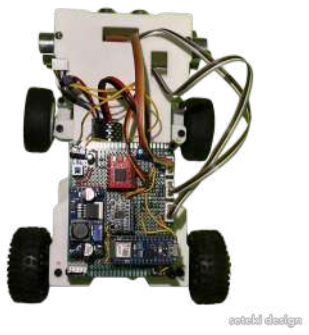
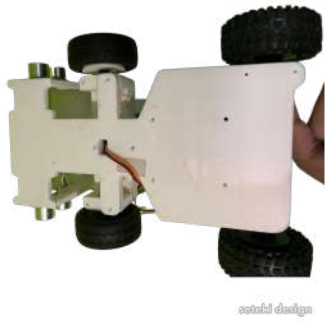
Diagram


Connection
Diagram

Part 5 / Robot Assembly

Here will be all the robot photos same as the carpet.

Every robot photo

Front	Right	Back
		
Left	Top	Bottom
		

Part 6 / Code

```
const float TURN_ANGLE_TOLERANCE = 2.0;      // Toler
const unsigned long tiempoEstabilizacion = 1000; // D
const unsigned long tiempoCorreccion = 1600;    // Dur
const unsigned long TIMEOUT_CORRECCION = 3700; // Tim
```

```
T|35.712 -> St:NORMAL|RYaw:20.0|SetP:0.0|TErr:-20.0|DT:1
[illegible]
```

Here will be all the codes of the robot and the platforms we use for each round.

Platforms

ARDUINO IDE	VISUAL STUDIO CODE
OPEN MV	

Motor Code / Driver:

Outputs

```
const int pwma = 9;  // PWM pin for speed control
const int ain2 = 10; // Motor direction pin AIN2
const int ain1 = 11; // Motor direction pin AIN1
const int stby = 12; // Standby pin (if applicable)
```

These pins interface with an H-bridge motor driver, we do it with the `TB6612FNG` :

- pwma controls motor speed via PWM (Pulse Width Modulation).
- ain1 and ain2 determine the rotation direction:
- AIN1 = HIGH and AIN2 = LOW → Forward
- AIN1 = LOW and AIN2 = HIGH → Reverse
- Both LOW → Stop

Forward Movement:

```
void moverAdelante(int velocidad) {
    digitalWrite(ain2, LOW);
    digitalWrite(ain1, HIGH);
    analogWrite(pwma, constrain(velocidad, 0, 255));
}
```

- Sets direction pins for forward movement.
- analogWrite(pwma, velocidad) modulates motor power.
- constrain() ensures the speed remains within [0, 255].

Reverse Movement:

```
void moverAtras(int velocidad) {
    digitalWrite(ain1, LOW);
    digitalWrite(ain2, HIGH);
    analogWrite(pwma, constrain(velocidad, 0, 255));
}
```

```
 Inverts the direction pins to drive the motor backwards.
```

Motor stop:

```
void detenerMotor() {
    digitalWrite(ain1, LOW);
    digitalWrite(ain2, LOW);
    analogWrite(pwma, 0);
}
```

- Both control pins set to LOW to disable the H-bridge path.
- PWM set to zero ensures no motion.

Speed Profiles and Motion States:

```
const int velocidadNormal = 255;
const int velocidadGiro = 255;
const int velocidadEstabilizacion = 200;
const int velocidadCorreccionMin = 225;
const int velocidadCorreccionMax = 255;
```

These values determine PWM duty cycles in various robot states:

```
 - velocidadNormal: cruising speed in NORMAL state.

 - velocidadGiro: used during turning (EN_MANIOBRA).

 - velocidadEstabilizacion: reduced speed post-turn.

 - velocidadCorreccionMin/Max: used during CORRIGIENDO for ramping velocity.
```

Servo Code

Configuration:

```
const int servoPin = 8;
Servo steeringServo;
```

- The servo motor is connected to digital pin 8 on the microcontroller.
- Servo is an instance of the ESP32Servo library (compatible with ESP32 architecture).
- It allows PWM signal generation to control angle-based rotation (typically 0° to 180°) via hardware timer abstraction.

Servo Angle Constraints and Calibration:

```
const int servoNeutral = 90;      // Angle to go straight
const int servoMinAngle = 60;     // Minimum limit to avoid damage
const int servoMaxAngle = 180;    // Maximum limit to avoid damage
```

- `servoNeutral` defines the center (straight) orientation of the steering — critical for path alignment.
- `servoMinAngle` and servoMaxAngle serve as safety constraints to prevent mechanical overtravel or hardware strain.
- These bounds are enforced when sending `servo angles` in PID control logic

PID-Based Servo Control:

```
float setpoint = 0.0;          // Desired heading
float error = 0.0, prevError = 0.0;
float pidOutput = 0.0;         // Will be mapped to steering angle
```

- The error between current yaw and desired heading is calculated.
- A PID controller generates an output (`pidOutput`) proportional to that error.
- This output is mapped to a servo angle, like:

```
int servoAngle = constrain(servoNeutral + pidOutput, servoMinAngle, servoMaxAngle);
steeringServo.write(servoAngle);
```

- This allows real-time adjustments to robot steering using closed-loop feedback from IMU yaw data.

Ultrasonics Code

Confguration:

```
#define TRIG_IZQUIERDO 3
#define ECHO_IZQUIERDO 2
#define TRIG_CENTRO 5
#define ECHO_CENTRO 4
#define TRIG_DERECHA 6
#define ECHO_DERECHA 7
```

Each ultrasonic sensor uses two pins: one for triggering a sound pulse (TRIG) and one for receiving the echo (ECHO).

The robot has three sensors:

- Left (IZQUIERDO)
- Center (CENTRO)
- Right (DERECHA)

These are used to sense obstacles, corridor walls, or available openings on each side.

Reading Function:

```
float medirDistancia(int trigPin, int echoPin) {
    digitalWrite(trigPin, LOW); delayMicroseconds(2);
    digitalWrite(trigPin, HIGH); delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    long duration = pulseIn(echoPin, HIGH, TIMEOUT_ULTRASONICO);
    if (duration == 0) return 999.9;
    float dist = (duration * 0.0343) / 2.0;
    if (dist < DISTANCIA_MIN_VALIDA || dist > DISTANCIA_MAX_VALIDA) return 999.9;
    return dist;
}
```

Filtering: Median Filter (Noise Reduction):

```
#define FILTER_SIZE 3
float lecturasIzquierdo[FILTER_SIZE];
float lecturasCentro[FILTER_SIZE];
float lecturasDerecho[FILTER_SIZE];
```

Each sensor uses a sliding window filter of size 3.

The filtering process involves:

- Storing new readings in an array.
- Using qsort() to sort the window.
- Rejecting invalid values (999.9).
- Returning the median valid value.

```
float calcularMediana(float arr[], int n);
```

- This eliminates transient spikes and false echoes.

First Challenge:

Explanation

IMU Sensor, configuration:

```
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
```

- The `MPU6050` is interfaced using the Adafruit unified sensor driver, abstracting raw register-level communication.
- The mpu object is an instance of `Adafruit_MPU6050` , which allows access to accelerometer and gyroscope data.

Caution

Due to GitHub problems some charts and GIFs can load a little bit slow, so we beg you to be patient while reading to not miss anything, THANKS!'

Gyroscope Offset Calibration:

```
void calibrarGiroscopio() {
    float sum = 0.0;
    Serial.println("Calibrando giroscopio...");
    delay(1500);
    for (int i = 0; i < numCalibSamples; i++) {
        sensors_event_t a, g, temp;
        mpu.getEvent(&a, &g, &temp);
        sum += g.gyro.z;
        delay(5);
    }
    gyroOffset = sum / numCalibSamples;
    Serial.print("Offset Giroscopio (Z rad/s): ");
    Serial.println(gyroOffset, 6);
}
```

- Purpose: Calculates a bias term for the Z-axis gyroscope by averaging static readings.
- `gyroOffset` represents the baseline drift due to sensor noise, temperature, or manufacturing variance.
- Result is stored in radians per second (rad/s).
- Critical to ensure accurate yaw integration in motion states.

Gyroscope Readings:

```
sensors_event_t a, g, temp;
mpu.getEvent(&a, &g, &temp);
```

This provides:

- `g.gyro.z` → Angular velocity about Z-axis (yaw rate)
- Output is in rad/s, which is converted to deg/s for most internal calculations.

Yaw Angle Estimation via Gyroscope Filter:

```
void calcularKalmanYaw(float gyroZ_rad, float dt) {
    float gyroRateCorrected_rad = gyroZ_rad - gyroOffset;
    float rate_deg = gyroRateCorrected_rad * 180.0f / PI - kalmanBias;
    kalmanYaw += rate_deg * dt;
    // ... Filter update math omitted ...
}
```

Logic / Explanation

The robot logic for the WRO Future Engineers first round is structured as a robust `finite state machine` (FSM) governing all motion and behavior through distinct states: `NORMAL` , `EN_MANIOBRA` , `ESTABILIZANDO` , `CORRIGIENDO` , `PARKING` , `ESQUIVAR_BLOQUES` , and DETENIDO. The system begins in the NORMAL state, driving forward using PWM signals `(analogWrite(pwma, velocidadNormal))` while continuously monitoring three ultrasonic sensors on the left, center, and right. These sensors provide noisy distance data, which is filtered using a median filter (e.g., `calcularMediana(` lecturasCentro, 3)) to improve measurement reliability. The robot uses these filtered distances to detect obstacles or openings; for instance, if the left distance suddenly increases compared to distancialzquierdoInicial multiplied by umbralRatioLateral, the robot interprets it as an available turn and enters the EN_MANIOBRA state.

In EN_MANIOBRA, the robot rotates while tracking its angular orientation using yaw estimation derived from gyroscope integration, with bias correction from a calibration phase (calibrarGiroscopio() computes gyroOffset). This angle is used to reach a precise turning goal, like +90° or -90° relative to a setpoint. Steering is executed via a servo motor on pin 8, controlled with steeringServo.write(), where PID logic maps angular error to a steering angle, ensuring controlled curvature. After completing a turn (checked via error threshold like TURN_ANGLE_TOLERANCE), the system transitions into `ESTABILIZANDO` , pausing or reducing speed (velocidadEstabilizacion = 200) to allow mechanical settling. Then it moves to CORRIGIENDO, where heading errors are corrected using a second PID controller. If the wall is too close laterally (e.g., less than 7.0 cm), a PD controller triggers lateral corrections toward a DISTANCIA_LATERAL_OBJETIVO_PD of 10.0 cm.

Once a fixed number of turns is executed (`NUM_CURVAS_ANTES_DE_PARAR` = 4), the system switches to PARKING. This complex state begins with lateral alignment, using a dedicated PD loop (Kp_parking_lateral, Kd_parking_lateral) to match the robot's side distance to the target tolerance. Once aligned, it transitions to frontal alignment based on the center ultrasonic sensor, advancing or reversing with full PWM (`VELOCIDAD_PARKING_AVANCE` = 255). Transitions between sub-states are managed by timeouts like `TIMEOUT_PARKING_SUBFASE` = 7000 and alternating between lateral and yaw correction when alignment fails. The system's heading tracking relies entirely on filtered gyroscope output derived from integrating the Z-axis angular velocity, corrected with `gyroOffset` to prevent drift. This heading is essential for PID-based navigation and precise maneuvering.

Second Challenge:

> **Important**
>
> Though named "Gyroscope Filter", this structure essentially implements a gyroscope-based yaw integrator with bias correction. :)

IMU Sensor calibration:

```
void calibrarGiroscopio() {
    float sum = 0.0;
    for (int i = 0; i < 2000; i++) {
        sensors_event_t a, g, temp;
        mpu.getEvent(&a, &g, &temp);
        sum += g.gyro.z;
        delay(5);
    }
    gyroOffset = sum / 2000.0;
}
```

PID Heading control:

```
float errorYaw = targetYaw - yaw;
float output = Kp * errorYaw + Kd * (errorYaw - lastError) / dt;
steeringServo.write(constrain(90 + output, servoMin, servoMax));
```

- The estimated yaw is compared to a `targetYaw` to compute angular error.
- This error feeds a PD controller (only proportional and derivative terms used).
- The controller output determines the new servo position, adjusting the robot's steering to reduce heading error.
- Example: If the robot must turn 90° right, `targetYaw` is updated to yaw + 90, and the PID loop continuously adjusts the servo until that angle is reached.

Camera Code

```
Open mv h7 plus with open mv software
```

Configuration

```
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.set_auto_gain(False)
sensor.set_auto_whitebal(False)
```

- The camera sensor is initialized with RGB565 pixel format and QVGA resolution (320x240).
- Auto-gain and auto-whitebalance are disabled to ensure stable and consistent color tracking, as these automatic adjustments can interfere with blob detection in color space.

LAB Color Thresholds:

```
threshold_color_1 = [0, 100, 4, 16, 6, 17]
threshold_color_2 = [0, 100, -38, -31, 31, 40]
```

- These arrays define manual LAB color thresholds for each target color (red and green in this case).
- The script supports multi-color tracking by iterating over a list of thresholds.
- LAB space is used instead of RGB because it provides better separation between brightness and chromaticity, making color segmentation more robust.

Image Grid Section Mapping:

```
def get_section_from_coords(cx, cy):
    ...
```

The 320x240 image is divided into 6 logical sections (2 rows × 3 columns). Each detected blob's center coordinates (cx, cy) are mapped to one of these numbered sections (1 through 6 on the top color, 7 through 12 on the second color). This simplifies positional awareness and allows decision-making (like evasive maneuvers) in the ESP32 based on blob location.

Distance Estimation via Pixel Position:

- The vertical position (y) of the bottom of the blob is used to estimate distance to the object using a second-degree polynomial model.
- This model is generated from empirical calibration using a separate script (ajuste.py), mapping pixel positions to real-world distances.

  Only the Y-axis position of the blob is required—no stereo vision or depth camera is used.

```
#### Serial Communication Over UART:
uart = pyb.UART(3, 115200, timeout_char=1000)
...
mensaje = f"{distancia_estimada:.2f},{final_section_id},{centro_x}\r\n"
uart.write(mensaje)
```

Communication to an external controller (e.g., ESP32) is handled via UART3 at 115200 baud using pyb.UART.

Each frame, the script sends:

- distance_estimated (in cm),
- section ID (adjusted by color),
- center X coordinate of the blob.

Format: "{distance},{section},{centerX}\r\n" — matching the expected input format of the ESP32.

If no blobs are detected in a frame, the script sends "0,0,0\r\n" to indicate absence of valid targets.

Explanation / Logic

The second-round code implements an advanced autonomous navigation strategy using a finite state machine (FSM) with the robot reacting dynamically to its environment through IMU orientation, ultrasonic sensors, and camera input from OpenMV.

The robot begins in the NORMAL state, driving forward with PID-based yaw correction using gyroscopic feedback from the MPU6050 (float error = setpoint - kalmanYaw;). The robot constantly monitors ultrasonic sensors to detect wall proximity and uses median filtering (calcularMediana()) to stabilize readings. These filtered distances inform whether the robot should trigger turns or corrections.

Upon detecting a narrowing corridor (distanciaCentroFiltrada < umbralRatioCentral * distanciaCentralInicial), the robot transitions to the EN_MANIOBRA state. It executes a turn using gyroscope-driven angular tracking until reaching a predefined yaw change (fabs(errorAngular) < TURN_ANGLE_TOLERANCE).

After the turn, it enters ESTABILIZANDO, where it pauses or moves slowly (moverAdelante(velocidadEstabilizacion)) to allow mechanical oscillations to settle. Then, it transitions into CORRIGIENDO, where it performs lateral or angular adjustments using PD or PID based on detected wall distances (DISTANCIA_LATERAL_MIN_ACTIVACION_PD triggers this).

When block detection is necessary (via OpenMV), the robot enters ESQUIVAR_BLOQUES. It uses PD control to align the X-coordinate of the detected blob (errorX_esquivar = SETPOINT - centerX) and make decisions such as backing up or sidestepping based on the section ID sent over UART (if(final_section_id == 5)).

If the robot identifies a designated end-of-course configuration, it switches to the PARKING state. There, it executes a two-stage alignment: first laterally aligning to a wall using PD (errorParkLat = distanciaIzquierdo - objetivo), then advancing or reversing until the central ultrasonic sensor confirms proper frontal alignment. Sub-states (SUB_PARKING_*) track progress, and timeouts (TIMEOUT_PARKING_SUBFASE) prevent deadlock.

Throughout, the robot counts maneuvers (contadorCurvas) and halts in DETENIDO after completing all required actions (NUM_CURVAS_ANTES_DE_PARAR).

State transitions depend heavily on time, distance thresholds, and angular deviation, ensuring robust performance in complex maze-like or obstacle-filled environments.

Video

Important

here you will find the video of the robot doing the challenges (HERE). :)

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

● C++ 91.2% ● Python 8.8%