universidade de aveiro

Discovery Rover

Arquiteturas para Sistemas Embutidos
André Clérigo 98485
Pedro Rocha 98256

TOC

Overview

Our project consists on the creation of a functional “Discovery Rover”. This machine is equipped with a couple of sensors and with remote motor control and a live webcam feed, it offers a new dimension in exploration, disaster management, and environmental monitoring.

The objective is to construct a device that works as proof of concept and could potentially be used as a beta platform for a commercially viable product.

Rover features

1. Remote Control: The rover's motors can be controlled remotely from a dashboard

2. Sensing: Two sensors that retrieve humidity, temperature and LPG gases every 3 sec

3. Monitor: Additionally on the dashboard we can see the sensor data IRL and also see the rover's POV

4. Data Recover: Our rover has a EEPROM and SPIFFS that act as a "Black Box"

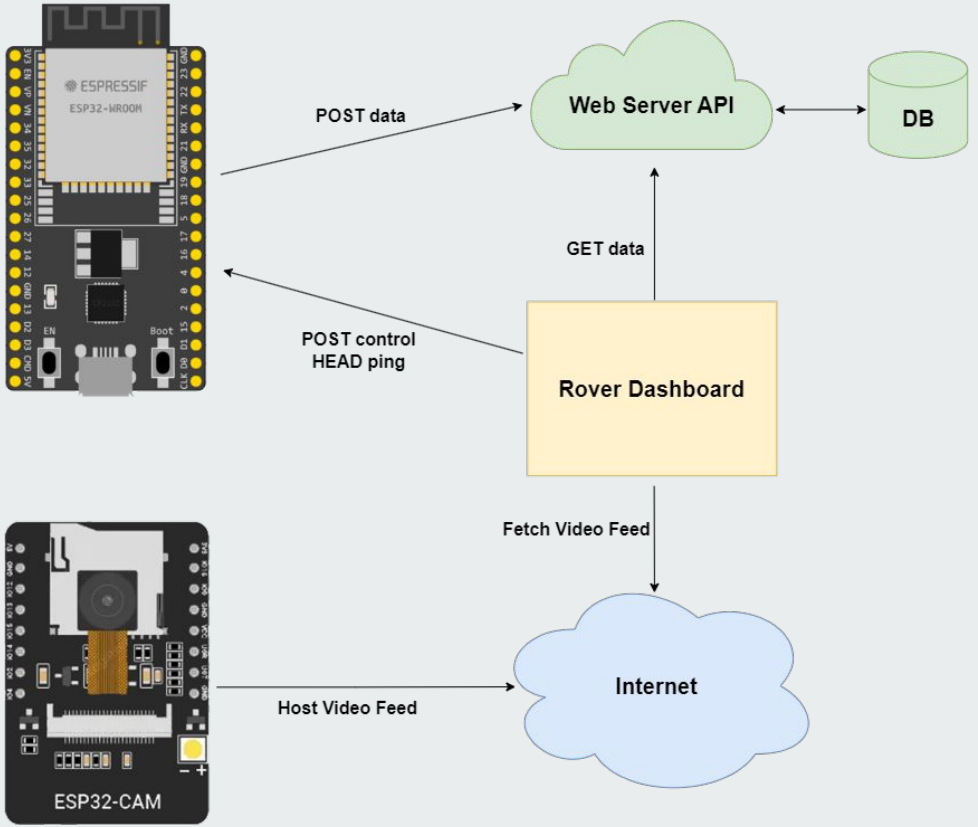
5. IP Information: Equipped with a LCD to know the rover's IP and configured server

6. Alert: Signals its location with a heartbeat sound and emits an alarm when LPG values are high

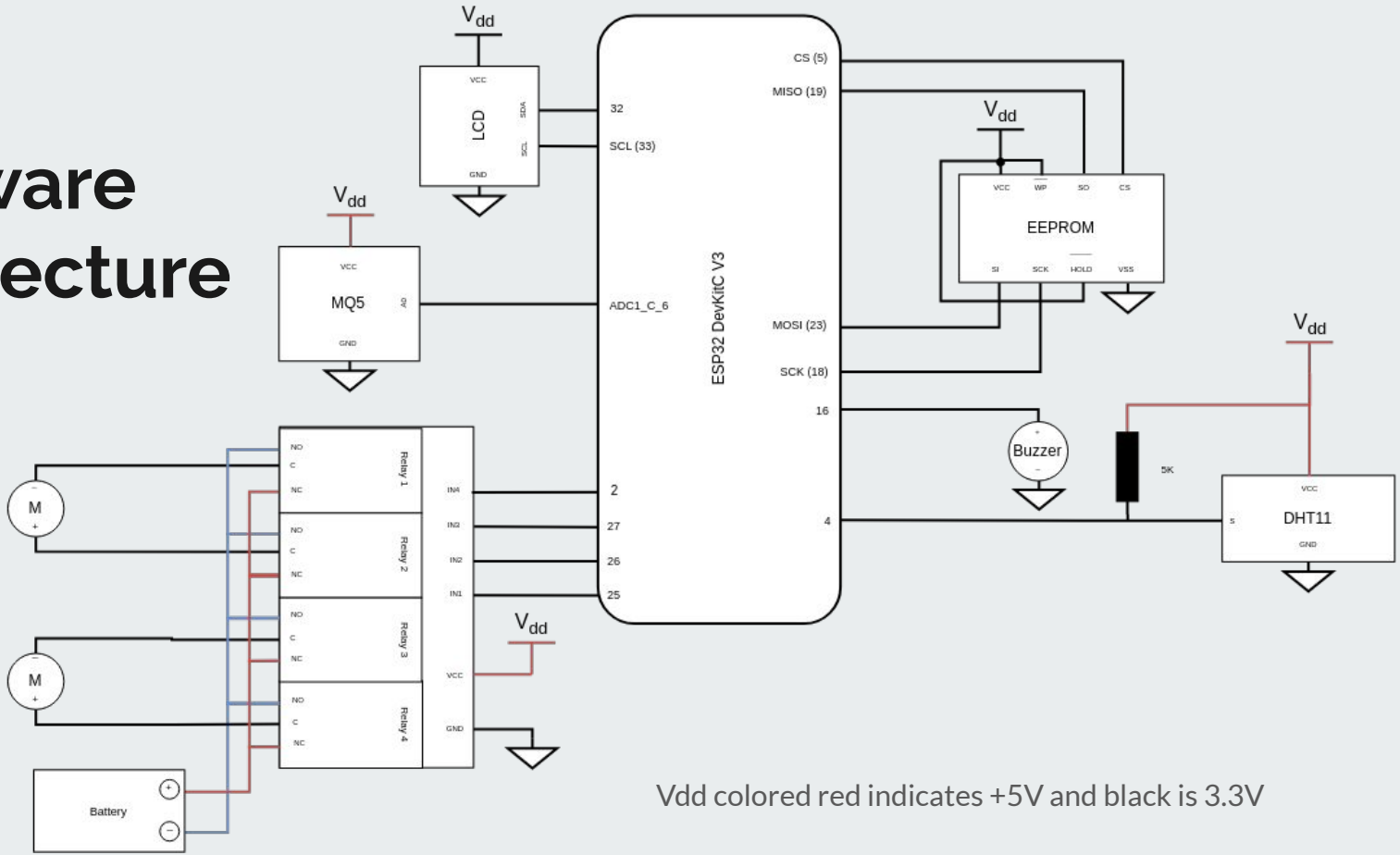
Communication Architecture

The ESP32 DevKitC V3 is simultaneously a Server to receive commands but it also a client of the Web Server API to send the data.

The Dashboard fetches information from the API, the ESP32-CAM and sends commands to the ESP32 DevKitC V3.



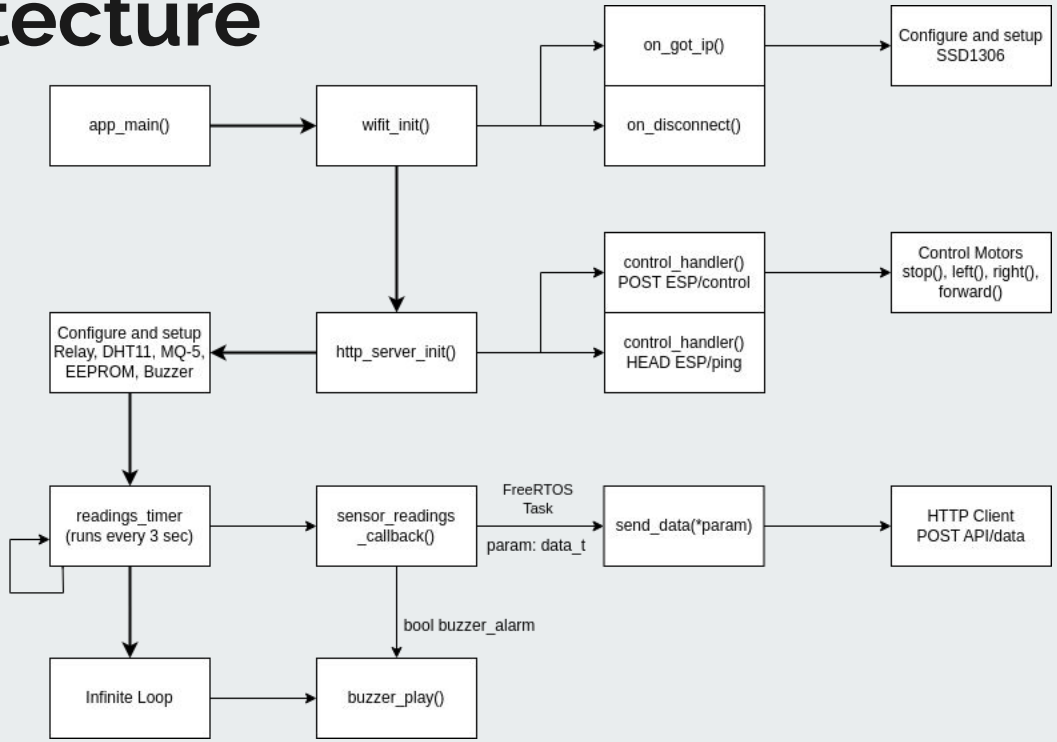
Hardware Architecture



Vdd colored red indicates +5V and black is 3.3V

Software Architecture

The process of our ESP begins by establishing a connection to the WiFi network. After that, it initiates an HTTP Server that remains active to receive commands. Next, it configures all devices and starts a timer retrieves sensor data and transmits it to an API every 3 seconds. Finally, the program enters an infinite while loop where it continuously plays the buzzer.

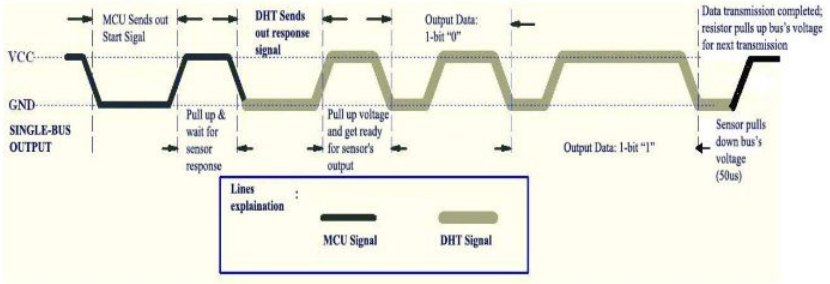


Implementation

DHT11 - Temperature and Relative Humidity Sensor

The sensor communicates using a single data bus.

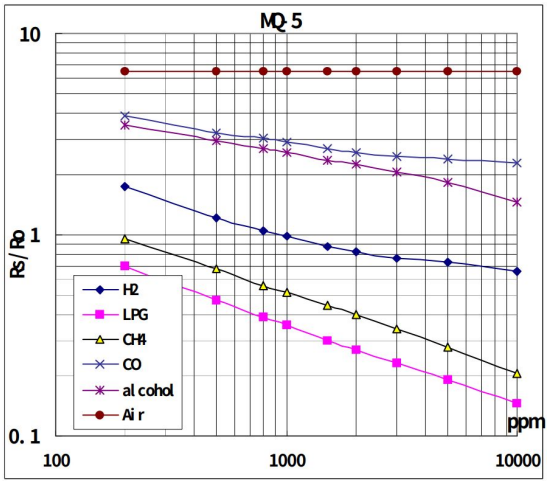
- The data transmitted consists of decimal and integral parts and is 40 bits in total.
- The MCU initiates communication by sending a start signal “0” and then pulling up and waiting.
- The DHT11 responds and sends a 40-bit data signal, the amount of time pulling up indicates if it is a “1” or a “0”.
- DHT11 returns to low-power mode until it receives another start signal.



Implementation

MQ-5 - LPG Gas sensor

The sensor measure gas concentrations and uses an ADC. The sensor has a dielectric membrane that changes the voltage in function of the gases present. We calibrate the values by following the oficial graph dependency (Rs/R0 = 6.5). With those configured we can get the Rs/R0 ratio and get a rough value of ppm of the gas we are testing.



Implementation

SSD1306 - 128 x 32 Dot Matrix OLED/PLED

The SSD1306 is a display controller used for OLED displays. It communicates with a MCU via the I2C protocol where we can:

1. Configure the pin numbers and frequency (400kHz) but also the parameters and commands for the SSD1306 device.
2. Display individual characters by providing the line number, pixel position, character data.
3. Clearing the Screen and Lines by updating the display buffer with blank characters.

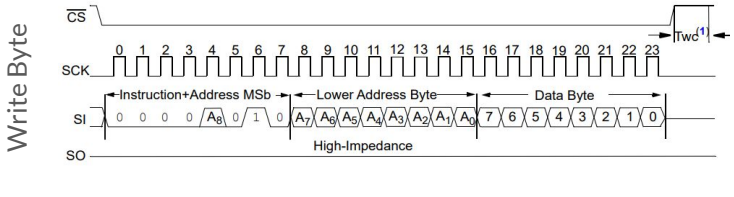
There is a lot more that can be done with this display but for the purposes of our project those were not needed.

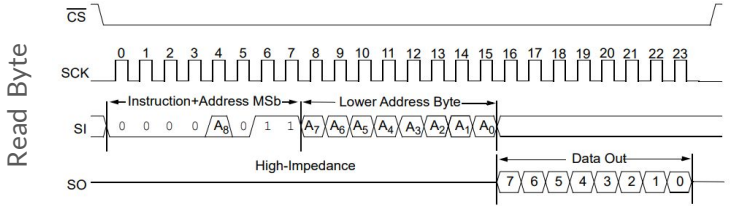
Implementation

EEPROM 25LC040A - 4-Kbit SPI Bus Serial EEPROM

The EEPROM is a non-volatile memory used for data storage. It communicates with a MCU using the SPI protocol, for the purposes of our project we used it to:

1. Initialize the SPI bus and device for the EEPROM by providing the SPI Host ID, CS, SCK, MOSI, MISO pin numbers, and Clock Speed (1MHz).
2. Write a byte to a specific address of the device.
3. Read a byte from a specific address of the device.





Implementation

Piezo Buzzer

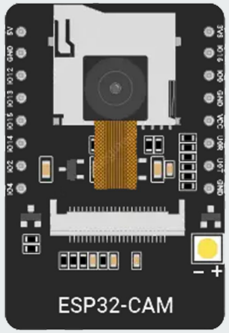
We used PWM to generate different sounds on the buzzer to signal different events.

By default the Buzzer uses a frequency of 2000 Hz with a duty cycle of 50% that simulates a heartbeat sound (100ms enabled and 1100ms disabled).

When in alarm mode the Buzzer switches between the frequencies of 1000 Hz and 2000 Hz every 500ms with a duty cycle of 50%.

Implementation

ESP32-CAM

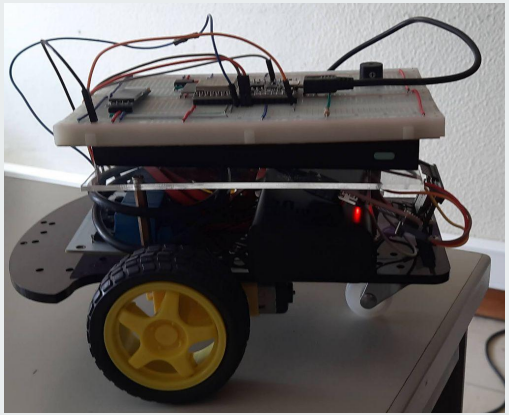


For the ESP32-CAM implementation we used the oficial espressif library for esp_camera. With that library we added the HTTP Server functionality to host a simple web server with the video feed of the ESP.

universidade de aveiro

Implementation

Rover



Our rover establishes a Wi-Fi connection, starts an HTTP server with two endpoints, one allows the rover to be controlled and the other to ping. The program reads temperature, humidity and gas values every 3 seconds using a Timer and sends the data to a server. When the values are above a threshold an alarm is triggered. Kconfig.projbuild files to add menus on Menuconfig where we can configure pins and credentials.

```
Espressif IoT Development Framework Configuration
    Build type  --->
    Bootloader config  --->
    Security features  --->
    Application manager  --->
    Serial flasher config  --->
    Partition Table  --->
    WiFi Configuration  --->
    Buzzer Configuration  --->
    DHT11 Configuration  --->
    EEPROM Configuration  --->
    MQ5 Configuration  --->
    SSD1306 Configuration  --->
    Compiler options  --->
    Component config  --->

[Space/Enter] Toggle/enter  [ESC] Leave menu           [S] Save
[O] Load                    [?] Symbol info           [/] Jump to symbol
[F] Toggle show-help mode   [C] Toggle show-name mode [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)
```

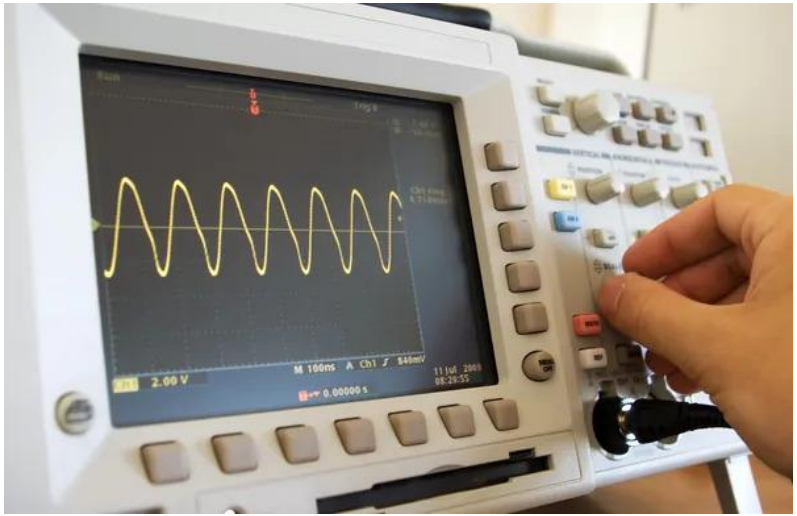
Test Methodology

General test methodology followed:

- Individual component test
- Building the API for that component

The order of components integration:

- Motor system with relays
- EEPROM
- DHT11
- LCD and WiFi
- Buzzer
- MQ-5



Recovering Data

The Discovery Rover is equipped with a dual-layered recovery system which consists of an on-board ESP Filesystem and an external EEPROM.

SPIFFS

- Stores a lot of data (in the order of Megabytes)
- Data is unmodified
- If the Rover fails (ESP32 failure) the data is lost

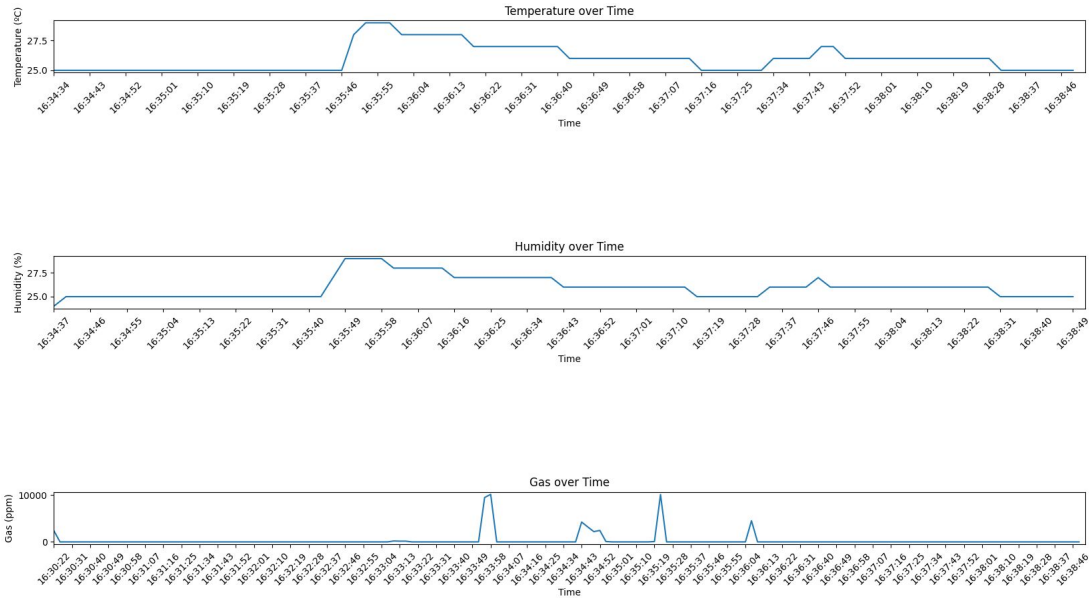
EEPROM

- Can't store a lot of data (in the order of Bytes)
- Data needs to be modified
- More robust to failures

Recovering with EEPROM

Our EEPROM dumps the contents with *read_byte*. A Python script monitors the serial port and analysis the data. When the program is done, it generates three plots with the last recorded temperature, humidity and gases values.

It can contain up to 8 minutes and 31 seconds of data. The values of temperature are whole values and the ppm is truncated to be in 1 byte.



Recovering with SPIFFS

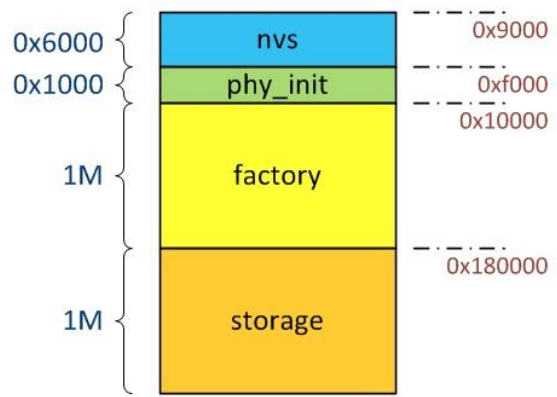
SPIFFS (Serial Peripheral Interface Flash File System) is used to access the flash memory as if it were a normal file system.

The program works by writing the current timestamp on 3 files (temp.txt, humid.txt and gas.txt) and only then it starts writing data. To analyse the data we do something similar to EEPROM but now instead of *read_byte* it reads the content of the files.
There is also the ability to download the data.

The values are stored with no change on the precision. Currently it can save up to 1MB of data which translates to more than 3 days of data.

partitions.csv

```
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     ,        0x6000,
phy_init, data, phy,     ,        0x1000,
factory,  app,  factory, ,        1M,
storage,  data, spiffs,  ,        1M
```



Recovering with SPIFFS

universidade de aveiro

Demo Preview

Future Work

The future work revolves around utilizing OTA updates and integrating it with CI/CD. The idea is basically to configure the ESP32 and Github, so that each time we push new source code to the repository Github generates a new binary file with Github Actions and ESP32 detects the changes and updates itself, improving greatly the IoT development process.

Inspired by: https://github.com/Fishwaldo/esp_ghota

Conclusions

In conclusion, our project to create the "Discovery Rover" has been a success. With temperature, humidity, and gas sensors, it offers real-time environmental monitoring, alerts to hazards, and logs data for analysis in case of the device's failure. A dashboard with sensor readings, remote motor control and a live webcam feed allow the user to better use the device and easily maneuver it from a distance.

Checklist

- A aplicação tira partido do FreeRTOS; ✅
- Uso de interrupções e DMA no contexto do projeto; ✅
- Os dados recolhidos do sensor e processados no ESP32 devem ser apresentados num dashboard remoto; ✅
- Ligação por terminal, independente do dashboard remoto; ✅
- Suporte de atualizações remotas (Over-the-Air) do sistema; ❌
- Pode ser incluído algum tipo de atuador cuja utilização faça sentido com o sensor usado (que seja controlado através do dashboard); ✅
- Pode ser suportado um sistema de ficheiros para armazenar dados localmente; ✅

(Extra nosso) - Uso de um microcontrolador extra para feedback visual no controlo do carro; ✅

Code Baseline

In our project the we used the esp_camera library from espressif to have access to the ESP32-CAM functionalities.

We also used a C library to use ASCII characters to the OLED Display “font8x8_basic.h”.

Evaluation

The contribution of our project between peers is 50/50.