Los Perdidos

Period 2

NHL Stenden University of Applied Sciences

15.01.2024

Name	Studentnumber
Lennon tijmes	5364612
Diogo Alves	5548772
Peter Möller	5361702
Fabiana Brando	5500907
Alexia Mîţ	5587832
Gonçalo Pego	5503981
Matin Shiran	5362962

Foreword

Contents

Technical Design

Introduction

We are “Los Perdidos”, a company that specializes in the production of commercial hardware. We were requested to make three different robot designs each with unique objectives for our clients who are looking forward to mass producing our product so it can be purchased for commercial and educational uses. Our first model produced by Alexia and Fabiana is called “King Julian” it is designed to follow a line and evade obstacles with agility. Our second model developed by Diogo, Matin and Goncalo “MotoMoto” is dedicated to solving a line maze with precision. And our last model created by Lennon and Peter is named “Mort” and it is in charge of finishing a physical maze with efficiency. After some trial and error all our models have been perfected to accomplish their tasks as swiftly as possible and are now ready to be presented to the customers.

You can learn more about our team and our products on our website:
https://los-perdidos.onrender.com/

And if you want to see our products in action you can watch our youtube video:
https://www.youtube.com/watch?v=WdPbkgiAgjo&ab_channel=alexiaa227_

Version history

Version	Date	Status	Description
0.1	06/12/2024	Draft	General information
0.2	14/1/2025	Draft	Added more structure, rewritten old data, added test-results.
1.0	15/1/2025	Finished	Added advisory report, Team-Website, TeamCode, cleaned up visuals

The clients

Our clients are two well-known scientists, Bart Oerlemans and Marcel Baron. Together, they've been leading the Worldwide organization named Bart Inc. Robots since 2018.

Their story starts the first time Marcel found out about "Lego League". He was so impressed, that he immediately ran and showed Bart the discovery he made: "LEGO® League guides youth through STEM (Science, Technology, Engineering, Mathematics) learning and exploration at an early age. From Discover, to Explore, and then to Challenge, students will understand the basics of STEM and apply their skills in an exciting competition while building habits of learning, confidence, and teamwork skills along the way. We should try and do the same!".

The factory is based in Emmen, region Drenthe, The Netherlands and their main goal is to sell the fastest and most reliable Relay-bots in Europe.

Standards and guidelines

Safety First: All components must run within safe voltage and current limits. Sensors and moving parts should operate without overheating or causing damage.

Flexibility: Robots are designed to allow easy upgrades or part replacements. The software is structured for future improvements.

Reliable Performance: Each bot should complete its tasks accurately and smoothly, with sensors and motors tuned for quick adjustments in real-time.

Clear Communication: Sensor data should be sent to the dashboard in real-time and shown visually for easy tracking.

User-Friendly Design: Neopixels will show what the bot is doing with clear, color-coded signals.

Thorough Testing: All bots will be tested repeatedly in their environments to ensure they perform consistently.

These guidelines keep our relay-bots efficient, adaptable, and ready for action.

Description (current/wanted) system

The proposed Maze Navigation System will consist of three autonomous robots. These robots are intended for deployment in distinct maze environments to demonstrate autonomous navigation and obstacle handling capabilities.

Each robot will independently operate within its assigned maze type:

1- Line Maze with Obstacles: This environment challenges the robot to detect and navigate around obstacles while maintaining alignment with the maze's pathway.

2- Line Maze with Dead Ends: This maze requires the robot to identify and backtrack from dead ends, using logic to find the optimal route to the goal.

3- Physical Maze with Walls: This scenario involves a 3D layout where the robot must detect and avoid walls, effectively utilizing spatial awareness and decision-making algorithms.

The robots will rely on direct input from their embedded sensors and controllers for navigation and system operations. Outputs, such as movement logs or navigation statistics, will be generated directly by the system for evaluation and analysis. To ensure adaptability, the system design incorporates modular input/output mechanisms to accommodate future enhancements. Initial development and configuration will be carried out by the technical team, with iterative testing and refinement to optimize performance in each maze type.

Architecture

Battle Bot King Julian: Line Follower

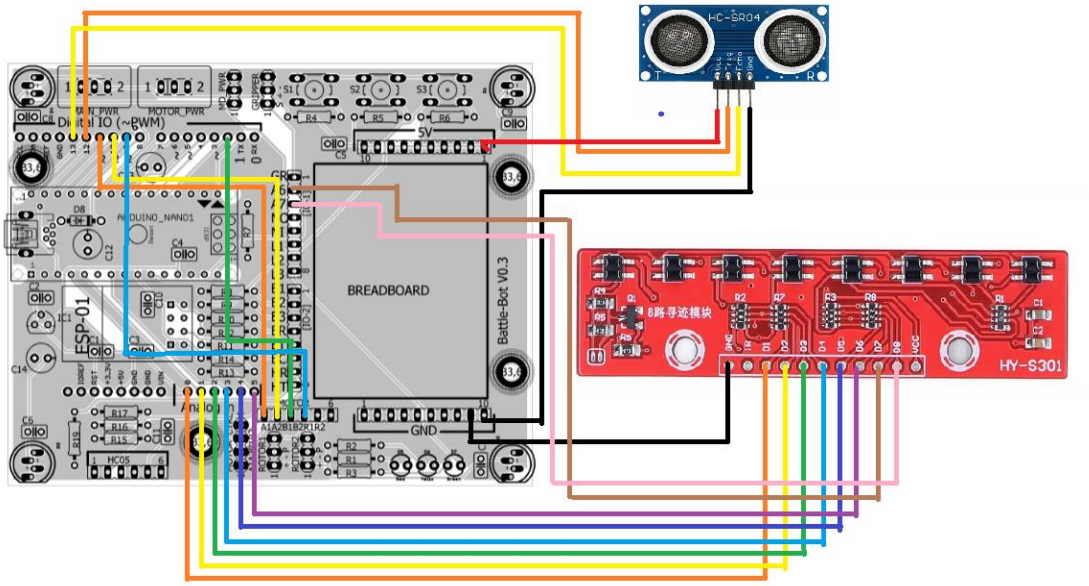
Component	Pin	Type	I/O
Neopixel	0	Digital	O
Motor	11, 9, 10, 3	-	-
Left Motor Forwards	10	Digital	I
Left Motor Backwards	11	Digital	I
Right Motor Forwards	9	Digital	I
Right Motor Backwards	3	Digital	I
Trigger of ultrasonic sensor (HC-SR04)	12	Digital	I
Echo of ultrasonic sensor (HC-SR04)	13	-	O
Gripper (Servo motor SG90)	-	Digital	O
Reflection Sensor Array QTR-8RC	A0 to A7	Analog	I

Component	Pin	Type	I/O
Neopixel	0	Digital	O
Motor	9, 5, 6, 10	-	-
Left Motor Forwards	9	Digital	I
Left Motor Backwards	5	Digital	I
Right Motor Forwards	6	Digital	I
Right Motor Backwards	10	Digital	I
Trigger of ultrasonic sensor (HC-SR04)	12	Digital	I
Echo of ultrasonic sensor (HC-SR04)	11	-	O
Gripper (Servo motor SG90)	-	Digital	O
Reflection Sensor Array QTR-8RC	A0 to A7	Analog	I

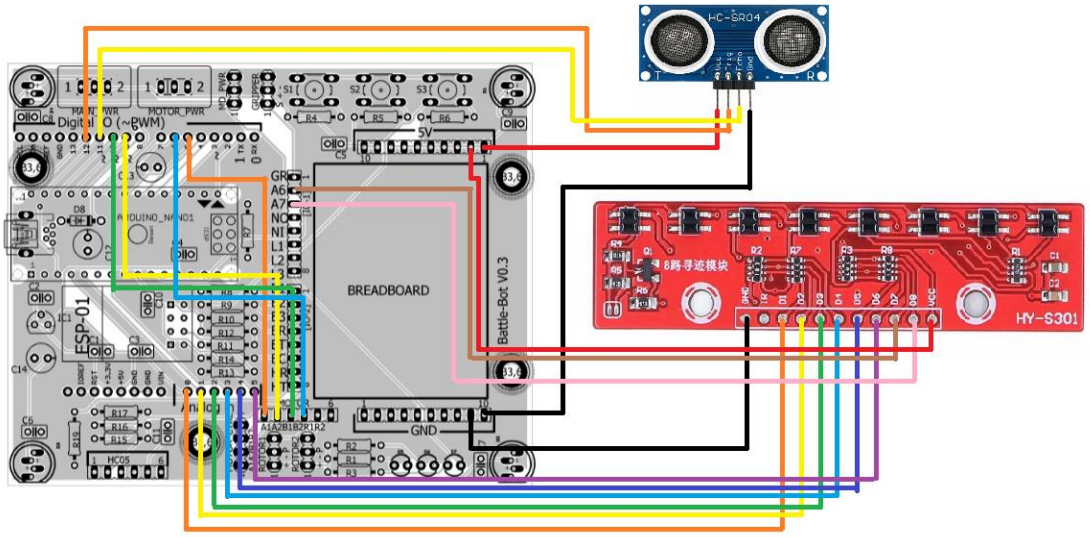
Component	Pin	Type	I/O
Neopixel	11	Analog	O
Left Motor Forwards	9	Analog	O
Left Motor Backwards	7	Digital	O
Right Motor Forwards	10	Analog	O
Right Motor Backwards	8	Digital	O
Trigger of ultrasonic sensor (HC-SR04)	12	Digital	O
Echo of ultrasonic sensor (HC-SR04)	13	Digital	I
Gripper (Servo motor SG90)	4	Analog	O
Reflection Sensor Array QTR-8RC	A2 – A7	Analog	I
Rotation Sensors (MH-Sensor-Series) Right	3	Interrupt	I
Rotation Sensors (MH-Sensor-Series) Left	2	Interrupt	I
Trigger of Right Ultrasonic Sensor (HC-SR04)	5	Digital	O
Echo of Right ultrasonic sensor (HC-SR04)	6	Digital	I
Trigger of Left Ultrasonic Sensor (HC-SR04)	A1	Digital	O
Echo of Left ultrasonic sensor (HC-SR04)	A0	Digital	I

Architecture overview (visual)

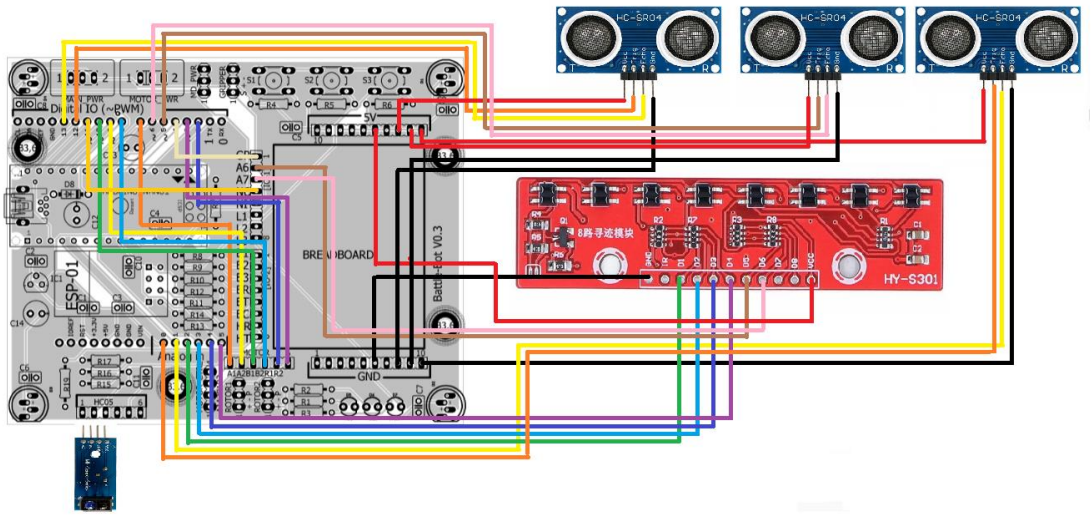
Battle Bot King Julian: Line Follower



Battle Bot MotoMoto: Line Maze



Battle Bot Mort: Physical Maze



Hardware

This part describes the hardware that is used in the relay-bot. As there are many parts it will be shortly explained what it is and some extra information about its use.

Arduino Nano microcontroller

The Arduino Nano acts as the central brain, responsible for executing the commands given to it. The microcontroller can interpret the Arduino programming language and translate it into actions. This allows it to control motors, make LED's blink, and much more. The PCB (Printed Circuit Board) connects sensors and actuators through pins and links the microcontroller to a computer for uploading code.

- Microcontroller: ATmega328
- Architecture: 8-bit AVR
- Interfaces: Mini USB, ISP, I^2C, SPI™, Serial, ICSP
- Operating voltage: 5V
- Flash memory: 32KB (2 KB used by bootloader)
- SRAM: 2KB
- Clock speed: 16MHz
- Analog IN pins: 8
- EEPROM: 1KB
- DC current per I/O pin: 20mA
- Input Voltage: 7 – 12 V
- Input Voltage (limits): 6 - 20V
- Digital I/O pins: 22 (6 PWM)
- PWM output: 6
- Weight: 7g
- Power consumption: 19mA

Powerbank

A 10.000mAh powerbank powers the robot.

On/Off Switches

The robot has two on/off switches:

- One controls the motors.
- The other controls the sensors.

SG90 Servo Motor with Grippers

The SG90 servo motor is used to control the grippers. It is a lightweight motor capable of rotating 90° in both directions. Which can be used to pick up objects.

- Torque: 1,6kg·cm
- Voltage: 4,8V – 7,2V
- Speed: 0,12 sec/60° (at 4,8V)

HC-SR04 Ultrasonic Sensor

The robot is equipped with an ultrasonic sensor (HC-SR04), which converts an electrical signa into l a 40kHz ultrasonic pulse (inaudible to humans). When the pulse is reflected back to the receiver, the sensor calculates the distance to the obstacle.

- Operating voltage: 5V DC
- Operating current: 15mA
- Measurement angle: 15°
- Range: 2cm – 4m

HC-SR04 Ultrasonic Sensor - Testing

Ultrasonic Sensor Testing The sensor was tested in a few ways: accuracy, angle and input signal. The sensor's output was converted to cm and then compared to an object placed a measured distance from the sensor. Both measurements matched. Distances below 2cm and 4m did not return a reading. The sensor also was tested against a non-perpendicular surface and at around 30° the sonar stopped detecting the surface. We tried to send multiple signals before first one returned - the sensor just returned 0. You have to send one TRIG signal at a time and wait for output. It also did not work when sending a TRIG pulse longer than 100us.

QYF-750 Line Sensor

The robot includes a QYF-750 line-sensor, which has 8 pins. This sensor works by emitting light and capturing the reflected light, enabling it to detect a line.

- The sensor uses phototransistors to reflect light onto the surface.
- It distinguishes between reflective areas (light) and non-reflective areas (dark).

- The data is transmitted to the microcontroller via the 8 digital outputs.

Line Sensor Testing

The values and functionality of the line sensor needed to be brought to attention. With a small test we can determine what the line sensor could give as data.
We first place the Robot on a white surface and read the values given multiple times and each reading has an interval of 2 seconds.
The second test is the same but placed on a black surface.

```
12:52:51.779 -> A1 = 751  A2 = 990  A3 = 990  A4 = 1000  A5 = 985  A6 = 1001
12:52:52.800 -> A1 = 751  A2 = 990  A3 = 990  A4 = 1000  A5 = 984  A6 = 1001
12:52:53.794 -> A1 = 750  A2 = 990  A3 = 990  A4 = 1000  A5 = 984  A6 = 1001
12:52:54.779 -> A1 = 753  A2 = 990  A3 = 990  A4 = 1000  A5 = 985  A6 = 1001
12:52:55.812 -> A1 = 754  A2 = 991  A3 = 990  A4 = 1000  A5 = 985  A6 = 1001
12:52:56.779 -> A1 = 754  A2 = 991  A3 = 990  A4 = 1000  A5 = 985  A6 = 1001
12:52:57.814 -> A1 = 756  A2 = 991  A3 = 990  A4 = 1000  A5 = 985  A6 = 1001
12:52:58.787 -> A1 = 750  A2 = 990  A3 = 989  A4 = 1000  A5 = 985  A6 = 1001
12:52:59.814 -> A1 = 750  A2 = 990  A3 = 989  A4 = 1000  A5 = 985  A6 = 1000
12:53:06.893 -> A1 = 749  A2 = 990  A3 = 989  A4 = 1001  A5 = 986  A6 = 1000
12:53:08.922 -> A1 = 749  A2 = 990  A3 = 989  A4 = 1001  A5 = 986  A6 = 1000
12:53:10.893 -> A1 = 749  A2 = 990  A3 = 989  A4 = 1001  A5 = 986  A6 = 1000
```

The wiring of the line sensor can be seen in the visual overview.

HC-12 Wireless Communication

A HC-12 module will be used to make wireless communication happen between the other Relay-Bots and the dashboard.

- Long-distance wireless transmission inside buildings limited to 50m
- Working frequency range (433.4-473.0MHz, up to 100 communication channels)

HC-12 Testing

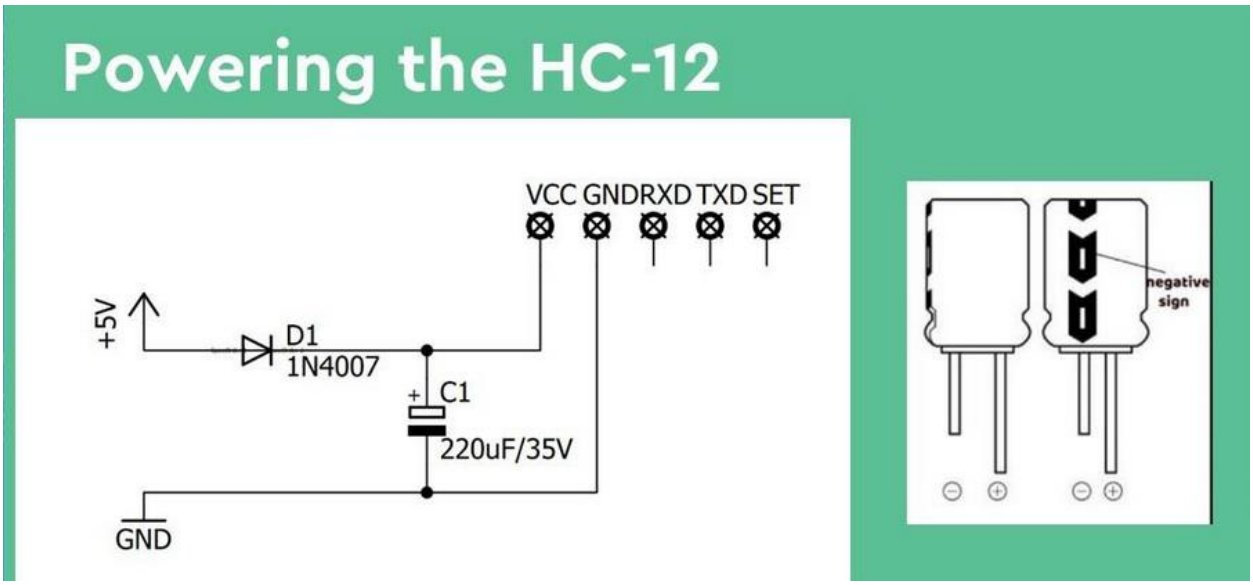
We tested the functionality of the HC-12, you cannot send 2 messages at the same time it will result in a time-out for an undefined amount of time and you won't be able to receive or send data during that.
The reach of the modules in an indoors setting is roughly 50m. We tested it by sending a message every couple seconds while walking away. You will receive invalid characters

once you get around those distances as shown below.

```
14:16:46.600 -> ������������a
14:17:41.180 -> a
14:17:46.879 -> b
14:18:00.495 -> Say something to gg
14:20:13.178 -> ������������hi
14:21:35.606 -> Our limit is around 100m.
14:21:42.637 -> Then we get bad signal
14:22:01.067 -> ������g
14:23:30.190 -> y
14:23:30.354 -> y
14:23:30.504 ->
14:23:30.643 -> y
14:23:30.816 -> y
14:23:32.428 -> Hey
14:23:40.621 -> Ho
```

It says in the messages that the limit is around 100m, but in realistic standpoints it shouldn't exceed 50m to make sure the data it transmit and receives is correct.

The setup for the wiring is as follows:



Arduino Uno R4 WIFI

The Arduino Uno will act as the central base station for our operations and communication between the Arduino Nano's and the Team Website. Further explanation will be given under the section Communication/Infrastructure.

- 32-bit microcontroller: RA4M1 from Renesas.

- ESP32-S3-MINI-1-N8: module for Wi-Fi® and Bluetooth® connectivity.
- 12x8 LED matrix.
- Operating voltage: 5V. (ESP32-S3 is 3.3 V).
- Input Voltage: 6 - 24V.
- Digital I/O Pins: 14.
- Analog input pins: 6.
- PWM pins: 6.
- External interrupts: 2,3.
- DC Current per I/O Pin: 8 mA.
- Clock speed RA4M1: 48 MHz.
- Clock speed ESP32-S3-MINI-1-N8: 240 MHz.
- Memory RA4M1: 256 kB Flash, 32 kB RAM.
- Memory ESP32-S3-MINI-1-N8: 384 kB ROM, 512 kB SRAM.

Software

To develop our Relay Bot, we will use the following software:

Arduino IDE:

The official way of programming the Arduino Nano is the **Arduino IDE**. It is a simple IDE designed specifically for sending code to Arduino. Additionally, this software includes a **library system**, which makes it easier to incorporate libraries into the code. Libraries allow you to add pre-written code to the Relay Bot's program, significantly shortening the development process. Only allowed to use a singular library, each subgroup has to make the choice on which they want to use.

Visual Studio Code:

Instead of the Arduino IDE, it is also possible to use **Visual Studio Code**. Various extensions are available for Visual Studio Code that enable Arduino programming.

This software offers built-in code autocomplete and a better debugging system compared to the Arduino IDE. However, setting up Visual Studio Code is more complex, and it is less reliable than the Arduino IDE.

To make Visual Studio Code compatible with Arduino, we use the **Platform IO** extension. This extension provides functionalities such as uploading and verifying code and selecting ports.

For using the Arduino's serial monitor another extension is needed, the most used one is the **Serial Monitor** extension. But there are other possibilities.

Communications/Infrastructure

HC-12:

HC-12 is a popular wireless communication module used for sending and receiving data via serial communication. It operates on the 433 MHz frequency and commonly used in embedded system because of low-power, short-range communication between two devices like our Arduino board.

In our Arduino one Arduino is sending data to another Arduino via HC-12 modules.

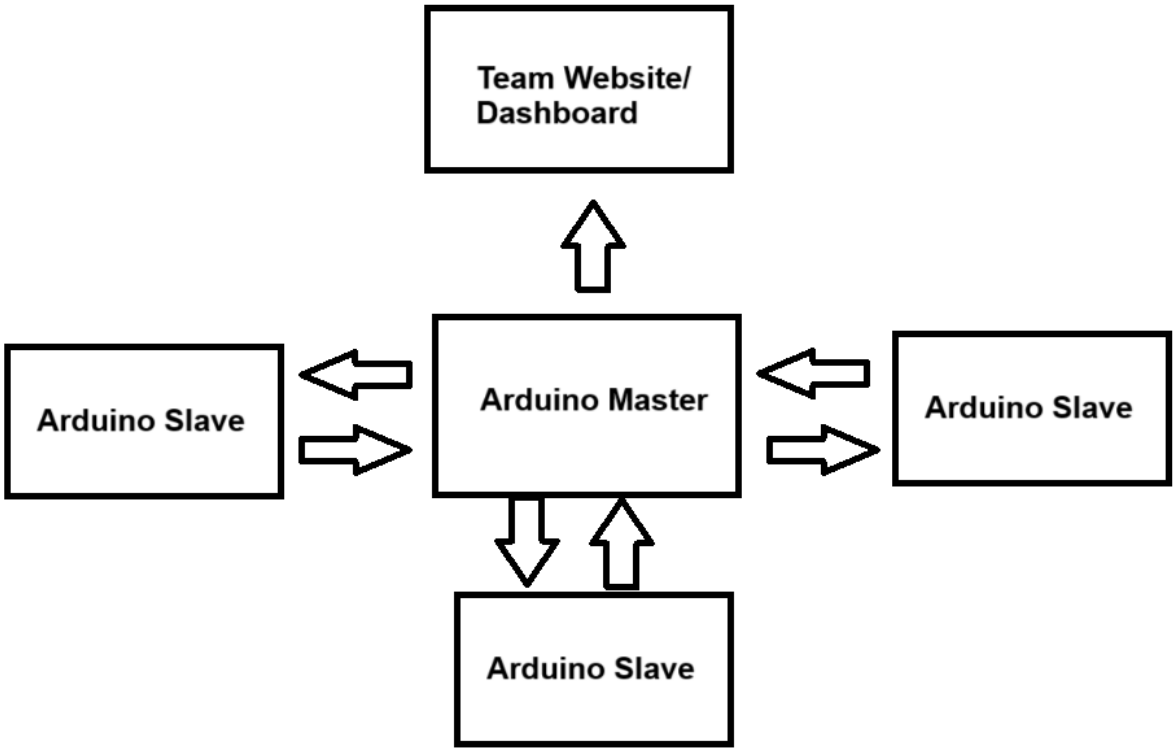
Master-Slave Protocol:

The Master-Slave Protocol is a module of communication in which one device (Master) controls the communication, and the others (Slaves) respond to the request. This is commonly used in communication systems where the Master device needs to coordinate and control the action of other devices.

- Master Device: Initiates communication, sends requests, and controls the flow of data.
- Slave Devices: Respond only when addressed by the master; they cannot initiate communication on their own.
- Applications: Common in serial communication protocols like I2C and SPI where one device (e.g., a microcontroller) acts as the master, and others (e.g., sensors, actuators) act as slaves.

In our relay bot the Master Arduino could be in charge of controlling the movement, while slave devices could be responsible for handling sensors or relays.

The master is also the one that communicates with the dashboard of the team-website to upload the data the slaves send to the master.



Database Communication:

For relay on our project, we are working on a line maze, and we share our code on git hub to test it and change it and test to make sure that it is working and finalize the code there. But the code space that we are using to connect our GitHub together is VS Code(visual Studio Code).

Data storage

The Relay Bot uses a database for data storage because it is a small project, data protection is not a high priority. RAID 1, also known as mirroring, ensures that all data is stored on two drives. This creates an exact copy of the data, contributing to a high level of data protection and reliability. But costs are lower without needing an extra hard drive. If more data protection is needed. Using a RAID setup would be advisable.

Data will also be stored through SQLite3 in a database where the sensor data of the individual robots will be stored and displayed on the team-website.

Advisory report

infrared distance sensor

Analysis of the Problem

- **Environmental Sensitivity**: Ultrasound sensors are very sensitive to environmental factors like temperature and humidity, which can lead to inaccuracies in distance measurements. On the other hand, infrared sensors are more stable in these conditions, making them more reliable in some environments.
- **Power Consumption**: One of the best things about infrared sensors is that they use less power. This is especially useful for battery-powered devices or systems that need to run for long periods without recharging.
- **Accuracy and Range**: The downside of infrared sensors is that they have a shorter range and aren't as accurate over long distances. If we need precise measurements over larger areas, ultrasound sensors would still be the better option.

Possible Solutions or Alternatives

- **Infrared with Modulation**: To tackle the range limitation, we can use modulated infrared signals. This will help increase the range and reduce interference from ambient light, making the system more reliable.
- **Hybrid Systems**: We could combine infrared sensors with ultrasound sensors in a hybrid setup. This way, we get the speed and cost-effectiveness of infrared while still maintaining the range and accuracy of ultrasound.
- **Sensor Calibration and Filtering**: To make infrared sensors more effective, it might be a good idea to calibrate them for our specific environment and use advanced filtering techniques to minimize noise and improve accuracy.

Recommendations

1. **Use Infrared for Short-Range Applications**: Infrared sensors are perfect when we need something fast, cost-efficient, and for short-range detection, like object detection or indoor navigation.
2. **Integrate Sensor Fusion**: We could think about integrating infrared sensors with other types of sensors (like ultrasound, cameras, or LiDAR) to improve accuracy, especially in more complex environments.
3. Infrared that can be used to replace the ultrasound we have in BattleBot: https://www.otronic.nl/nl/ir-analoge-afstand-meetsensor-module-infrarood-

ber.html?source=googlebase&gad_source=1&gclid=Cj0KCQiAs5i8BhDmARIsAGE4xHw8TsHKZOzNIBaCEssTDgppQSuo9tjOWDh1CnQiqa2Lsjl5t5xoAp4aAqeAEALw_wcB

Specifications

Size: 40mm x 18mm x 12mm/1.56" x 0.70" x 0.46"(inch) (approx.)
GP2D12 is an Infrared Range Sensor which is commonly used in various fields for the distance between the robot and the obstacles.
Specifications:
Measuring range: 4cm to 30cm
The maximum allowable Angle: > 40
The power supply voltage: 4.5 to 5.5V
The average current consumption: 35mA
Peak current consumption: about 200mA
The frequency of updates/cycle: 25Hz/40ms
Analog output noise: < 200mV

Improving Communication

Our group has faced some challenges with communication, which has impacted our ability to work as effectively as we'd like. Looking back, it's clear that for future projects, it's crucial to establish clear communication from the beginning. It's important that everyone knows how we're going to stay in touch and share information throughout the project.

By making sure everyone is aligned on communication methods and staying connected as a team, we can avoid situations where members feel disconnected. This will help us stay on track and work more smoothly, leading to a more efficient and successful project.

User Stories

As an user, I want my relay-bot to be able to follow a line so that it can finish some gnarly parkours which would be really cool.

As an user, I want to have the relay-bot be able to pick up objects so that I can move stuff from one point to another so that I don't have to do it.

As an user, I want the relay-bot be able to send it's speed to a website so that I can see how fast it actually goes.

As an user, I want the robot be able to see objects so it can avoid them, so it doesn't get damaged.

As an user, I want the robot to be able to have light indicators when it turns so I can see when he turns.

As a user, I want the robot to find the line with the sensor so I can track the Line.

As a user, I want each sensor on the robot to track the line separately and send it to the terminal to check which position the line is on the sensor, so I can adjust the line independently.

As a user, I want the robot to move forward with the line in the middle 2 sensors.

As a user, I want the relay-bot to have a manual override mode, so I can control it remotely in case it gets stuck or I want to steer it directly.

As a user, I want the relay-bot to monitor and display its battery level on the website, so I know when it needs to be recharged.

As a user, I want to adjust the relay-bot's speed dynamically, so it can handle different terrains or tasks more effectively.

As a user, I want the relay-bot to have an emergency stop feature, so I can immediately halt its movement in case of unexpected situations.

As a user, I want to control the relay-bot using voice commands, so I can operate it hands-free.

As a user, I want the relay-bot to follow different line colors, so it can handle various tracks without needing modifications.

As a user, I want the relay-bot to enter energy-saving mode when idle, so it conserves battery power.

As a user, I want to view the relay-bot's real-time status on the website dashboard, so I can monitor its performance and adjust as needed.

As a user, I want the website to allow me to control the relay-bot remotely, so I can issue commands like start, stop, speed adjustments, and switch between modes.

Requirements

Team	Requirement ID	Priority
Los Perdidos	1	Must have
Functional Requirements		
The relay-bot needs to be able to follow a line.		
Non-Functional Requirements		
None		
Acceptation Criteria		
The relay-bot should effortlessly follow the line without making mistakes.		

Team	Requirement ID	Priority
Los Perdidos	2	Must have
Functional Requirements		
The relay-bot needs to be able to pick up an object.		
Non-Functional Requirements		
None		
Acceptation Criteria		
The relay-bot is holding an object without letting it drop.		

Team	Requirement ID	Priority
Los Perdidos	3	Must have
Functional Requirements		
The relay-bot needs to be able to send it’s sensor data’s to the dashboard		
Non-Functional Requirements		
Make the readings graphical.		
Acceptation Criteria		
The relay-bot is accurately showing the correct sensor data		

Team	Requirement ID	Priority

Los Perdidos	4	Must have
Functional Requirements		
The relay-bot needs to be able to see objects		
Non-Functional Requirements		
None		
Acceptation Criteria		
The relay-bot is able to perceive objects and be able to avoid them.		

Team	Requirement ID	Priority
Los Perdidos	5	Must have
Functional Requirements		
The relay-bot should be able to use neopixels to indicate what it is doing.		
Non-Functional Requirements		
It should have different colors.		
Acceptation Criteria		
During driving it should be correctly showing the neopixels.		

Team	Requirement ID	Priority
Los Perdidos	6	Must have
Functional Requirements		
The relay-bot needs to find the line with the sensor		
Non-Functional Requirements		
None		
Acceptation Criteria		
The relay-bot should find the Line and get values		

Team	Requirement ID	Priority
Los Perdidos	7	Must have
Functional Requirements		
The sensors should individualy find the line		
Non-Functional Requirements		
None		
Acceptation Criteria		
The sensor should know which position is the line		

Team	Requirement ID	Priority
Los Perdidos	8	Must have

Functional Requirements
The relay-bot should move forward with the line
Non-Functional Requirements
None
Acceptation Criteria
The relay-bot should find the Line and move along keeping the line on the middle 2 sensors.

Team	Requirement ID	Priority
Los Perdidos	9	Must not have
Functional Requirements		
The relay-bot should be able to switch to manual override mode.		
Non-Functional Requirements		
None		
Acceptation Criteria		
The relay-bot should be controllable remotely using a manual override mode when activated.		

Team	Requirement ID	Priority
Los Perdidos	10	Must have
Functional Requirements		
The relay-bot should monitor its battery level. The relay-bot should send battery level data to the website for display.		
Non-Functional Requirements		
None		
Acceptation Criteria		
The website should display the current battery level in real-time.		

Team	Requirement ID	Priority
Los Perdidos	11	Must have
Functional Requirements		
The relay-bot should be able to adjust its speed based on user input.		
Non-Functional Requirements		
None		
Acceptation Criteria		
The relay-bot should change speed as per user request, adjusting for different terrains or tasks.		

Team	Requirement ID	Priority
Los Perdidos	12	Should not have
Functional Requirements		
The relay-bot should have an emergency stop button that immediately halts its movement.		
Non-Functional Requirements		
None		
Acceptation Criteria		
The relay-bot should immediately stop when the emergency stop button is pressed.		

Team	Requirement ID	Priority
Los Perdidos	13	Must not have
Functional Requirements		
The relay-bot should recognize and respond to basic voice commands (e.g., move forward, stop).		
Non-Functional Requirements		
None		
Acceptation Criteria		
The relay-bot should move or stop according to voice commands.		

Team	Requirement ID	Priority
Los Perdidos	14	Should have
Functional Requirements		
Thr relay-bot should be able to follow lines of different colors.		
Non-Functional Requirements		
None		
Acceptation Criteria		
The relay-bot should follow different line colors without requiring changes to its configuration.		

Team	Requirement ID	Priority
Los Perdidos	15	Shouldn’t have
Functional Requirements		
The relay-bot should automatically enter energy-saving mode when idle for a specified period.		
Non-Functional Requirements		
None		
Acceptation Criteria		

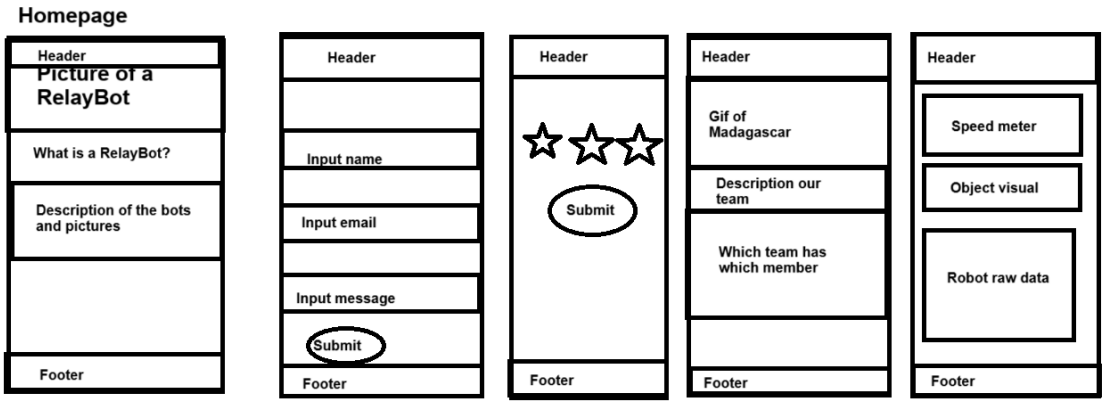
The relay-bot should reduce power consumption when not in active use.

Team	Requirement ID	Priority
Los Perdidos	16	Must have
Functional Requirements		
The website should display the relay-bot's real-time status (e.g., battery level, current speed, mode, etc.). The website should update the status regularly.		
Non-Functional Requirements		
None		
Acceptation Criteria		
The website should reflect the current state of the relay-bot in real-time, allowing users to view important metrics like battery and speed.		

Team	Requirement ID	Priority
Los Perdidos	17	Should not have
Functional Requirements		
The website should provide a remote control interface for controlling the relay-bot. The interface should allow for starting, stopping, adjusting speed, and switching between modes.		
Non-Functional Requirements		
None		
Acceptation Criteria		
The website should have a control panel that lets users issue commands to the relay-bot remotely.		

Team-Website

We had a rudimentary sketch about the team-website as shown below.



The pages all lead to each other and is visually representative of the group Los-Perdidos.
The page can be found on: https://los-perdidos.onrender.com/

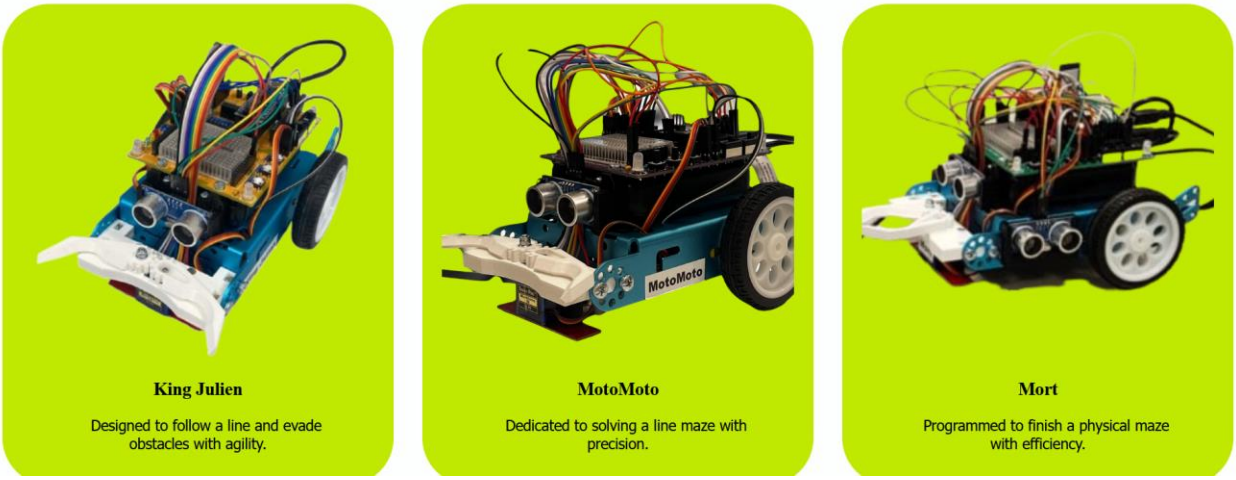
And also on the team GitHub: https://github.com/Lennon-Tijmes/Los-Perdidos/tree/Team-Website

Below are the pictures of the Team-Website and dashboard.
Homepage:

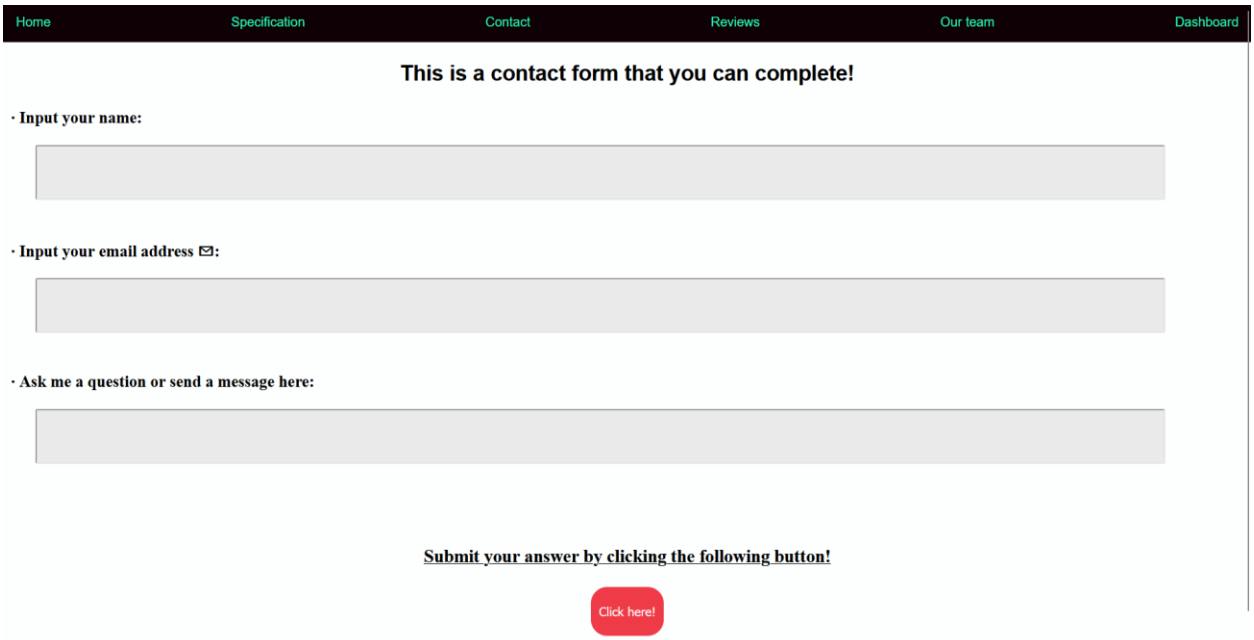
What is a Relaybot?

A Relaybot is a robot designed to transport or deliver items from one location to another. They typically use sensors, cameras, and artificial intelligence (AI) to navigate environments, avoid obstacles, and reach their destinations safely. They may follow pre-programmed routes, utilize maps, or adapt to real-time changes in their surroundings. In some cases, relay robots can be controlled or monitored remotely, while others operate fully autonomously. These robots are designed to improve efficiency, productivity, and safety by reducing human labor in tasks that can be repetitive, time-consuming, or hazardous.

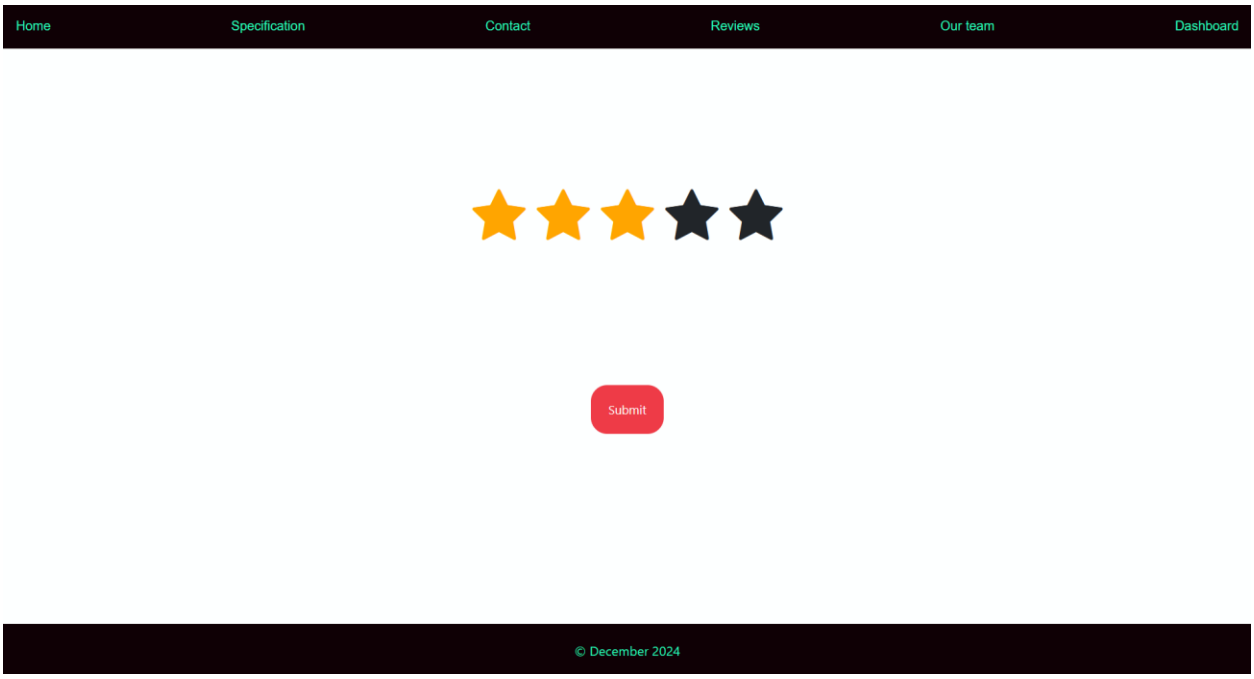
Meet our Relaybots:



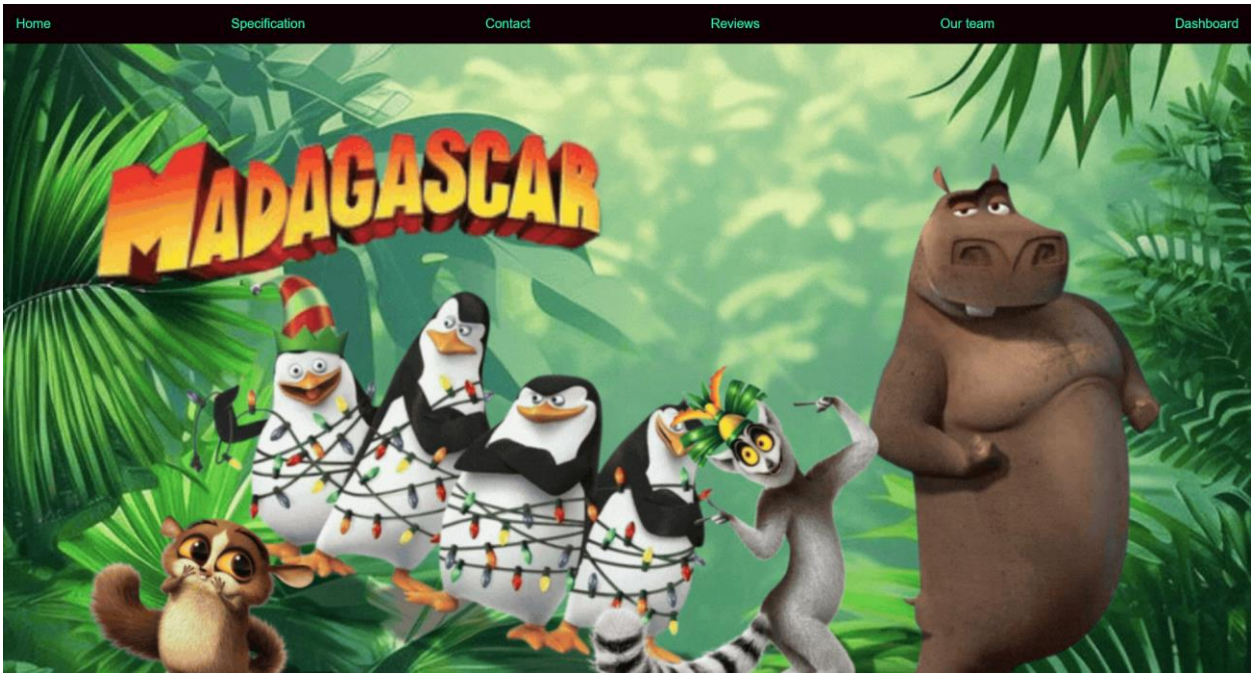
Contact page:



Review page:


Home
Specification
Contact
Reviews
Our team
Dashboard
Submit
© December 2024

Our Team:


Home
Specification
Contact
Reviews
Our team
Dashboard
MADAGASCAR

We are "Los Perdidos," a company that specializes in the production of commercial hardware. With a commitment to innovation and quality, we design and manufacture a wide range of hardware solutions tailored to meet the needs of businesses across various industries. Our team is dedicated to providing exceptional customer service, ensuring that every product is crafted with precision, efficiency, and the highest industry standards.

Meet our professionals

Team 1

- Fabiana
- Alexia

Team 2

- Matin
- Goncalo
- Diogo

Team 3

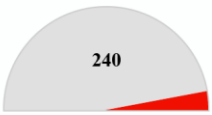
- Peter
- Lennon

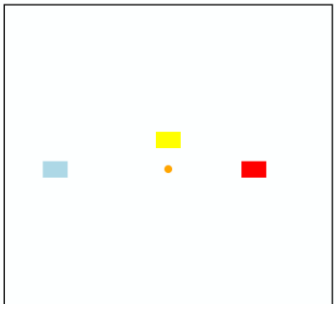
© December 2024

Dashboard:

Home Specification Contact Reviews Our team Dashboard

Robot Dashboard





Mort

Speed	Object Left	Object Right	Object Middle	Line
240.0	20.8	15.7	5.4	On Line
40.0	40.8	25.7	5.4	On Line
255.0	2.1	1.7	1.4	On Line
200.0	20.5	15.0	18.0	No Line Detected
45.5	150.0	120.0	135.0	No Line Detected
1.5	10.2	11.0	9.8	detected

Moto-Moto

Speed	Object Distance	Line
60.0	200.0	Line Detected

King Julien

Speed	Object Distance	Line
50.0	100.0	Line Detected

© 2025

RelayBot – codes

The codes below are the current RelayBot codes and another link to their corresponding branches.

Line Follower – King Julien

King Julien's task is to follow a line and avoid an object while delivering a cone to the next robot.

The code can be found on: https://github.com/Lennon-Tijmes/Los-Perdidos/blob/Line-Follower/lf

```
//#define motorStop

// Motor speed definitions for object evasion
#define SPEED_LEFT 200
#define SPEED_RIGHT 220
#define TURN_SPEED_LEFT 60
#define TURN_SPEED_RIGHT 70

#define MOTOR_LEFT_FWD 9
#define MOTOR_LEFT_BWD 11
#define MOTOR_RIGHT_FWD 10
#define MOTOR_RIGHT_BWD 3
```

```
// Sensor pins for ultrasonic
#define TRIG_PIN 12
#define ECHO_PIN 13

// Line following sensor array (assuming 8 sensors)
const unsigned char TRACK_SENSORS[] = {A7, A6, A5, A4, A3, A2,
A1, A0};  // 8 sensors
int sensorReadings[8] = {0, 0, 0, 0, 0, 0, 0, 0};
const unsigned int thresholdBlack = 800;  // Adjust for black
line detection
bool lastDirectionWasLeft = false;

void setup() {
  // Set motor direction pins as outputs
  pinMode(MOTOR_LEFT_FWD, OUTPUT);
  pinMode(MOTOR_LEFT_BWD, OUTPUT);
  pinMode(MOTOR_RIGHT_FWD, OUTPUT);
  pinMode(MOTOR_RIGHT_BWD, OUTPUT);

  // Set ultrasonic sensor pins
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);

  // Start serial communication for debugging
  Serial.begin(9600);
}

void loop() {
  long duration, distance;

  if(millis() % 200 == 0)
  {
    // Measure distance from the ultrasonic sensor
    digitalWrite(TRIG_PIN, LOW);
    delayMicroseconds(2);  // Ensure trigger is off
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);  // Send a 10µs pulse
    digitalWrite(TRIG_PIN, LOW);
    duration = pulseIn(ECHO_PIN, HIGH);  // Measure pulse
duration
```

```
    distance = duration * 0.034 / 2;     // Calculate distance
in cm

    // If an object is detected within 20 cm, evade
    if (distance > 0 && distance <= 20) {
      Serial.write("evade\n");
      evadeObstacle();  // Temporarily evade obstacle
    }
  }
  followLine();  // Follow the line when no obstacle is detected
}

// Line following function (using your original logic with 50ms
delay)
void followLine() {
  // Read sensor values
  for (int i = 0; i < 8; i++) {
    sensorReadings[i] = analogRead(TRACK_SENSORS[i]);  // Read
analog values from the sensors
  }

  // Check the position of the line
  int leftCount = 0;
  int rightCount = 0;

  for (int i = 0; i < 4; i++) {  // Check left side sensors (0-
3)
    if (sensorReadings[i] > thresholdBlack) {
      leftCount++;
    }
  }

  for (int i = 4; i < 8; i++) {  // Check right side sensors (4-
7)
    if (sensorReadings[i] > thresholdBlack) {
      rightCount++;
    }
  }

  // If both left and right sides detect the line, move forward
  if (leftCount > 0 && rightCount > 0) {
```

```
    moveForward();
  }
  // If left side detects the line more, turn left
  else if (leftCount > 0) {
    turnLeft();
    lastDirectionWasLeft = true;  // Remember the last direction
  }
  // If right side detects the line more, turn right
  else if (rightCount > 0) {
    turnRight();
    lastDirectionWasLeft = false;  // Remember the last
direction
  }
  // If no sensors detect the line, continue turning in the last
known direction
  else {
    if (lastDirectionWasLeft) {
      turnLeft();  // Continue turning left
    } else {
      turnRight();  // Continue turning right
    }
  }
}

// Object evasion function
void evadeObstacle() {

 /// Turn right to evade
  turnRight();
  delay(700);
  moveForward();
  delay(500);
  turnLeft();
  delay(2000);
  moveForward();
  delay(800);
  turnLeft();
  delay(2000);
  moveForward();
  delay(1000);
  turnRight();
```

```
  delay(2000);
  moveForward();
  delay(400);



}

void moveForward() {
  analogWrite(MOTOR_LEFT_FWD, SPEED_LEFT);
  analogWrite(MOTOR_RIGHT_FWD, SPEED_RIGHT);
}

void turnLeft() {
  // Serial.write("left\n");
  analogWrite(MOTOR_RIGHT_FWD, TURN_SPEED_RIGHT);
}

void turnRight() {
  // Serial.write("right\n");
  analogWrite(MOTOR_LEFT_FWD, TURN_SPEED_LEFT);
}
```

Line Maze – MotoMoto

MotoMoto's job is to figure out a line maze with sharp 90-degree turns, which already requires some solid precision to keep it on track. On top of that, there's an obstacle sitting in the way. When it spots this obstacle, MotoMoto has to pull off a smooth 180-degree turn to dodge it and find another path.

The code can be found on: https://github.com/Lennon-Tijmes/Los-Perdidos/tree/Line-Maze

Line Maze MotoMoto code:

```
#include <Adafruit_NeoPixel.h>

// Pin and configuration constants
#define   GRIPPER_PIN           8      // Servo pin for the
gripper
```

```
#define   GRIPPER_OPEN    1800   // Pulse width to open the
gripper
#define   GRIPPER_CLOSED  900    // Pulse width to close the
gripper
#define   SERVO_INTERVAL        20    // Time between servo
pulses (ms)
#define   GRIPPER_TOGGLE        1000  // Toggle gripper every
second
#define   TRIG_PIN              12    // Ultrasonic sensor
trigger pin
#define   ECHO_PIN              11    // Ultrasonic sensor echo
pin
#define   MOTOR_LEFT_BACKWARD   5     // Motor control pin (left
backward)
#define   MOTOR_LEFT_FORWARD    9     // Motor control pin (left
forward)
#define   MOTOR_RIGHT_BACKWARD  10    // Motor control pin
(right backward)
#define   MOTOR_RIGHT_FORWARD   6     // Motor control pin
(right forward)
#define   MOTOR_DEVIATION       14    // Correction for motor
speed imbalance
#define   SLAVE_ID              2     // Identifier for this
slave robot
#define   NEOPIXEL_PIN          13    // NeoPixel LED control
pin
#define   NUMPIXELS             4     // Number of NeoPixel LEDs

// Light sensor pins and initial threshold
const int LIGHT_SENSOR[8] = {A0, A1, A2, A3, A4, A5, A6, A7};
int LIGHT_VALUE  =   850;  // Initial light threshold value

// NeoPixel object setup
Adafruit_NeoPixel pixels(NUMPIXELS, NEOPIXEL_PIN, NEO_GRB +
NEO_KHZ800);

// Timing variables for various intervals
long currentMillis;
bool currentLightState = false;
unsigned long previousMillis = 0;
const long interval = 100; // Interval for periodic tasks (ms)
```

```
bool end = false;

// Ultrasonic sensor timing variables
unsigned long distanceMillis = 0;
float duration = 0;

void setup() {
  // Configure pins
  pinMode(GRIPPER_PIN, OUTPUT);
  digitalWrite(GRIPPER_PIN, LOW);
  pinMode(MOTOR_LEFT_BACKWARD, OUTPUT);
  pinMode(MOTOR_LEFT_FORWARD, OUTPUT);
  pinMode(MOTOR_RIGHT_BACKWARD, OUTPUT);
  pinMode(MOTOR_RIGHT_FORWARD, OUTPUT);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  pixels.begin();

  // Serial communication setup
  Serial.begin(9600);
  Serial.println("Slave (MotoMoto) started");

  // Initialize light sensor pins
  for (int i = 0; i < 8; i++) {
    pinMode(LIGHT_SENSOR[i], INPUT);
  }

  // Turn off lights and begin initial calibration
  stopLights();
  start();
}

void loop() {
  // Main loop: line-following and LED control
  followTheLine();
  forwardLights();

  // Check for incoming serial messages from the master
  if (Serial.available()) {
    String message = Serial.readStringUntil('\n');
    Serial.print("Received: ");
```

```
    Serial.println(message);

    // Respond to polling requests from the master
    if (message.length() >= 2 && message[0] == SLAVE_ID + '0' &&
message[1] == '?') {
      Serial.print("Responding to Master: Slave ");
      Serial.print(SLAVE_ID);
      Serial.println(" Response");

      // Simulated sensor data
      float speed = random(50, 150) / 10.0; // Random speed (5.0
to 15.0)
      float object_middle = random(10, 100) / 10.0; // Random
object distance (cm)
      String line = "On Line";  // Placeholder for line sensor
state

      // Format the response string
      String response = String(speed, 1) + "," +
                        String(object_middle, 1) + "," +
                        line;

      // Send the response back to the master
      Serial.println(response);
    }
  }
}

void followTheLine() {
  // Reads light sensors and adjusts the robot's movement
accordingly
  if (analogRead(LIGHT_SENSOR[7]) > LIGHT_VALUE ||
analogRead(LIGHT_SENSOR[6]) > LIGHT_VALUE) {
    specialTurnLeft();
  } else if (analogRead(LIGHT_SENSOR[2]) > LIGHT_VALUE ||
analogRead(LIGHT_SENSOR[3]) > LIGHT_VALUE) {
    driveForward(210);
  } else if (analogRead(LIGHT_SENSOR[5]) > LIGHT_VALUE ||
analogRead(LIGHT_SENSOR[4]) > LIGHT_VALUE) {
    turnLeft(205);
  } else if (analogRead(LIGHT_SENSOR[0]) > LIGHT_VALUE ||
```

```
analogRead(LIGHT_SENSOR[1]) > LIGHT_VALUE) {
    turnRight(200);
  } else if (analogRead(LIGHT_SENSOR[0]) < LIGHT_VALUE &&
analogRead(LIGHT_SENSOR[3]) < LIGHT_VALUE &&
analogRead(LIGHT_SENSOR[5]) < LIGHT_VALUE &&
analogRead(LIGHT_SENSOR[7]) < LIGHT_VALUE) {
    specialTurnRight();
  } else {
    driveForward(210);
  }
}

void theEnd() {
  // Executes the end sequence when the task is complete
  motorStop();
  driveBackwards(245);
  delay(300);
  motorStop();
  for (int i = 0; i < 8; i++) {
    servo(GRIPPER_OPEN);
    delay(100);
  }
  delay(10);
  driveBackwards(245);
  delay(3000);
  winLights();
  motorStop();
}

void servo(int pulse) {
  // Controls the servo motor by generating a pulse of a given
width
  static unsigned long timer;
  static int pulse1;
  if (pulse > 0) {
    pulse1 = pulse;
  }
  if (millis() > timer) {
    digitalWrite(GRIPPER_PIN, HIGH);
    delayMicroseconds(pulse1);
    digitalWrite(GRIPPER_PIN, LOW);
```

```
    timer = millis() + SERVO_INTERVAL;
  }
}

void driveForward(int speed) {
  // Moves the robot forward at a given speed
  analogWrite(MOTOR_LEFT_BACKWARD, 0);
  analogWrite(MOTOR_LEFT_FORWARD, speed - MOTOR_DEVIATION);
  analogWrite(MOTOR_RIGHT_BACKWARD, 0);
  analogWrite(MOTOR_RIGHT_FORWARD, speed);
}

void turnLeft(int speed) {
  // Turns the robot left by adjusting motor speeds
  analogWrite(MOTOR_LEFT_BACKWARD, speed);
  analogWrite(MOTOR_LEFT_FORWARD, 0);
  analogWrite(MOTOR_RIGHT_BACKWARD, 0);
  analogWrite(MOTOR_RIGHT_FORWARD, speed);
}

void specialTurnLeft() {
  // Performs a special left turn when necessary
  driveForward(210);
  delay(300);
  if (analogRead(LIGHT_SENSOR[5]) > LIGHT_VALUE &&
analogRead(LIGHT_SENSOR[2]) > LIGHT_VALUE) {
    theEnd();
    while (true) {}
  }
  turnLeft(215);
  delay(300);
  while (true) {
    delay(1);
    if (analogRead(LIGHT_SENSOR[2]) > LIGHT_VALUE ||
analogRead(LIGHT_SENSOR[3]) > LIGHT_VALUE) {
      break;
    }
  }
  motorStop();
}
```

```
void specialTurnRight() {
  // Performs a special right turn when necessary
  analogWrite(MOTOR_LEFT_BACKWARD, 0);
  analogWrite(MOTOR_LEFT_FORWARD, 255);
  analogWrite(MOTOR_RIGHT_BACKWARD, 255);
  analogWrite(MOTOR_RIGHT_FORWARD, 0);
  while (true) {
    delay(1);
    if (analogRead(LIGHT_SENSOR[4]) > LIGHT_VALUE ||
analogRead(LIGHT_SENSOR[5]) > LIGHT_VALUE) {
      break;
    }
  }
  motorStop();
}

void turnRight(int speed) {
  // Turns the robot right by adjusting motor speeds
  analogWrite(MOTOR_LEFT_BACKWARD, 0);
  analogWrite(MOTOR_LEFT_FORWARD, 255);
  analogWrite(MOTOR_RIGHT_BACKWARD, 255);
  analogWrite(MOTOR_RIGHT_FORWARD, 0);

  delay(10);

  analogWrite(MOTOR_LEFT_BACKWARD, 0);
  analogWrite(MOTOR_LEFT_FORWARD, speed);
  analogWrite(MOTOR_RIGHT_BACKWARD, speed);
  analogWrite(MOTOR_RIGHT_FORWARD, 0);
}

void driveBackwards(int speed) {
  // Moves the robot backward at a given speed
  analogWrite(MOTOR_LEFT_BACKWARD, speed);
  analogWrite(MOTOR_LEFT_FORWARD, 0);
  analogWrite(MOTOR_RIGHT_BACKWARD, speed);
  analogWrite(MOTOR_RIGHT_FORWARD, 0);
}

void motorStop() {
  // Stops all motors
```

```
  analogWrite(MOTOR_LEFT_BACKWARD, 0);
  analogWrite(MOTOR_LEFT_FORWARD, 0);
  analogWrite(MOTOR_RIGHT_BACKWARD, 0);
  analogWrite(MOTOR_RIGHT_FORWARD, 0);
}

void start() {
  // Initializes the robot and calibrates the light sensors
  Serial.print("Current LIGHT_VALUE: ");
  Serial.println(LIGHT_VALUE);

  int blackLineSum = 0;
  int blackLineCount = 0;

  for (int i = 0; i < 3; i++) {
    while (getDistance() > 24);
  }
  stopLights();
  forwardLights();
  driveForward(255);

  while (blackLineCount < 4) {
    while (true) {
      if (analogRead(LIGHT_SENSOR[3]) > LIGHT_VALUE) {
        break;
      }
    }
    while (true) {
      if (analogRead(LIGHT_SENSOR[3]) < LIGHT_VALUE) {
        break;
      }
    }
    blackLineSum += getAverageLightValue();
    blackLineCount++;
  }
  motorStop();

  LIGHT_VALUE = blackLineSum / blackLineCount;

  Serial.print("New LIGHT_VALUE: ");
  Serial.println(LIGHT_VALUE);
```

```
  for (int i = 0; i < 100; i++) {
    delay(10);
    servo(GRIPPER_CLOSED);
  }
  turnLeft(200);
  delay(600);
  while (true) {
    if (analogRead(LIGHT_SENSOR[4]) > LIGHT_VALUE) {
      break;
    }
  }
  motorStop();
}

int getAverageLightValue() {
  // Calculates the average light sensor value
  int sum = 0;
  for (int i = 0; i < 8; i++) {
    sum += analogRead(LIGHT_SENSOR[i]);
  }
  return sum / 8;
}

float getDistance() {
  // Calculates the distance to an object using the ultrasonic
sensor
  if (millis() >= distanceMillis) {
    distanceMillis = millis() + 200;
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);
    duration = pulseIn(ECHO_PIN, HIGH);
    Serial.println(0.017 * duration);
    return 0.017 * duration;
  }
}

void forwardLights() {
  // Sets the NeoPixel LEDs to indicate forward movement
  pixels.setPixelColor(0, pixels.Color(255, 0, 0));
  pixels.setPixelColor(1, pixels.Color(255, 0, 0));
```

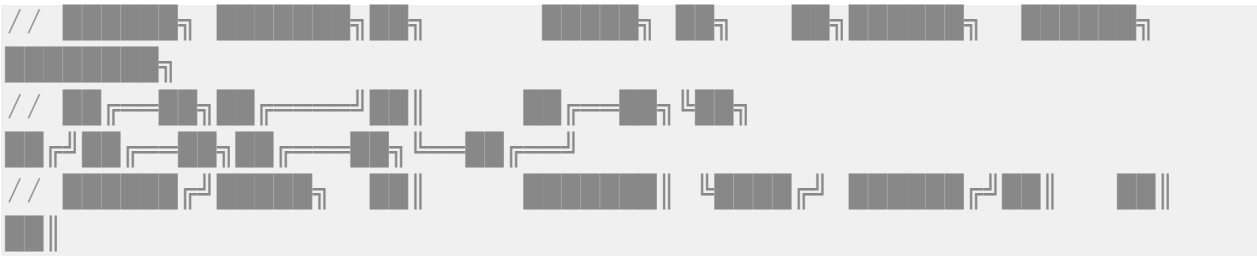
```
  pixels.setPixelColor(2, pixels.Color(255, 0, 0));
  pixels.setPixelColor(3, pixels.Color(255, 0, 0));
  pixels.show();
}

void stopLights() {
  // Sets the NeoPixel LEDs to indicate a stopped state
  pixels.setPixelColor(0, pixels.Color(0, 255, 0));
  pixels.setPixelColor(1, pixels.Color(0, 255, 0));
  pixels.setPixelColor(2, pixels.Color(0, 255, 0));
  pixels.setPixelColor(3, pixels.Color(0, 255, 0));
  pixels.show();
}

void winLights() {
  // Sets the NeoPixel LEDs to indicate task completion
  pixels.setPixelColor(0, pixels.Color(0, 0, 255));
  pixels.setPixelColor(1, pixels.Color(0, 0, 255));
  pixels.setPixelColor(2, pixels.Color(0, 0, 255));
  pixels.setPixelColor(3, pixels.Color(0, 0, 255));
  pixels.show();
}
```

Physical Maze – Mort

Mort's task is to solve a physical maze through the use of the ultrasonic sensors as described in the hardware part. It should be able to navigate from point a to point b without getting stuck.

The code can be found on: https://github.com/Lennon-Tijmes/Los-Perdidos/blob/Physical-Maze/MORT_PANIEK/MORT_PANIEK.ino

Here is the code:



```
// ██╔══██╗██╔══╝  ██║     ██╔══██║  ╚██╔╝  ██╔══██╗██║   ██║
██║
// ██║  ██║███████╗███████╗██║  ██║   ██║   ██████╔╝╚██████╔╝
██║
// ╚═╝  ╚═╝╚══════╝╚══════╝╚═╝  ╚═╝   ╚═╝   ╚═════╝  ╚═════╝
╚═╝
//                  ███╗   ███╗ ██████╗ ██████╗ ████████╗
//                  ████╗ ████║██╔═══██╗██╔══██╗╚══██╔══╝
//                  ██╔████╔██║██║   ██║██████╔╝   ██║
//                  ██║╚██╔╝██║██║   ██║██╔══██╗   ██║
//                  ██║ ╚═╝ ██║╚██████╔╝██║  ██║   ██║
//                  ╚═╝     ╚═╝ ╚═════╝ ╚═╝  ╚═╝   ╚═╝



// #define DEBUG 1                    // Enables all debug
messages
// #define START_AT_MAZE 1            // Is robot starting inside
the maze without barrel?
#define WAIT_FOR_SIGNAL 1             // Wait for start signal? if
not, start driving immediately
#define NEW_INTERRUPTS_HANDLING 1     // Instead of disabling all
interrupts during rotation sensor function, just disable one pin
//#define NEW_GRIPPER_HANDLING 1      // Sometimes the gripper
gets forced open when robot drives into a wall, this keeps it
closed (DOESN'T WORK)
//#define ENABLE_AUTOMATIC_LIGHTS 1 // Automatically update
lights depending on what the robot is doing
#define STARTUP_DELAY 2000            // Wait a bit to let sonars
and other things get warmed up



// Pins
const unsigned char LINE_SENSOR_PINS[] = { A2, A3, A4, A5, A6,
A7 }; // Pin numbers for line sensor (pin 16 - pin 21)
#define SONAR_LEFT_TRIG_PIN         5  // Left  sonar TRIG pin
#define SONAR_LEFT_ECHO_PIN         6  // Left  sonar ECHO pin
#define SONAR_FRONT_TRIG_PIN        12 // Front sonar TRIG pin
#define SONAR_FRONT_ECHO_PIN        13 // Front sonar ECHO pin
#define SONAR_RIGHT_TRIG_PIN        A1 // Right sonar TRIG pin
```

```
(pin 15)
#define SONAR_RIGHT_ECHO_PIN      A0 // Right sonar ECHO pin
(pin 14)
#define ROTATION_SENSOR_LEFT_PIN  2  // Left  wheel rotation
sensor interrupt pin
#define ROTATION_SENSOR_RIGHT_PIN 3  // Right wheel rotation
sensor interrupt pin
#define MOTOR_LEFT_POWER_PIN      9  // Left  motor power pin
#define MOTOR_RIGHT_POWER_PIN     10 // Right motor power pin
#define MOTOR_LEFT_MODE_PIN       7  // Left  motor mode pin
#define MOTOR_RIGHT_MODE_PIN      8  // Right motor mode pin
#define GRIPPER_PIN               4  // Gripper control pin
#define PIXELS_PIN                11 // Pixels control pin

// Phase
#define PHASE_STARTUP_WAIT      100 // Wait a few seconds to let
sonars and other things warm up
#define PHASE_WAIT_FOR_SIGNAL   101 // Wait for start signal
#define PHASE_DRIVE_TO_SQUARE   102 // Drive forward till black
square, grab the pin and turn left
#define PHASE_FOLLOW_LINE_START 103 // Follow the line until
inside of the maze
#define PHASE_DRIVE_MAZE        104 // Solve the maze
#define PHASE_FOLLOW_LINE_END   105 // Follow the finishing line
and put pin inside black square
#define PHASE_FINISH            106 // Stop driving
unsigned char programPhase;         // Current phase of the
program
unsigned long programPhaseStartTime;// Current phase start time
unsigned long programStartTime;     // Program start time

// Motors
#define STOP              200 // Stop both motors
#define FORWARDS          201 // Drive forwards, both motors
full speed forwards
#define BACKWARDS         202 // Drive backwards, both motors
full speed backwards
#define SMOOTH_LEFT       203 // Turn left slowly, right motor
full speed, left motor 75%
#define SMOOTH_RIGHT      204 // Turn right slowly, left motor
full speed, right motor 75%
```

```
#define LEFT              205 // Turn left, right motor full
speed, left motor half speed
#define RIGHT             206 // Turn right, left motor full
speed, right motor half speed
#define LEFT_BACK         207 // Turn left backwards, right
motor full speed backwards, left motor half speed backwards
#define RIGHT_BACK        208 // Turn right backwards, left
motor full speed backwards, right motor half speed backwards
#define ROTATE_LEFT       209 // Rotate in place anti-cloclwise,
right motor full speed, left motor full speed backwards
#define ROTATE_RIGHT      210 // Rotate in place      cloclwise,
left motor full speed, right motor full speed backwards
#define ADJUST_LEFT       211 // for line following
#define ADJUST_RIGHT      212 // for line following
#define ADJUST_LEFT_HARD  213 // for line following
#define ADJUST_RIGHT_HARD 214 // for line following
// Left and right motors are slightly different so to drive
straight we need to set them at slightly different speeds
// This also changes randomly every robot restart, so we set the
difference to 5 to drive most straight on average
int leftSpeed = 250;  // Left motor speed
int rightSpeed = 255; // Right motor speed

// Rotation sensor
unsigned int leftRotationTicks = 0;  // Left  wheel "rotation
ticks" that rotation sensor counted, there's 20 in wheel's full
rotation
unsigned int rightRotationTicks = 0; // Right wheel "rotation
ticks" that rotation sensor counted, there's 20 in wheel's full
rotation

// Line sensor
// This could be one variable but let's not fix what's not
broken
unsigned int colorBlack = 801; // Line sensor value above this
indicates black
unsigned int colorWhite = 799; // Line sensor value above this
indicates white
bool allBlack = false; // True if all line sensor's values are
above colorBlack
bool allWhite = false; // True if all line sensor's values are
```

```
below colorWhite

// Signal receiver
#define SLAVE_ID 3 // Slave ID for master/slave communication

// NeoPixel
#include <Adafruit_NeoPixel.h> // Load NeoPixel library
#define NUM_PIXELS 4           // Number of LEDs
Adafruit_NeoPixel pixels(NUM_PIXELS, PIXELS_PIN, NEO_RGB +
NEO_KHZ800); // Initialize NeoPixel

#define OFF    300 // Easy to use LED color
#define WHITE  301 // Easy to use LED color
#define RED    302 // Easy to use LED color
#define ORANGE 303 // Easy to use LED color
#define YELLOW 304 // Easy to use LED color
#define GREEN  305 // Easy to use LED color
#define BLUE   306 // Easy to use LED color
#define PURPLE 307 // Easy to use LED color

#define LED_BACK_LEFT   0 // Back left LED id
#define LED_BACK_RIGHT  1 // Back right LED id
#define LED_FRONT_RIGHT 2 // Front left LED id
#define LED_FRONT_LEFT  3 // Front right LED id

// Function to set up pins, interrupts, variables and everything
else needed
void setup()
{
  Serial.begin(9600);                                  // Begin the
serial monitor
  pixels.begin();                                      //
Initialize the pixels thing
  Serial.println("Slave started");                     // Send
message to base
  pinMode(SONAR_LEFT_TRIG_PIN, OUTPUT);                //
Initialize the left  sonar trig pin as output
  pinMode(SONAR_LEFT_ECHO_PIN, INPUT);                 //
Initialize the left  sonar echo pin as input
  pinMode(SONAR_FRONT_TRIG_PIN, OUTPUT);               //
Initialize the front sonar trig pin as output
```

```
  pinMode(SONAR_FRONT_ECHO_PIN, INPUT);              //
Initialize the front sonar echo pin as input
  pinMode(SONAR_RIGHT_TRIG_PIN, OUTPUT);             //
Initialize the right sonar trig pin as output
  pinMode(SONAR_RIGHT_ECHO_PIN, INPUT);              //
Initialize the right sonar echo pin as input
  pinMode(ROTATION_SENSOR_LEFT_PIN, INPUT_PULLUP);   //
Initialize the left  rotation sensor as pullup input
  pinMode(ROTATION_SENSOR_RIGHT_PIN, INPUT_PULLUP); //
Initialize the right rotation sensor as pullup input
  pinMode(MOTOR_LEFT_POWER_PIN, OUTPUT);             //
Initialize the left  motor power pin as output
  pinMode(MOTOR_RIGHT_POWER_PIN, OUTPUT);            //
Initialize the right motor power pin as output
  pinMode(MOTOR_LEFT_MODE_PIN, OUTPUT);              //
Initialize the left  motor mode  pin  as output
  pinMode(MOTOR_RIGHT_MODE_PIN, OUTPUT);             //
Initialize the right motor mode  pin  as output
  pinMode(GRIPPER_PIN, OUTPUT);                      //
Initialize the gripper pin as output
  for (char i = 0; i < 6; i++)                       //
Initialize the line sensor pins as input
  {
    pinMode(LINE_SENSOR_PINS[i], INPUT);
  }

  // Attach interrupts for both rotation sensors

attachInterrupt(digitalPinToInterrupt(ROTATION_SENSOR_LEFT_PIN),
countRotationsLeft, RISING);

attachInterrupt(digitalPinToInterrupt(ROTATION_SENSOR_RIGHT_PIN)
, countRotationsRight, RISING);

  programStartTime = millis();
  changePhase(PHASE_STARTUP_WAIT); // Start at phase
PHASE_STARTUP_WAIT
  lights(WHITE); // Set lights to white
}

// The program's main loop, run many times per millisecond
```

```
void loop()
{
  updateSonar();      // Tick the sonar managing function,
letting up update sonar readings whenever it decides to
  updateLineSensor(); // Update line sensor readings

  // Since the robot has multiple distinct tasks to do, they're
split into "phases"
  // Depending on current phase saved in programPhase variable,
appriopriate function is ticked
  switch (programPhase)
  {
    case PHASE_STARTUP_WAIT: phase_startupWait(); break;
    case PHASE_WAIT_FOR_SIGNAL: phase_waitForSignal(); break;
    case PHASE_DRIVE_TO_SQUARE: phase_driveToSquare(); break;
    case PHASE_FOLLOW_LINE_START: phase_followLineStart();
break;
    case PHASE_DRIVE_MAZE: phase_driveMaze(); break;
    case PHASE_FOLLOW_LINE_END: phase_followLineEnd(); break;
    case PHASE_FINISH: phase_finish(); break;
  }

  // Sometimes the gripper gets forced open when robot drives
into a wall, this keeps it closed
  // DOESN'T WORK - the gripper requires precise delay between
impulses and this method cannot guarantee them
  #ifdef NEW_GRIPPER_HANDLING
    updateGripper();
  #endif
  // Automatically update lights depending on what the robot is
doing
  // It seems to slow down the main loop quite a bit, making
robot perform poorly
  #ifdef ENABLE_AUTOMATIC_LIGHTS
    updateLights();
  #endif
}

// Set program phase and note down phase start time which some
phases use
void changePhase(unsigned char phase)
```

```
{
  programPhase = phase;
  programPhaseStartTime = millis();
}



//////////////////////
// ROTATION SENSOR //
//////////////////////

// Debounce time for more accurate rotation sensor reading, at
max robot speed sensor should tick every 40ms
#define DEBOUNCE_TIME_MS 30;

// Handle the interrupts of the rotation sensor for the left
wheel
void countRotationsLeft()
{
  static unsigned long timer;
  #ifdef NEW_INTERRUPTS_HANDLING

detachInterrupt(digitalPinToInterrupt(ROTATION_SENSOR_LEFT_PIN))
;
  #else
    noInterrupts();
  #endif

  if (millis() > timer)
  {
    leftRotationTicks++;
    timer = millis() + DEBOUNCE_TIME_MS;
  }

  #ifdef NEW_INTERRUPTS_HANDLING

attachInterrupt(digitalPinToInterrupt(ROTATION_SENSOR_LEFT_PIN),
countRotationsLeft, RISING);
  #else
    interrupts();
  #endif
```

```
}

// Handle the interrupts of the rotation sensor for the right
wheel
void countRotationsRight()
{
  static unsigned long timer;
  #ifdef NEW_INTERRUPTS_HANDLING

detachInterrupt(digitalPinToInterrupt(ROTATION_SENSOR_RIGHT_PIN)
);
  #else
    noInterrupts();
  #endif

  if (millis() > timer)
  {
    rightRotationTicks++;
    timer = millis() + DEBOUNCE_TIME_MS;
  }

  #ifdef NEW_INTERRUPTS_HANDLING

attachInterrupt(digitalPinToInterrupt(ROTATION_SENSOR_RIGHT_PIN)
, countRotationsRight, RISING);
  #else
    interrupts();
  #endif
}



//////////////////
// LINE SENSOR //
//////////////////

int lineSensorValue[6] = { 0, 0, 0, 0, 0, 0 };

// Read all the line sensor pins
// Could be optimized since lineSensorValue is unused but there
was no need
```

```
void updateLineSensor()
{
  for (unsigned char i = 0; i < 6; i++)
  {
    lineSensorValue[i] = analogRead(LINE_SENSOR_PINS[i]);
  }

  // True if all the line sensor bits are looking at black
  allBlack =    (lineSensorValue[0] >= colorBlack)
             && (lineSensorValue[1] >= colorBlack)
             && (lineSensorValue[2] >= colorBlack)
             && (lineSensorValue[3] >= colorBlack)
             && (lineSensorValue[4] >= colorBlack)
             && (lineSensorValue[5] >= colorBlack);

  // True if all the line sensor bits are looking at white
  allWhite =    (lineSensorValue[0] <= colorWhite)
             && (lineSensorValue[1] <= colorWhite)
             && (lineSensorValue[2] <= colorWhite)
             && (lineSensorValue[3] <= colorWhite)
             && (lineSensorValue[4] <= colorWhite)
             && (lineSensorValue[5] <= colorWhite);
}



////////////
// SONAR //
////////////

// The HC-SR04 manual recommends sending a 10 microsecond pulse
to TRIG pin
// Used in delayMicroseconds function - 10 microseconds is a
short enough delay to not interfere with the rest of the program
// Using delayMicroseconds ensures that the sonar works
correctly
#define SONAR_SIGNAL_DURATION_US 10

// Minimum delay before switching to next sonar, so signal from
previous sonar doesn't interfere
#define SONAR_DELAY_MS 10
```

```
// We assume the maze is 7x7, which means the longest distance
to measure should be 7*30cm = 210cm
// Speed of sound is around 35cm/ms=210cm/6ms, so the sound
signal should take at most 12ms to come back to the robot
// Accounting for some inaccuracies it's safe to set the timeout
to 15ms
#define SONAR_RECEIVER_TIMEOUT_MS 15

// Sonar phases
#define SONAR_LEFT_SEND_SIGNAL  0
#define SONAR_LEFT_READ_SIGNAL  1
#define SONAR_FRONT_SEND_SIGNAL 2
#define SONAR_FRONT_READ_SIGNAL 3
#define SONAR_RIGHT_SEND_SIGNAL 4
#define SONAR_RIGHT_READ_SIGNAL 5
#define SONAR_UPDATE_DISTANCES  6

// Microseconds to cm converter using the speed of sound
// 340m/s = 34000cm/s = 34cm/ms = 0.034cm/us, sound travels to
object and back so divide by 2 for distance to object
const double microsecondsToCentimeters = 0.017;

#define SONAR_TOO_FAR 1023    // distance reading for when sonar
returns value above 210cm
#define SONAR_NO_READING 1024 // distance value for sonar timing
out before signal can come back
double distanceLeft = SONAR_NO_READING;  // distance in cm from
left sonar
double distanceFront = SONAR_NO_READING; // distance in cm from
front sonar
double distanceRight = SONAR_NO_READING; // distance in cm from
right sonar
bool sonarsStuck = false; // True when sonar readings haven't
changed since last read, used for detecting whether robot is
stuck

// Sonar managing function - uses sonars to gauge distance to a
wall on left, front and right
// Rather than waiting for sonar's sound signal to come back,
//   the function quits and checks for result on next call -
```

```
should be called every program loop
// Uses one sonar at a time to avoid polluting the reading from
another sonars' signal
// After all 3 sonars are read, updates global distance
variables
void updateSonar()
{
  static unsigned char sonarPhase = 0;          // Sonar
manager's phase, makes sure only one sonar is used at a time
  static unsigned long sonarLastActionTime = 0; // Timestamp of
when sonar sent or received a signal, depending on phase. Used
for ensuring delay between actions that need it
  static unsigned long sonarSignalDuration = 0; // Variable to
save sonar's reading in microseconds
  static double currentDistanceLeft = SONAR_NO_READING;  //
Current sonar reading, befoe it's put into a global variable
  static double currentDistanceFront = SONAR_NO_READING; //
Current sonar reading, befoe it's put into a global variable
  static double currentDistanceRight = SONAR_NO_READING; //
Current sonar reading, befoe it's put into a global variable

  switch (sonarPhase)
  {
    // Left sonar

    // Send pulse on left sonar
    case SONAR_LEFT_SEND_SIGNAL:
      if (millis() - sonarLastActionTime > SONAR_DELAY_MS)
      {
        digitalWrite(SONAR_LEFT_TRIG_PIN, HIGH); // Start
sending pulse to sonar
        delayMicroseconds(SONAR_SIGNAL_DURATION_US); // Delay
function used to ensure sonar works correctly, 10us delay
doesn't affect the robot's performance
        digitalWrite(SONAR_LEFT_TRIG_PIN, LOW); // Stop sending
pulse to sonar
        sonarLastActionTime = millis();
        sonarPhase = SONAR_LEFT_READ_SIGNAL;
      }
      break;
```

```
    // Get HIGH state duration from ECHO pin, or move to next
phase if no signal after SONAR_RECEIVER_TIMEOUT_MS milliseconds
    case SONAR_LEFT_READ_SIGNAL:
      sonarSignalDuration = pulseIn(SONAR_LEFT_ECHO_PIN, HIGH);
      if (sonarSignalDuration > 0) // Duration of HIGH pulse is
0 until sonar receives its sound signal
      {
        // if more than 210cm mark reading as too far, otherwise
we save the reading
        if (sonarSignalDuration > 12353)
        {
          currentDistanceLeft = SONAR_TOO_FAR;
        }
        // HC-SR04 sonar returns time between signal being sent
and received in microseconds, so for easier use we convert it to
cm
        else
        {
          currentDistanceLeft = (double)sonarSignalDuration *
microsecondsToCentimeters;
        }

        sonarSignalDuration = 0; // reset variable just in case,
even if doesn't seem necessary
        sonarLastActionTime = millis();
        sonarPhase = SONAR_FRONT_SEND_SIGNAL;
      }
      else if (millis() - sonarLastActionTime >
SONAR_RECEIVER_TIMEOUT_MS) // if we didn't receive signal within
SONAR_RECEIVER_TIMEOUT_MS ms, mark as "no reading" and move on
      {
        currentDistanceLeft = SONAR_NO_READING;
        sonarLastActionTime = millis();
        sonarPhase = SONAR_FRONT_SEND_SIGNAL;
      }
      break;

    // Front sonar

    // Send pulse on front sonar
    case SONAR_FRONT_SEND_SIGNAL:
```

```
      if (millis() - sonarLastActionTime > SONAR_DELAY_MS)
      {
        digitalWrite(SONAR_FRONT_TRIG_PIN, HIGH);
        delayMicroseconds(SONAR_SIGNAL_DURATION_US);
        digitalWrite(SONAR_FRONT_TRIG_PIN, LOW);
        sonarLastActionTime = millis();
        sonarPhase = SONAR_FRONT_READ_SIGNAL;
      }
      break;

    // Get HIGH state duration from ECHO pin, or move to next
phase if no signal after SONAR_RECEIVER_TIMEOUT_MS milliseconds
    case SONAR_FRONT_READ_SIGNAL:
      sonarSignalDuration = pulseIn(SONAR_FRONT_ECHO_PIN, HIGH);
      if (sonarSignalDuration > 0) // Duration of HIGH pulse is
0 until sonar receives its sound signal
      {
        // if more than 210cm mark reading as too far, otherwise
we save the reading
        if (sonarSignalDuration > 12353)
        {
          currentDistanceFront = SONAR_TOO_FAR;
        }
        // HC-SR04 sonar returns time between signal being sent
and received in microseconds, so for easier use we convert it to
cm
        else
        {
          currentDistanceFront = (double)sonarSignalDuration *
microsecondsToCentimeters;
        }

        sonarSignalDuration = 0; // reset variable just in case,
even if doesn't seem necessary
        sonarLastActionTime = millis();
        sonarPhase = SONAR_RIGHT_SEND_SIGNAL;
      }
      else if (millis() - sonarLastActionTime >
SONAR_RECEIVER_TIMEOUT_MS) // if we didn't receive signal within
SONAR_RECEIVER_TIMEOUT_MS ms, mark as "no reading" and move on
      {
```

```
        currentDistanceFront = SONAR_NO_READING;
        sonarLastActionTime = millis();
        sonarPhase = SONAR_RIGHT_SEND_SIGNAL;
      }
      break;

    // Right sonar

    // Send pulse on right sonar
    case SONAR_RIGHT_SEND_SIGNAL:
      if (millis() - sonarLastActionTime > SONAR_DELAY_MS)
      {
        digitalWrite(SONAR_RIGHT_TRIG_PIN, HIGH);
        delayMicroseconds(SONAR_SIGNAL_DURATION_US);
        digitalWrite(SONAR_RIGHT_TRIG_PIN, LOW);
        sonarLastActionTime = millis();
        sonarPhase = SONAR_RIGHT_READ_SIGNAL;
      }
      break;

    // Get HIGH state duration from ECHO pin, or move to next
phase if no signal after SONAR_RECEIVER_TIMEOUT_MS milliseconds
    case SONAR_RIGHT_READ_SIGNAL:
      sonarSignalDuration = pulseIn(SONAR_RIGHT_ECHO_PIN, HIGH);
      if (sonarSignalDuration > 0) // Duration of HIGH pulse is
0 until sonar receives its sound signal
      {
        // if more than 210cm mark reading as too far, otherwise
we save the reading
        if (sonarSignalDuration > 12353)
        {
          currentDistanceRight = SONAR_TOO_FAR;
        }
        // HC-SR04 sonar returns time between signal being sent
and received in microseconds, so for easier use we convert it to
cm
        else
        {
          currentDistanceRight = (double)sonarSignalDuration *
microsecondsToCentimeters;
        }
```

```
        sonarSignalDuration = 0; // reset variable just in case,
even if doesn't seem necessary
        sonarLastActionTime = millis();
        sonarPhase = SONAR_UPDATE_DISTANCES;
      }
      else if (millis() - sonarLastActionTime >
SONAR_RECEIVER_TIMEOUT_MS) // if we didn't receive signal within
SONAR_RECEIVER_TIMEOUT_MS ms, mark as "no reading" and move on
      {
        currentDistanceRight = SONAR_NO_READING;
        sonarLastActionTime = millis();
        sonarPhase = SONAR_UPDATE_DISTANCES;
      }
      break;

    case SONAR_UPDATE_DISTANCES:
      // If none of the sonar readings changed since last read,
we mark robot as potentially stuck
      sonarsStuck = distanceLeft == currentDistanceLeft &&
distanceFront == currentDistanceFront && distanceRight ==
currentDistanceRight;

      distanceLeft = currentDistanceLeft;
      distanceFront = currentDistanceFront;
      distanceRight = currentDistanceRight;
      sonarPhase = SONAR_LEFT_SEND_SIGNAL;
      break;
  }
}



//////////
// MOTOR //
//////////

// The motors sometimes don't move at all when given speed below
150
// We will let wheels start spinning with max motor speed, then
after WHEEL_INERTIA_DELAY_MS change to desired speed
```

```
#define WHEEL_INERTIA_DELAY_MS 2

#define FORWARDS_MODE LOW
#define BACKWARDS_MODE HIGH

// The motors require very weird input - seems to be a 9-bit
number split over 2 pins
// This function makes using motors much simpler, now speed is
between -255 and 255
void setMotors(int leftMotorSpeed, int rightMotorSpeed)
{
  if (leftMotorSpeed >= 0)
  {
    digitalWrite(MOTOR_LEFT_MODE_PIN, FORWARDS_MODE);
    analogWrite(MOTOR_LEFT_POWER_PIN, 255);
    delay(WHEEL_INERTIA_DELAY_MS);
    analogWrite(MOTOR_LEFT_POWER_PIN, leftMotorSpeed);
  }
  else /* leftMotorSpeed < 0 */
  {
    digitalWrite(MOTOR_LEFT_MODE_PIN, BACKWARDS_MODE);
    analogWrite(MOTOR_LEFT_POWER_PIN, 0);
    delay(WHEEL_INERTIA_DELAY_MS);
    analogWrite(MOTOR_LEFT_POWER_PIN, 255 + leftMotorSpeed);
  }

  if (rightMotorSpeed >= 0)
  {
    digitalWrite(MOTOR_RIGHT_MODE_PIN, FORWARDS_MODE);
    analogWrite(MOTOR_RIGHT_POWER_PIN, 255);
    delay(WHEEL_INERTIA_DELAY_MS);
    analogWrite(MOTOR_RIGHT_POWER_PIN, rightMotorSpeed);
  }
  else /* rightMotorSpeed < 0 */
  {
    digitalWrite(MOTOR_RIGHT_MODE_PIN, BACKWARDS_MODE);
    analogWrite(MOTOR_RIGHT_POWER_PIN, 0);
    delay(WHEEL_INERTIA_DELAY_MS);
    analogWrite(MOTOR_RIGHT_POWER_PIN, 255 + rightMotorSpeed);
  }
}
```

```
// Using drive(FORWARDS) looks much nicer than setMotors(250,
255), so this function exists
unsigned char lastMode = 255;
void drive(unsigned char mode)
{
  if (mode == lastMode) return;

  switch (mode)
  {
    case STOP: setMotors(0, 0); break;
    case FORWARDS: setMotors(leftSpeed, rightSpeed); break;
    case BACKWARDS: setMotors(-leftSpeed, -rightSpeed); break;

    case SMOOTH_LEFT: setMotors(150, rightSpeed); break;
    case SMOOTH_RIGHT: setMotors(leftSpeed, 150); break;
    case LEFT: setMotors(100, rightSpeed); break;
    case RIGHT: setMotors(leftSpeed, 100); break;
    case LEFT_BACK: setMotors(-100, -rightSpeed); break;
    case RIGHT_BACK: setMotors(-leftSpeed, -100); break;

    case ROTATE_LEFT: setMotors(-leftSpeed, rightSpeed); break;
    case ROTATE_RIGHT: setMotors(leftSpeed, -rightSpeed); break;

    case ADJUST_LEFT: setMotors(160, 240); break;
    case ADJUST_RIGHT: setMotors(240, 165); break;
    case ADJUST_LEFT_HARD: setMotors(-190, 240); break;
    case ADJUST_RIGHT_HARD: setMotors(240, -190); break;
  }

  lastMode = mode;
}



//////////////
// GRIPPER //
//////////////

#define GRIPPER_OPEN 1600   // Pulse duration for gripper which
it interprets as opening
```

```
#define GRIPPER_CLOSED 1010 // Pulse duration for gripper which
it interprets as closing
unsigned long gripperState = GRIPPER_CLOSED;

#ifdef NEW_GRIPPER_HANDLING
  void setGripper(unsigned long openClose)
  {
    gripperState = openClose;
  }

  // Keep gripper in its desired state by constantly sending
signals
  // DOESN'T WORK - adds too much delay to main loop
  void updateGripper()
  {
    static unsigned long timer = micros();

    if (micros() - timer >= gripperState)
    {
      digitalWrite(GRIPPER_PIN, LOW);
      digitalWrite(GRIPPER_PIN, HIGH);
      timer = micros();
    }
  }
#else
  // Immediately sets the gripper position to the given pulse
  void setGripper(unsigned int pulse)
  {
    for (unsigned char i = 0; i < 8; i++)
    {
      digitalWrite(GRIPPER_PIN, HIGH);
      delayMicroseconds(pulse);
      digitalWrite(GRIPPER_PIN, LOW);
    }
  }
#endif
```

```
////////////////////////
// PHASE_STARTUP_WAIT //
////////////////////////

void phase_startupWait()
{
  if (startupDelay())
  {
    #ifdef START_AT_MAZE
      changePhase(PHASE_DRIVE_MAZE);
    #else
      changePhase(PHASE_WAIT_FOR_SIGNAL);
    #endif
  }
}

bool startupDelay()
{
  return millis() - programStartTime > STARTUP_DELAY;
}



////////////////////////////
// PHASE_WAIT_FOR_SIGNAL //
////////////////////////////

void phase_waitForSignal()
{
  #ifndef WAIT_FOR_SIGNAL
    changePhase(PHASE_DRIVE_TO_SQUARE);
    return;
  #endif

  if (hasReceivedSignal())
  {
    changePhase(PHASE_DRIVE_TO_SQUARE);
  }
}
```

```
bool hasReceivedSignal()
{
  if (Serial.available())
  {
    String message = Serial.readStringUntil('\n');
    Serial.print("Received: ");
    Serial.println(message);

    if (message.length() >= 2 && message[0] == SLAVE_ID + '0' &&
message[1] == '?')
    {
      Serial.print("Responding to Master: Slave ");
      Serial.print(SLAVE_ID);
      Serial.println(" Response");
      return true;
    }
  }
  else
  {
    // Maybe flash lights to say that we cannot connect to base
  }

  return false;
}



//////////////////////////
// PHASE_DRIVE_TO_SQUARE //
//////////////////////////

// Drive from starting spot to black square, pick up pin then go
left
// Here using delays is fine because no other code needs to run
at this time
void phase_driveToSquare()
{
  // Drive to the square
  while (!allBlack)
  {
    updateLineSensor();
```

```
    drive(FORWARDS);
    delay(130);
  }

  updateLineSensor();

  // When on the square grab pin and go left
  if (allBlack)
  {
    drive(FORWARDS);
    delay(100);
    setGripper(GRIPPER_CLOSED);
    drive(ROTATE_LEFT);
    delay(460);
    drive(FORWARDS);
    delay(500);
    changePhase(PHASE_FOLLOW_LINE_START);
  }
}



////////////////////////////
// PHASE_FOLLOW_LINE_START //
////////////////////////////

// Follow the black line until we're inside the maze or until 3
seconds pass
void phase_followLineStart()
{
  static unsigned char lastDirection = STOP;

  updateLineSensor();
  bool turnLeft =   (lineSensorValue[0] >= colorBlack) ||
(lineSensorValue[1] >= colorBlack);
  bool goForwards = (lineSensorValue[2] >= colorBlack) ||
(lineSensorValue[3] >= colorBlack);
  bool turnRight =  (lineSensorValue[4] >= colorBlack) ||
(lineSensorValue[5] >= colorBlack);

  if (allWhite)
```

```
  {
    if (lastDirection == LEFT)
    {
      drive(ADJUST_LEFT);
    }
    else
    {
      drive(ADJUST_RIGHT);
    }
  }
  else
  {
    if (goForwards)
    {
      drive(FORWARDS);
    }
    else if (turnRight)
    {
      drive(ADJUST_RIGHT);
      lastDirection = RIGHT;
    }
    else if (turnLeft)
    {
      drive(ADJUST_LEFT);
      lastDirection = LEFT;
    }
  }

  updateSonar();

  if (allWhite && distanceLeft < 30 && distanceFront < 40 &&
distanceRight < 30)
  {
    changePhase(PHASE_DRIVE_MAZE);
  }
  if (millis() - programPhaseStartTime > 3000)
  {
    changePhase(PHASE_DRIVE_MAZE);
  }
}
```

```
//////////////////////
// PHASE_DRIVE_MAZE //
//////////////////////

// If we start checking for black line immeditely after entering
the maze, we'll find the start line and think we're at the
finish
#define CHECK_FINISH_LINE_DELAY_MS 20000

unsigned long activeTaskTimer = millis();

void phase_driveMaze()
{
  static unsigned long stuckTimer = millis(); // Used for
assuring delay between checking whether robot is stuck
  static unsigned int stuckRotations = 0;     // Keeps track of
wheel rotations to see whether robot is stuck
  static unsigned char stuckCounter = 0;      // Increases if
robot is stuck, at certain value we unstuck the robot
  static unsigned char previousTask = STOP;   // Keeps track of
previous task for stuck checks
  previousTask = lastMode;

  if (sonarsStuck)
  {
    doTask(BACKWARDS, 100);
    sonarsStuck = false;
  }

  // If a task is ongoing, let it continue
  if (millis() < activeTaskTimer)
  {
    return;
  }

  // If a sonar failed to read distance, wait until it's fixed
  if (distanceLeft >= SONAR_NO_READING || distanceFront >=
SONAR_NO_READING || distanceRight >= SONAR_NO_READING)
  {
```

```
    doTask(STOP, 50);
    lights(PURPLE);
    return;
  }

  // If we're outside the maze, pause - we don't know if we're
at the beginning or end
  if (distanceLeft == SONAR_TOO_FAR && distanceRight ==
SONAR_TOO_FAR)
  {
    doTask(STOP, 100);
    hazardLights(PURPLE);
    return;
  }

  // Maze AI - since the maze has no loops, we can just follow
left wall until finish
  if (distanceFront < 10) // If we're too close to a wall at the
front - back off, unless we're in a dead end - then turn around
  {
    if (distanceLeft < 15 && distanceRight < 15)
    {
      turnAround();
    }
    else
    {
      doTask(BACKWARDS, 150);
    }
  }
  else if (distanceLeft > 25) // If no wall on the left, go left
  {
    doTask(LEFT, 500); // After some testing, turning for 500ms
brought much better results than shorter turns
  }
  else if (distanceFront > 25) // If wall on the left, but no
wall in front, go forward
  {
    drive(FORWARDS);
  }
  else if (distanceRight > 20) // If walls on left and front,
but no wall on right, go right
```

```
  {
    doTask(RIGHT, 500); // After some testing, turning for 500ms
brought much better results than shorter turns
  }
  else if (distanceLeft < 4) // If too close to a wall, turn
away from it
  {
    doTask(RIGHT, 100);
  }
  else if (distanceRight < 4) // If too close to a wall, turn
away from it
  {
    doTask(LEFT, 100);
  }
  // Just noticed these two below don't make much sense
  // At this point left distance is 4-25, front 10-25, right 4-
20 so we should just turn around...
  else if (distanceFront > 10)
  {
    drive(FORWARDS);
  }
  else
  {
    turnAround();
  }

  // If wheels are not spinning, go backwards
  if (millis() - stuckTimer > 200)
  {
    if (leftRotationTicks + rightRotationTicks <= stuckRotations
+ 5) // If wheel ticks changed by less than 5 in 200ms, go
backwards to get unstuck
    {
      doTask(BACKWARDS, 200);
    }
    else // Otherwise update the rotation counter
    {
      stuckRotations = leftRotationTicks + rightRotationTicks;
    }

    stuckTimer = millis();
```

```
  }

  // If robot is stuck in forwards-backwards loop, turn left a
little
  if (lastMode != previousTask)
  {
    if (lastMode == FORWARDS && previousTask == BACKWARDS ||
lastMode == BACKWARDS && previousTask == FORWARDS)
    {
      stuckCounter++;
    }
    else
    {
      stuckCounter = 0;
    }
  }
  if (stuckCounter > 6) // If robot went between FORWARDS and
BACKWARDS 7+ times, turn left to get unstuck
  {
    doTask(LEFT, 100);
  }
}

// Drive for a set duration of time
void doTask(unsigned char task, unsigned long duration)
{
  drive(task);
  activeTaskTimer = millis() + duration;
}

// Choose best way to turn around
void turnAround()
{
  if (distanceRight > distanceLeft)
  {
    if (distanceLeft > 5)
    {
      doTask(ROTATE_RIGHT, 750);
    }
    else
    {
```

```
      doTask(ROTATE_RIGHT, 750); // This used to be driving
back-left, then forwards-right but no time to recreate it
    }
  }
  else
  {
    if (distanceRight > 5)
    {
      doTask(ROTATE_LEFT, 750);
    }
    else
    {
      doTask(ROTATE_LEFT, 750); // This used to be driving back-
right, then forwards-left but no time to recreate it
    }
  }
}

// Returns whether the two numbers are no more than 'diff'
apart, currently unused
bool numbersAreClose(double a, double b, double diff)
{
  return abs(a - b) < diff;
}



//////////////////////////
// PHASE_FOLLOW_LINE_END //
//////////////////////////

void phase_followLineEnd()
{
  static unsigned long timer = millis();
  static unsigned char lastDirection = LEFT;

  updateLineSensor();
  bool turnLeft =  (lineSensorValue[0] > colorBlack) ||
(lineSensorValue[1] > colorBlack);
  bool forwards =  (lineSensorValue[2] > colorBlack) ||
(lineSensorValue[3] > colorBlack);
```

```
  bool turnRight = (lineSensorValue[4] > colorBlack) ||
(lineSensorValue[5] > colorBlack);

  if (millis() - timer > 500)
  {
    if (!allBlack)
    {
      if (!allWhite)
      {
        if (forwards)
        {
          drive(FORWARDS);
        }
        else if (turnRight)
        {
          drive(ADJUST_RIGHT);
          lastDirection = RIGHT;
        }
        else if (turnLeft)
        {
          drive(ADJUST_LEFT);
          lastDirection = LEFT;
        }
      }
      else
      {
        if (lastDirection == RIGHT)
        {
          drive(ADJUST_RIGHT_HARD);
        }
        else
        {
          drive(ADJUST_LEFT_HARD);
        }
      }
    }
    else
    {
      setGripper(GRIPPER_OPEN);
      drive(STOP);
      changePhase(PHASE_FINISH);
```

```
    }

    timer = millis();
  }
}



//////////////////
// PHASE_FINISH //
//////////////////

void phase_finish()
{
  discoLights();
}



///////////
// DEBUG //
///////////

void printDebug(String message)
{
  #ifdef DEBUG
    Serial.println(String("[") + millis() + "] " + message);
  #endif
}



///////////////
// NEOPIXELS //
///////////////

#define BLINK_DELAY_MS 250

// Automatically update lights depending on what the robot is
doing
// But it seems to slow down main loop making maze driving bad
```

```
:(
void updateLights()
{
  switch (lastMode)
  {
    case STOP: brakeLights(); break;
    case FORWARDS: lights(WHITE); break;
    case BACKWARDS: backwardsLights(RED); break;

    case LEFT: leftBlinker(ORANGE); break;
    case RIGHT: rightBlinker(ORANGE); break;
    case LEFT_BACK: backLeftBlinker(ORANGE); break;
    case RIGHT_BACK: backRightBlinker(ORANGE); break;

    case ROTATE_LEFT: leftBlinker(YELLOW); break;
    case ROTATE_RIGHT: rightBlinker(YELLOW); break;

    case ADJUST_LEFT: light(LED_FRONT_LEFT, ORANGE); break;
    case ADJUST_RIGHT: light(LED_FRONT_RIGHT, ORANGE); break;
    case ADJUST_LEFT_HARD: light(LED_FRONT_LEFT, RED); break;
    case ADJUST_RIGHT_HARD: light(LED_FRONT_RIGHT, RED); break;
  }
}

void lights(unsigned int color)
{
  switch (color)
  {
    case OFF: lights(0, 0, 0); break;
    case WHITE: lights(255, 255, 255); break;
    case RED: lights(255, 0, 0); break;
    case ORANGE: lights(0, 160, 16); break;
    case YELLOW: lights(0, 225, 32); break;
    case GREEN: lights(0, 255, 0); break;
    case BLUE: lights(0, 0, 255); break;
    case PURPLE: lights(160, 32, 255); break;
    default: lights(0, 0, 0);
  }
}

void light(unsigned int ledID, unsigned int color)
```

```
{
  switch (color)
  {
    case OFF: light(ledID, 0, 0, 0); break;
    case WHITE: light(ledID, 255, 255, 255); break;
    case RED: light(ledID, 255, 0, 0); break;
    case ORANGE: light(ledID, 135, 99, 24); break;
    case YELLOW: light(ledID, 0, 225, 32); break;
    case GREEN: light(ledID, 0, 255, 0); break;
    case BLUE: light(ledID, 0, 0, 255); break;
    case PURPLE: light(ledID, 160, 32, 255); break;
    default: light(ledID, 0, 0, 0);
  }
}

void lights(unsigned int red, unsigned int green, unsigned int
blue)
{
  pixels.setPixelColor(LED_FRONT_LEFT, pixels.Color(red, green,
blue));
  pixels.setPixelColor(LED_FRONT_RIGHT, pixels.Color(red, green,
blue));
  pixels.setPixelColor(LED_BACK_LEFT, pixels.Color(red, green,
blue));
  pixels.setPixelColor(LED_BACK_RIGHT, pixels.Color(red, green,
blue));
  pixels.show();
}

void light(unsigned int ledID, unsigned int red, unsigned int
green, unsigned int blue)
{
  pixels.setPixelColor(ledID, pixels.Color(red, green, blue));
  pixels.show();
}

void leftBlinker(unsigned int color)
{
  static bool lightOn = false;
  static unsigned long timer = millis();
```

```
  if (millis() - timer > BLINK_DELAY_MS)
  {
    if (lightOn)
    {
      light(LED_FRONT_LEFT, WHITE);
      light(LED_BACK_LEFT, WHITE);
    }
    else
    {
      light(LED_FRONT_LEFT, color);
      light(LED_BACK_LEFT, color);
    }

    lightOn = !lightOn;
    timer = millis();
  }
}

void rightBlinker(unsigned int color)
{
  static bool lightOn = false;
  static unsigned long timer = millis();

  if (millis() - timer > BLINK_DELAY_MS)
  {
    if (lightOn)
    {
      light(LED_FRONT_RIGHT, WHITE);
      light(LED_BACK_RIGHT, WHITE);
    }
    else
    {
      light(LED_FRONT_RIGHT, color);
      light(LED_BACK_RIGHT, color);
    }

    lightOn = !lightOn;
    timer = millis();
  }
}
```

```
void backLeftBlinker(unsigned int color)
{
  static bool lightOn = false;
  static unsigned long timer = millis();

  if (millis() - timer > BLINK_DELAY_MS)
  {
    if (lightOn)
    {
      light(LED_FRONT_LEFT, WHITE);
      light(LED_BACK_LEFT, RED);
    }
    else
    {
      light(LED_FRONT_LEFT, color);
      light(LED_BACK_LEFT, color);
    }

    lightOn = !lightOn;
    timer = millis();
  }
}

void backRightBlinker(unsigned int color)
{
  static bool lightOn = false;
  static unsigned long timer = millis();

  if (millis() - timer > BLINK_DELAY_MS)
  {
    if (lightOn)
    {
      light(LED_FRONT_RIGHT, WHITE);
      light(LED_BACK_RIGHT, RED);
    }
    else
    {
      light(LED_FRONT_RIGHT, color);
      light(LED_BACK_RIGHT, color);
    }
```

```
    lightOn = !lightOn;
    timer = millis();
  }
}

void brakeLights()
{
  light(LED_BACK_LEFT, RED);
  light(LED_BACK_RIGHT, RED);
}

void backwardsLights(unsigned int color)
{
  static bool lightOn = false;
  static unsigned long timer = millis();

  if (millis() - timer > BLINK_DELAY_MS)
  {
    if (lightOn)
    {
      light(LED_BACK_LEFT, WHITE);
      light(LED_BACK_RIGHT, WHITE);
    }
    else
    {
      light(LED_BACK_LEFT, color);
      light(LED_BACK_RIGHT, color);
    }

    lightOn = !lightOn;
    timer = millis();
  }
}

void hazardLights(unsigned int color)
{
  static bool lightOn = false;
  static unsigned long timer = millis();

  if (millis() - timer > BLINK_DELAY_MS)
  {
```

```
    if (lightOn)
    {
      lights(WHITE);
    }
    else
    {
      lights(color);
    }

    lightOn = !lightOn;
    timer = millis();
  }
}

void discoLights()
{
  static unsigned long timer = millis();

  if (millis() - timer > 100)
  {
    pixels.setPixelColor(LED_BACK_LEFT, pixels.Color(random(0,
255), random(0, 255), random(0, 255)));
    pixels.setPixelColor(LED_BACK_RIGHT, pixels.Color(random(0,
255), random(0, 255), random(0, 255)));
    pixels.setPixelColor(LED_FRONT_LEFT, pixels.Color(random(0,
255), random(0, 255), random(0, 255)));
    pixels.setPixelColor(LED_FRONT_RIGHT, pixels.Color(random(0,
255), random(0, 255), random(0, 255)));
    pixels.show();

    timer = millis();
  }
}
```

Teamcode

In the team code we have the rules surrounding the project which we are bound to.

Deployment/Method of Operation

1. **Project members are expected to be present at agreed times and places.**
    - a. Absence is tolerated only for valid reasons, such as unforeseen events (e.g., illness, bereavement, accidents) or scheduled events (e.g., wedding, funeral, doctor's appointment), which should be communicated beforehand.
    - b. If a project member is absent without a valid reason, they will receive a warning, which will be recorded in a shared Excel file on OneDrive.
    - c. Members delayed by traffic or public transportation are allowed to be late once, but consistent tardiness is not acceptable.
2. **Project members must report absences before any scheduled meeting or class.**
    - a. Members should inform the group of absences via the designated WhatsApp group.
    - b. Failure to notify in advance will result in a warning (see point 1 - B).
    - c. Absences without a valid reason also result in a warning (see point 1 - B).
    - d. If oversleeping occurs, it must be reported, or a warning will be issued (see point 1 - B); oversleeping is only allowed once.
3. **Each project member is expected to arrive on time for scheduled meetings.**
    - a. If late, the member should report it via the WhatsApp groups.
    - b. A grace period of 15 minutes is allowed for group meetings.
    - c. Arriving late without a valid reason results in a warning (see point 1 - B).
4. **Project members must be accessible.**
    - a. Members should have each other's contacts to facilitate communication.
    - b. Inaccessibility is permitted with a valid reason, such as power outages, work, illness, family matters, or lack of reception. Proof (news report, work schedule, doctor's note) may be requested by the project members.
5. **Project members are expected to honour their commitments.**

- a. Failure to meet commitments without a valid reason results in a warning (see point 1 - B).
- b. If a deadline cannot be met, this should be communicated at least 48 hours in advance. Extensions may be granted through group consensus.
- c. In cases of force majeure, where the member is entirely unable to fulfill their commitment (e.g., severe illness), the other members will cover the work as best as possible.
- d. If a member must leave early, they should notify the group in advance and provide a summary of their completed work.

6. **Each project member must contribute an equitable share of the work. Free-riding is not tolerated.**
    - a. In cases where a member is unable to contribute due to severe learning disorders, illness, etc., the team will provide support.
    - b. If free-riding occurs, the project leader or coordinator will address the issue with the member. If no progress is made, further instances will result in warnings.
7. **Project members should complete assigned tasks and meet deadlines, which will be recorded in meeting minutes.**
    - a. Failing to meet these expectations will result in a warning.
    - b. If the assigned task or commitment is unreasonably large for the allotted time, this should be discussed promptly with the group.
8. **Project members are expected to actively participate in group work, meetings, and classes.**
    - a. Persistent failure to participate actively results in a warning.
    - b. Individual personalities and skill levels will be taken into account.
9. **Project members should hold each other accountable for fulfilling commitments.**

Motivation

1. **Each project member should contribute to a positive working atmosphere.**
    - a. Repeated failure to do so will result in a warning.

2. **Each member is expected to show respect for the ideas of other members.**
    - a. To foster project progress and well-being, members should consider and explore as many ideas as possible.
    - b. Repeated failure to do so will result in a warning.
3. **Each project member should respect the input and opinions of others.**
    - a. Considering and exploring as much input and opinion as possible will benefit the project.
    - b. Repeated failure to do so will result in a warning.
4. **Each project member should be loyal to the group.**
    - a. Honesty and adherence to group agreements are essential for project success and group well-being.
    - b. Repeated violations will result in a warning, including behaviours such as avoiding group discussions and meetings or dishonesty.
5. **Each member should contribute as much as possible to solving problems.**
    - a. This is critical for both academic and project progress.
    - b. Repeated and intentional failure to do so will result in a warning.

Respect

1. **All members should consider the interests of other members.**
    - a. Members should be mindful of each other's needs, as they are a team.
    - b. Repeated failure to do so will result in a warning.
    - c. Non-project-related issues should remain outside of the project.
2. **All members should respect differences.**
    - a. Members and Lennon come from diverse backgrounds, which may result in differences in behaviour, dietary habits, religions, customs, and more.
    - b. If a member feels disadvantaged because of their background, this should be addressed during group meetings.
    - c. Continued issues arising from these differences should be discussed at the request of the affected member in group meetings.

- d. Failure to respect differences will result in a warning.

3. **All members should respect different working styles.**
    - a. Everyone has their own approach to work; different methods can achieve the same goals.
    - b. If a method does not yield the desired result, the issue should be addressed in group meetings.
    - c. A lack of respect for different working methods will result in a warning.
4. **No exclusion or bullying should occur among project members.**
    - a. It is inappropriate to exclude or bully other members and Lennon; all members are expected to behave as adults.
    - b. If someone is being excluded or bullied, the issue should be addressed in group meetings with the victim's consent.
    - c. Perpetrators will receive a warning if such behaviour occurs.
5. **No intimidation or discrimination should occur among project members.**
    - a. Intimidation or discrimination is inappropriate and will not be tolerated at any level.
    - b. Any deliberate attempt to harm a member through intimidation or discrimination will result in an immediate warning.

Understanding Members' Circumstances

1. **Members should consider team members with disabilities.**
    - a. Members should offer assistance to those with disabilities as needed.
    - b. Repeated failure to do so will result in a warning.
2. **Members should consider team members with psychological challenges.**
    - a. Members should offer support to those with psychological difficulties as needed.
    - b. Repeated failure to do so will result in a warning.
3. **Members should consider members who are ill.**
    - a. Members should offer assistance to those who are unwell.

- b. If a member is too ill to meet deadlines, other members should step in to cover their work.
- c. Repeated failure to do so will result in a warning for both points a and b.

4. **Members should consider the personal circumstances of other members.**
    - a. Members should support those experiencing personal difficulties as needed.
    - b. If a member cannot meet deadlines due to personal issues, others should step in to cover their work.
    - c. Repeated failure to do so will result in a warning for both points a and b.

Warnings

1. **A warning will be issued by the project members when any of the above points are violated.**
2. **The project group will keep a record of warnings in a shared Excel file on OneDrive.**
3. **Every warning will be discussed with the group.**
4. **Progress will be reviewed the following week.**
5. **Upon a third warning, The teachers will be contacted to discuss potential consequences, including possible removal from the group with the teachers approval.**
6. **Evidence should be presented to the group and teachers when issuing a warning.**