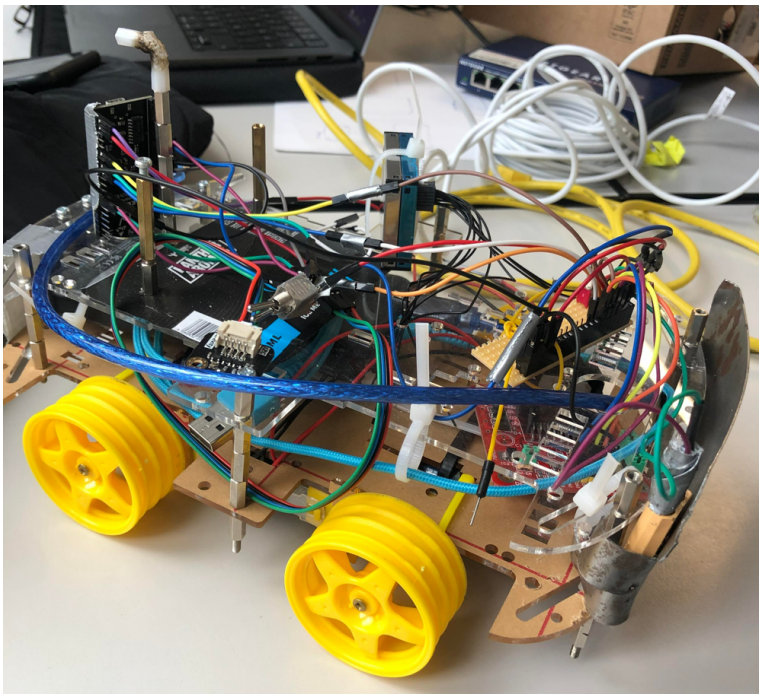
Project 3

Space Planet Rover Vehicle



Authors:

Lara Turunc, Justyna Kluska, Severin Nyffenegger, Thomas Joensen

Date of Submission: 14.10.2022

Supervisors: Kaj Norman Nielsen, Kristian Pontoppidan

Characters with Spaces: 12.325

Summary

The following report documents the work of group 6 on the Space Planet Robot Vehicle which is made based on a Robo-Car project from the 1st semester. We developed this machine with 2 sensors: SGP40 Sensor - the air quality sensor and the SM-UART-04L Laser Dust Sensor. Moreover, the car contains an esp 32 with a digital camera, and a Raspberry Pie which is used outside of the vehicle as the MQTT broker.

Table of Content:

1. Introduction

Nowadays advanced technology helps people in space exploration. Rovers are sent into space to help us get to know more about conditions on the other planets, which for now has happened on Mars, the moon, and to a lesser extent, on asteroids as seen with the Hayabusa2 mission, and its MINERVA-II micro rovers. These vehicles are built in such a way that they can move across the surface of extra-terrestrial bodies and work in hostile conditions. They give us the opportunity to collect images and ground samples. Among other things, this can help us understand the composition of asteroids and the life cycles of planets.

This is the general idea of why we are working on our Space Planet Rover. The final product will include two different sensors that can check the condition of the air and enter polluted areas. Moreover, it will contain a camera that will provide visuals for mission control.

The project has to be made in a group of 4, 2nd-semester students. The work on the project is done mostly during school hours, but also after classes. All the components for the mars rover are provided by the teachers, and the vehicle is built from the chassis of the robot car from the 1st semester. The time frame for the projects is 3 weeks. The final product is a remotely controlled mars rover. The project will end with a group presentation for the rest of the class and teachers when the results of our work will be presented.

The work on a project will be planned using the tools from project management classes. All our progress will be documented in the report, containing documentation of the building and testing the hardware as well as writing, testing, and implementing the codes. To visualize our way of thinking we will also use diagrams and schematics.

2. Requirements

The main requirement is a vehicle that can be remotely operated in a hostile environment. The rover needs to enter the polluted areas and be able to measure different properties in the atmosphere for categorizing. The camera needs to be attached on board to provide visuals for mission control.

3. The Business

When it comes to business, there isn't a whole lot to mention. So far space expiration is an incredibly costly venture, with almost exclusively scientific gains. In fact, the majority of space agencies are run by government bodies, which further implies business as a secondary aspect. That said, a certain south african seems to be on the cusp of breaking the trend of non commercial space travel. So perhaps capitalism may yet make its way into that industry.

4. Hardware block diagram

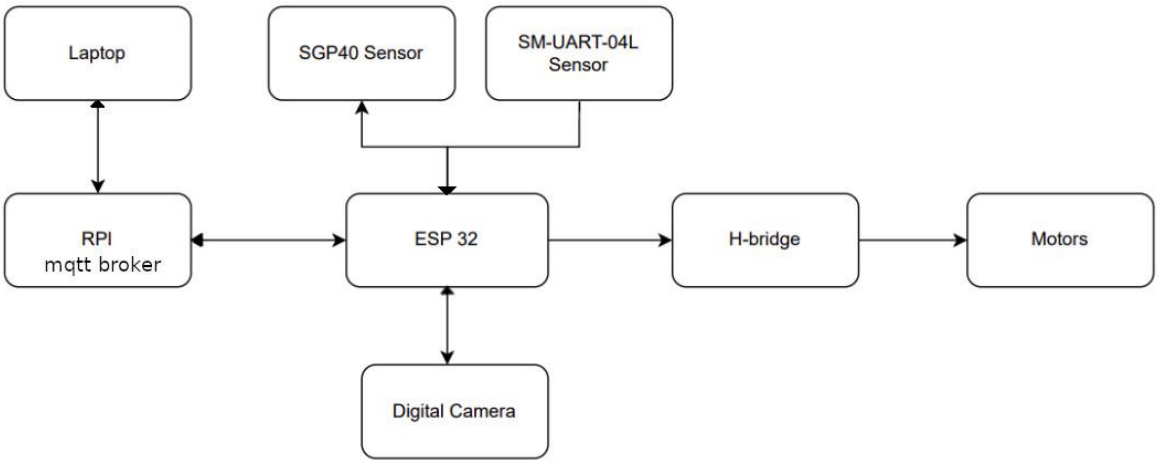


figure 1

Simple diagram to illustrate the interactions between components.

5. Design

5.1. System Block Diagram

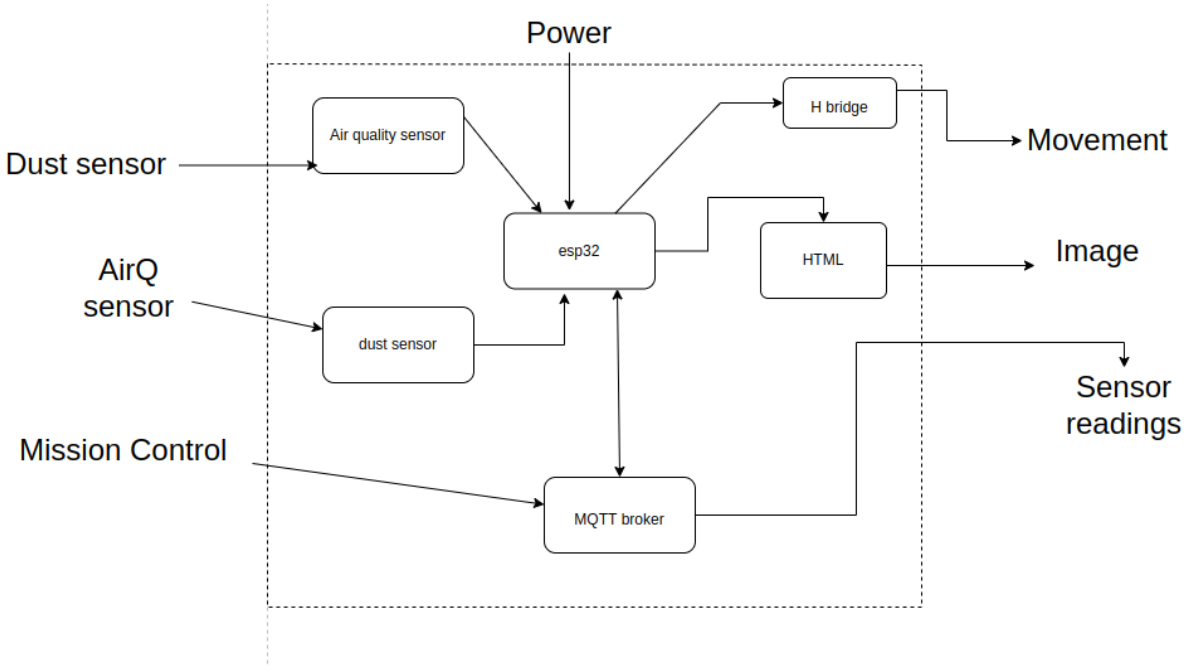


figure 2

Same but with different terms

5.2. UML Diagram

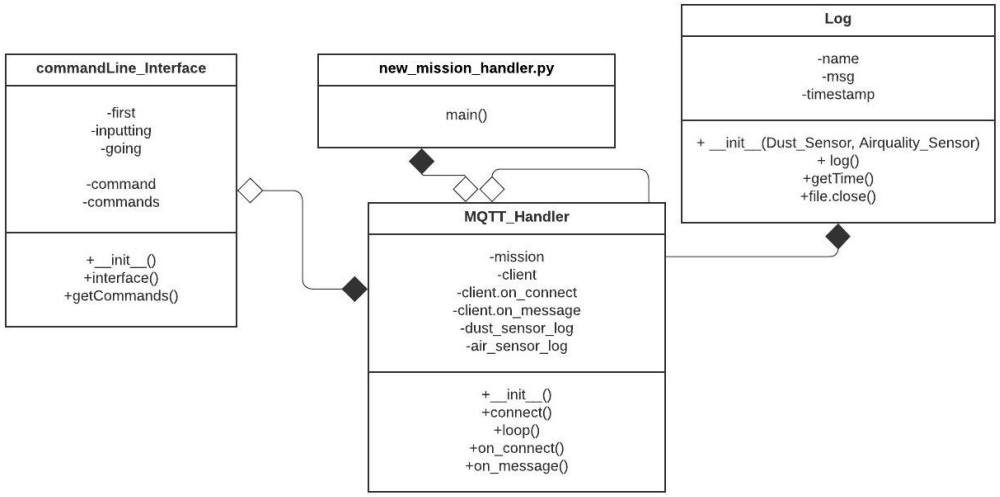
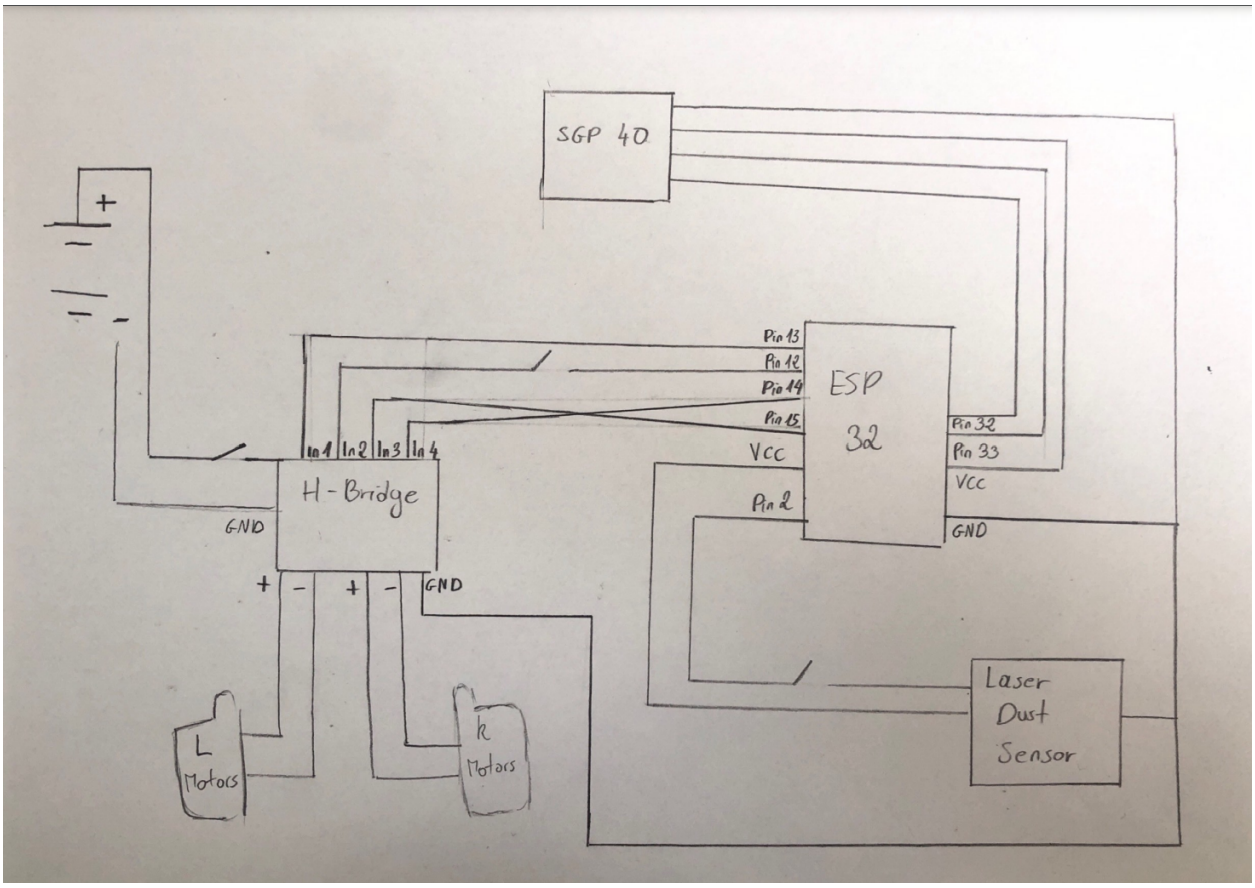


figure 3

note: new_mission_handler.py is not a class, but simply a script.

5.3. Wiring Diagram

figure 4



To show our wire connections and pinout we did the wiring diagram. It clearly shows the connections between the given components. In our implementation, we used 3 different switches that can be seen on the diagram.

6. Implementation

6.1. MQTT Broker

In order to use MQTT to communicate between mission control and the rover, we were required to use mosquitto's mqtt broker for ubuntu.

We set up one of our Raspberry Pi as the MQTT Broker.
To set it up we used the following bash commands:

Install mosquito or mosquitto-clients with apt.

```
$ sudo apt install -y mosquitto
```

Enable the service via systemctl

```
$ sudo systemctl enable mosquitto.service
```

To enable external use, you can either:
use a text editor like nano

```
$ sudo nano /etc/mosquitto.conf
```

and add the following lines,

```
  listener 1883
  allow_anonymous
```

or use the tee command

```
$ echo 'listener 1883' | sudo tee -a /etc/mosquitto/conf.d
$ echo 'allow_anonymous' | sudo tee -a /etc/mosquitto/conf.d
```

6.2. ESP 32

The ESP32 is the central part of our rover and therefore we needed it to be operational. We connected each motor and sensor to specific pins on the esp to control the components. Though it was a challenge to find them, the ESP32Wrover does have enough pins to connect each component and also run the camera. A challenge we encountered, and the reason for the apparently random switches is that some of the pins we used are required to upload code to it.

The ESP32 needs to take commands and parameters from the MQTT broker and control the motors and sensors. After the setup of the MQTT, we changed the callback

method with the code snipped we received. In the callback method, we call the function that runs all our commands. The function is called runMission().

```
void runMission(){
 for(int i = 0; i <= missionCmdIndex; i++){
   String message = missionCmd[i];
   command = getCommand(message);
   parameter = getParameter(message);

   switch (command){
     case 'f':
       forward(parameter);
       break;
     case 'b':
       backward(parameter);
       break;
     case 'r':
       pivotR(parameter);
       break;
     case 'l':
       pivotL(parameter);
       break;
     case 'w':
       Delay(parameter);
       break;
     case 'a':
       for (int i = 0; i < parameter; i++) {
         Airquality_Sensor();
         Dust_Sensor();
         delay(500);
         }
       break;
   }}}
```

The runMission()-function cycles through the char-Array, MissionCmd[], with instructions via the for loop, which itself cycles through the index of the message. So using the char-Array, the specific commands and parameters, letters and numbers, are extracted via the ‘getter’ functions. For example, if we have the commands f30, l45, w5, a4 and stop, the switch-case calls the forward()-method with the parameter 30, which runs the car until it has driven approximately 30 cm. That’s right, we tried to implement easier metrics than seconds. After that, it would call the left turn, which turns it in-place, for roughly 45 degrees. Then, a 5 second waiting period, and finally it calls both sensor methods and publishes the output 4 times.

6.3. Motors

```
#define RF 12

#define RB 13

#define LF 15

#define LB 14
```

The motors need to be implemented in the code as left- forward / backward and right- forward / backward. With these definitions, we were able to write methods with analogWrite() for going forward, backward, pivoting left, and pivoting right. After we managed to get the wheels turning, the next step was to run the motors with a specific parameter. To drive forward and backward we chose centimeters and for turning we chose degrees. We counted the milliseconds it needed to go one degree and one meter, and with that we were able to add a delay function according to the parameter.

```
void forward(int t) {
  stop();

  analogWrite(RF, 200);
  analogWrite(LF, 255);

  DelayD(t);
}
```

```
void backward(int t) {
  stop();

  analogWrite(LB, 255);
  analogWrite(RB, 200);

  DelayD(t);
}
```

```
void pivotL(int t){
  stop();
  analogWrite(RB, 255);
  analogWrite(LF, 255);

  DelayL(t);
}
```

```
void pivotR(int t){
  stop();
  analogWrite(LB, 255);
  analogWrite(RF, 255);

  DelayR(t);
}
```

figures 5,6,7,8

The keen eye will notice a difference in motor speeds, which is done to correct the left side going faster than the right, and turn the car while going straight. Also the difference in delay method.

6.4. Relevant delays

In order to have somewhat accurate driving, we need specific types of delays.

The first one is the one that permits a delayed start, or mid mission pauses, which can be used if you need data from a specific time of day.

For the rest we have one for driving, DelayD() and one for each turn, DelayL() and DelayR()
As you'll see, The Delay() is basically a vestige from previous iterations as it simply delays with 1000 * parameter. It was previously used for all delays.

Now though, driving has a factor of 28, so it delays 28 milliseconds, as that's the approximated time it takes to drive one centimeter. This is of course not entirely accurate as we did this through trial and error.
For further development, one could also be added for reversing, as it drives slightly slower in reverse.

The same can be said for the turns, where left is faster than right, therefore requiring a smaller delay to match a rough approximation of degrees.

Three flaws of this approach is not incorporating a battery voltage check, as the delays must be increased at lower power, not not spending the gruelly hours it would take to completely get the distances and angles perfect,
and of course switching to a non-blocking delay.[1]

```
void Delay(int t){
  if (t > 0) {
    delay(t * 1000);
    stop();
    delay(20);
  } else {
    delay(5);
    stop();
  }
}
void DelayD(int t){
  if (t > 0) {
    delay(t * 28);
    stop();
    delay(20);
  } else {
    delay(5);
    stop();
  }
}

void DelayL(int t){
  if (t > 0) {
    delay(t * 9);
    stop();
    delay(20);
  } else {
    delay(5);
    stop();
  }
}
void DelayR(int t){
  if (t > 0) {
    delay(t * 11);
    stop();
    delay(20);
  } else {
    delay(5);
    stop();
  }
}
```

figure 9

[1] We did try this, but since the camera still works with blocking delays, it wasn't prioritised.

6.5. Laser Dust Sensor

The code to implement the dust sensor was primarily given to us by the teachers. We only needed to wire and pinMode it correctly and add the MQTT part.

```
Serial2.begin(9600,SERIAL_8N1, 2,-1);
```

We used pin 2 to connect it to the ESP32. Since it is a strapping pin, we had to add a switch in order to upload the code to the ESP32 successfully. As trying it without breaking the connection would result in an upload failure with error code 2 in the arduino ide.

```
1. String output = String(dust25);
2. output.toCharArray(msg, MSG_BUFFER_SIZE);
3. client.publish("RoboCar/Sensor/Dust", msg);
```

Because of the fact that we can only upload chars to the MQTT, we had to implement a String to char-Array method, which may seem arbitrary since they're basically the same thing in some languages. In line 1 we take the output (dust25) of the dust sensor and convert it to a String (output). In line 2 we convert this String to a char array in order to post it successfully in line 3 to our specific topic on the MQTT.
A nifty conversion and sorting algorithm was provided by Jonas[2], so hopefully we get the correct data. And just to mention it, this could be a major oversight if he supplied us with an incomplete code to tests us.

[2] you already know who he is

6.6. Air Quality Sensor

```
#define I2C_SDA 32
#define I2C_SCL 33
```

The air quality sensor needs two pins to work as it relies on the I²C protocol. We took pins 32 for the clock and 33 for data.

```
1. uint16_t index = mySgp40.getVocIndex();
2. String testString = String(index);
3. testString.toCharArray(msg, MSG_BUFFER_SIZE);
4. client.publish("RoboCar/Sensor/Air", msg);
```

I²C is a funny, yet useful protocol, as it outputs as it starts with the master sending a start condition, which prompts a specific slave to return 8 bits of data followed by a single bit receipt. When the transmission is complete, a stop condition is returned to the master.

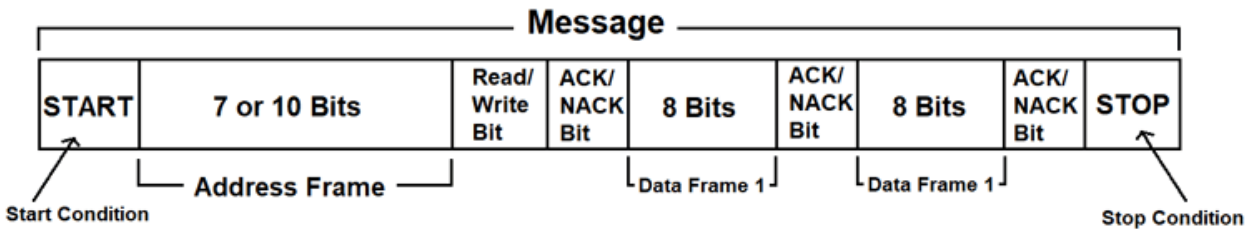


figure 10, from Circuit Digest

Having said that we can't publish an integer on the mqtt, only chars, we convert it into a String in line 2. As it is stated in line 3, the conversion from a String to a char is the same as with the dust-sensor. In line 4 we post it to our specific topic on the mqtt.

6.7. Camera

In order to get a better running time on the ESP32, we implemented a simpler and more efficient camera setup code. With this code, the quality, resolution and effects are already predefined, which allows us to get a smoother picture and an actual live feed, since it eliminates some overhead.

7. Python

MQTT Handler: Car control and sending commands happen here as we publish the commands that are given in the terminal interface. In the on_connect function, we subscribe to a specific topic in the MQTT client and use its IP to make them communicate. After typing the command line, we can see the data from the sensors in the files named after the date of their creation. We print the topics in the on_message function and get the data from the sensors.

```
def on_connect(self, client, userdata, flags, rc):
    self.mission.interface()
    commands = self.mission.getCommands()
    print(commands)

    print("Connected with result code " + str(rc))

    self.client.subscribe([("RoboCar/Sensor/Air", 0), ("RoboCar/Sensor/Dust", 0)])
    #client.subscribe([("gateway/a555b555c555d555/rx", 0), ("gateway/new_topic/rx", 0)])
    self.client.publish("RoboCar/Mission/", commands)
```

```
    def on_message(self, client, userdata, msg):
        print(msg.topic + " " + str(msg.payload))

        #data = msg.payload.decode("utf-8")
        #print(data)
        print(msg.topic == 'RoboCar/Sensor/Air')
        if msg.topic == 'RoboCar/Sensor/Air':
            airquality = msg.payload
            print(airquality)
            self.air_sensor_log.log(airquality)
            #self.air_sensor_log.close(self)
```

The data files are saved separately in “air” and “dust” folders.

```python
import os
import datetime

LOG_FOLDER = 'logs'


class Log:
    def __init__(self, name):
        cwd = os.getcwd()
        if not os.path.exists(cwd+"/"+LOG_FOLDER+"/"):
            os.mkdir(cwd+"/"+LOG_FOLDER+"/")
        self.path=cwd+"/"+LOG_FOLDER +"/"+ name+"/"
        if not os.path.exists(self.path):
            os.mkdir(self.path)
```

```python
    def log(self, msg):
        self.x = datetime.datetime.now()
        self.file = open(self.path + str(self.x.day) + "." + str(self.x.month) + "." + "." + str(self.x.year) + ".txt", "a")
        self.file.write(self.getTime()+str(msg)+"\n")
        #self.file.close()

    def getTime(self):
        x = datetime.datetime.now()
        return "["+x.strftime("%X")+"] "
    #def close(self):
        #self.file.close()
```

Interface: This is a user-friendly interface that makes controlling the car easier. After you start the program, you can give the rover directions with certain commands. The terminal interface commands are as follows: forward, backward, right, left and sense. 'wait' for waiting, 'stop' to finish and send the command, and 'end' to close the program early. Commands are called with getCommand function in the mission handler.

```
Run:  new_mission_handler
/Users/laraturunc/PycharmProjects/pythonProject1/venv/bin/python /Users/laraturunc/PycharmProjects/pythonProject1/new_mission_handler.py
Welcome to mission control, press the any-key to continue..

Possible commands are: Forward, backward, pivot left, pivot right, turn left, turn right.
 for driving forward and backwards type input as centimeters and for turning type input as degrees (0 for indefinite)  ::
 When done type stop

input first command  ::
```

```
if self.__command == 'forward' or self.__command == 'f' : #forward
    print(self.__command, self.__duration)
    self.__commands += 'f'
    self.__commands += self.__duration
    self.__commands += ' '
```

Areas with room for improvement include

A need to visualise data externally.
The mqtt loop blocking the program indefinitely as we couldn't crack the loop(x secs) or loop_start() multithread in time.

8. Test

8.1. MQTT Test

We initially mistook setting up our own broker as a much more arduous task than it actually was, so we used the shared MQTT broker in the class for testing control of the car, output of the mission control and reception of sensor data. MQTT in programming simply requires the correct IP address, and authentication if needed, so switching to our own broker later was a minor task.

In truth it required a rather minimal setup. After installing the broker, it was a simple task to start the service and start pinging it and listening to it through local_host. Afterward, opening the port and allowing anonymous use was next.

There isn't much to say for the tests we conducted with the MQTT itself, as it either is or isn't set up correctly. All you can really test for is if you wrote the messages and topics correctly.

As for testing the implementation, we've been testing the car by posting the sequences with instructions through MQTT-box. Later on we've been testing the mission control's capacity to send correct messages using MQTT, as well as being able to receive and log data from the sensors.

8.2. Motors and H-Bridge Test

Our first task we did was to get the wheels turning in the right direction. We tested each individual pin on the h-bridge to know which pin is responsible for driving forward / backward and if it is on the left side or the right side. After that we created the pivoting methods and realized that the motors are not equally strong on the right and left. We fixed this issue by dropping the speed on the right side to 200. Afterwards the car drove straight and turned as we wanted it.

A serious issue we faced was inconsistent results to begin with, as the socket we'd made for the esp had too much internal resistance to properly power the h bridge.

8.3. Laser Dust Sensor Test

The code to convert the hex numbers to integers was provided to us by the teachers. Our job was to get the wiring right and implement the MQTT.

We took the data-sheet from the sensor and wired it accordingly. After connecting it to the ESP32, we realized that we are using a strapping pin and need to add a switch. With the switch added, we were able to upload the code to the ESP32 and also transfer the data over MQTT to our topic. We never really ran into issues after that.

8.4. Air Quality Sensor Test

After adding the set-up code to our Arduino file, we added the MQTT part. From there on, it ran downright flawlessly.

8.5. The Camera Test

At first we used the provided camera set-up code from the library. However, after implementing the code together with the motors and sensors, the live feed we received was of very poor quality and it only updated every few seconds. The average frame rate per hour could be counted on two hands.

Luckily, we found another set-up and program online, which gave us a much smoother picture and actual live feed without all the fluff of the original.[3]

[3] citation needed - Ask Cosmin

8.6. The Python Code Test

On the python side, we had our biggest difficulties, because of the complexity of using object-oriented programming and MQTT, and a first time python user writing most of it. We first had to set up the MQTT and try to subscribe and publish to the right topic.

Afterward, we created the interface on the terminal to save a mission. One of the problems we had was to take the created mission from the interface class and implement it into the MQTT-handler. We managed to solve the problem by creating an instance of the interface in the MQTT-handler. The mission was a String at that point and we could publish it on “RoboCar/Mission/”. The rover subscribed to the same topic and was able to take commands from it.

Another issue we had was the logging of the sensor data. We create a file, as soon as we receive something on the MQTT topic “RoboCar/Sensor/Air” or “RoboCar/Sensor/Dust”. The sensor-data gets logged into the file with a time stamp.

8.7. Integration Test

We started to test the entire wrover with the MQTT very early on. At first, we needed to adapt the mission-handler in the Arduino. Around a week before the deadline, our wrover took the mission from the python interface correctly. During this week, our goal was to enhance our driving with centimeters and degrees. We also added a wait-function, so we could delay our mission within the given time. Tweaking the python interface was also on the list.

9. Prove

Driving the rover with mqttbox: https://youtu.be/5Oq17_CJ7LY

Driving the rover with python: https://www.youtube.com/watch?v=HM2AVd1rxUU&feature=share

Logging files: https://youtu.be/HM2AVd1rxUU

Camera : https://youtu.be/trdzIjL23Rw

10. Conclusion

To conclude, we have managed to create a functional planet wrover that meets the teacher's requirements of the project. We can navigate our wrover from our laptop through the MQTT broker. Both sensors work smoothly, therefore we can read the output from them on our pc. We only used hardware given by teachers and made necessary soldering. We managed to finish our project before the deadline. We worked on the project during and after the scheduled school hours. Both workload and atmosphere in our group were great because we were communicating efficiently with each other and we dealt with emerging problems on a regular basis. Although the project was a challenge for us, we are satisfied with its effects and we are waiting for more exercises.

11. Appendices

11.1. Link to zipped source code (We can only upload docx or pdf to the assignment)

https://github.com/ThommyJabrony/EAAA-public/blob/main/G6-Rover-Source_code.zip

11.2. C code:

```
#include "esp_camera.h"
#include <WiFi.h>
#include "esp_timer.h"
#include "img_converters.h"
#include "Arduino.h"
#include "fb_gfx.h"
#include "soc/soc.h"             // disable brownout problems
#include "soc/rtc_cntl_reg.h"    // disable brownout problems
#include "esp_http_server.h"

#define PART_BOUNDARY "123456789000000000000987654321"



#define CAMERA_MODEL_WROVER_KIT

#define PWDN_GPIO_NUM    -1
```

```
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM     21
#define SIOD_GPIO_NUM     26
#define SIOC_GPIO_NUM     27

#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       19
#define Y4_GPIO_NUM       18
#define Y3_GPIO_NUM        5
#define Y2_GPIO_NUM        4
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22

static const char* _STREAM_CONTENT_TYPE = "multipart/x-mixed-replace;boundary="
PART_BOUNDARY;
static const char* _STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n";
static const char* _STREAM_PART = "Content-Type: image/jpeg\r\nContent-Length:
%u\r\n\r\n";

httpd_handle_t camera_httpd = NULL;
httpd_handle_t stream_httpd = NULL;

static const char PROGMEM INDEX_HTML[] = R"rawliteral(
<html>
 <head>
```

```
   <title>Mars Rover live feed</title>
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <style>
     body { font-family: Arial; text-align: center; margin:0px auto; padding-top:
30px;}
     table { margin-left: auto; margin-right: auto; }
     td { padding: 8 px; }
     .button {
       background-color: #2f4468;
       border: none;
       color: white;
       padding: 10px 20px;
       text-align: center;
       text-decoration: none;
       display: inline-block;
       font-size: 18px;
       margin: 6px 3px;
       cursor: pointer;
       -webkit-touch-callout: none;
       -webkit-user-select: none;
       -khtml-user-select: none;
       -moz-user-select: none;
       -ms-user-select: none;
       user-select: none;
       -webkit-tap-highlight-color: rgba(0,0,0,0);
     }
     //camera resolution
     img {  width: 200 ;
       max-width: 100% ;
```

```
      height: 200 ;
    }
  </style>
</head>
<body>
  <h1>Group 6 live feed</h1>
  <img src="" id="photo" >

 <script>
 function toggleCheckbox(x) {
   var xhr = new XMLHttpRequest();
   xhr.open("GET", "/action?go=" + x, true);
   xhr.send();
 }
 window.onload = document.getElementById("photo").src =
window.location.href.slice(0, -1) + ":81/stream";
</script>
</body>
</html>
)rawliteral";
//Camera setup

#include "camera_pins.h"
#include <DFRobot_SGP40.h>
#include <Wire.h>
#define I2C_SDA 32
#define I2C_SCL 33
DFRobot_SGP40 mySgp40;
//Setup for airquality sensor
```

```
char dustMsg[30];
int dust25 = 0;
//Dust sensor setup

#include <PubSubClient.h>
const char* ssid = "ITEK 2nd";
const char* password = "Four_Sprints_F21v";
const char* mqtt_server = "10.120.0.94";
WiFiClient espClient;
PubSubClient client(espClient);
unsigned long lastMsg = 0;
#define MSG_BUFFER_SIZE (50)
char msg[MSG_BUFFER_SIZE];
char command;
int parameter = 0;
String missionCmd[100];
int value = 0;
int missionCmdIndex = 0;
//MQTT setup

#define RF 12
#define RB 13
#define LF 15
#define LB 14
//Motor setup

void startCameraServer();
```

```
void setup() {
 Serial.print("Camera Ready! Use 'http://");
 Serial.print(WiFi.localIP());
 Serial.println("' to connect");
 // Camera setup


 Serial.begin(9600);
 Serial2.begin(9600,SERIAL_8N1, 2,-1);
 Serial.end();
 //Setup Dust sensor


 Wire.begin(I2C_SDA, I2C_SCL);
 Serial.begin(115200);
 while(mySgp40.begin(/*duration = */10000) !=true){
   Serial.println("failed to init chip, please check if the chip connection is
fine");
   delay(1000);
 }
 Serial.println("-------------------------------------------");
 Serial.println("sgp40 initialized successfully!");
 Serial.println("-------------------------------------------");
 //Setup for airquality sensor


 Serial.begin(115200);
 setup_wifi();
 client.setServer(mqtt_server, 1883);
 client.setCallback(callback);
 //MQTT setup
```

```
pinMode(RF, OUTPUT);
pinMode(RB, OUTPUT);
pinMode(LF, OUTPUT);
pinMode(LB, OUTPUT);
//Motor setup
 WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); //disable brownout detector

Serial.begin(115200);
Serial.setDebugOutput(false);
 camera_config_t config;
config.ledc_channel = LEDC_CHANNEL_0;
config.ledc_timer = LEDC_TIMER_0;
config.pin_d0 = Y2_GPIO_NUM;
config.pin_d1 = Y3_GPIO_NUM;
config.pin_d2 = Y4_GPIO_NUM;
config.pin_d3 = Y5_GPIO_NUM;
config.pin_d4 = Y6_GPIO_NUM;
config.pin_d5 = Y7_GPIO_NUM;
config.pin_d6 = Y8_GPIO_NUM;
config.pin_d7 = Y9_GPIO_NUM;
config.pin_xclk = XCLK_GPIO_NUM;
config.pin_pclk = PCLK_GPIO_NUM;
config.pin_vsync = VSYNC_GPIO_NUM;
config.pin_href = HREF_GPIO_NUM;
config.pin_sscb_sda = SIOD_GPIO_NUM;
config.pin_sscb_scl = SIOC_GPIO_NUM;
config.pin_pwdn = PWDN_GPIO_NUM;
```

```
config.pin_reset = RESET_GPIO_NUM;
config.xclk_freq_hz = 20000000;
config.pixel_format = PIXFORMAT_JPEG;
 if(psramFound()){
  config.frame_size = FRAMESIZE_VGA;
  config.jpeg_quality = 10;
  config.fb_count = 2;
} else {
  config.frame_size = FRAMESIZE_SVGA;
  config.jpeg_quality = 12;
  config.fb_count = 1;
}
 // Camera init
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
  Serial.printf("Camera init failed with error 0x%x", err);
  return;
}
// Wi-Fi connection
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");
 Serial.print("Camera Stream Ready! Go to: http://");
Serial.println(WiFi.localIP());
```

```
  // Start streaming web server
 startCameraServer();
 //Camera setup
}

void loop(){
 MQTT_handler();
}

void runMission(){
 for(int i = 0; i <= missionCmdIndex; i++){
   String message = missionCmd[i];
   command = getCommand(message);
   parameter = getParameter(message);

   switch (command){
     case 'f':
       forward(parameter);
       break;
     case 'b':
       backward(parameter);
       break;
     case 'r':
       pivotR(parameter);
       break;
     case 'l':
       pivotL(parameter);
       break;
```

```
      case 'w':
        Delay(parameter);
        break;
      case 'a':
        for (int i = 0; i < parameter; i++) {
          Airquality_Sensor();
          Dust_Sensor();
          delay(500);
          }
        break;
    }
 }
}

void stop(){
 analogWrite(LB, 0);
 analogWrite(RB, 0);
 analogWrite(LF, 0);
 analogWrite(RF, 0);

}

void forward(int t) {
 stop();

 analogWrite(RF, 200);
 analogWrite(LF, 255);
```

```
 DelayD(t);
}

void backward(int t) {
 stop();

 analogWrite(LB, 255);
 analogWrite(RB, 200);

 DelayD(t);
}

void pivotR(int t){
 stop();
 analogWrite(LB, 255);
 analogWrite(RF, 255);

 DelayR(t);
}
void pivotL(int t){
 stop();
 analogWrite(RB, 255);
 analogWrite(LF, 255);

 DelayL(t);
}

void Delay(int t){
```

```
 if (t > 0) {
   delay(t * 1000);
   stop();
   delay(20);
 } else {
   delay(5);
   stop();
 }
}
void DelayD(int t){
 if (t > 0) {
   delay(t * 28);
   stop();
   delay(20);
 } else {
   delay(5);
   stop();
 }
}

void DelayL(int t){
 if (t > 0) {
   delay(t * 9);
   stop();
   delay(20);
 } else {
   delay(5);
   stop();
```

```
 }
}
void DelayR(int t){
 if (t > 0) {
   delay(t * 11);
   stop();
   delay(20);
 } else {
   delay(5);
   stop();
 }
}

void Airquality_Sensor(){
   uint16_t index = mySgp40.getVoclndex();
    Serial.print("vocIndex = ");
   String testString = String(index);
   testString.toCharArray(msg, MSG_BUFFER_SIZE);
   Serial.println(index);
   delay(1000);

   client.publish("RoboCar/Sensor/Air", msg);
}

void Dust_Sensor(){
    if(Serial2.available()){

   if((Serial2.read() == 0X42) && (Serial2.read() == 0x4D)) // if we got the
headers
```

```
    {
      int msgIdx=0;
      for(int i = 0;i<30;i++){
        while(!Serial2.available()); //wait for byte
        dustMsg[msgIdx++] = Serial2.read();
        //if we get a byte put it in the array and incerement index

      }
      dust25 = (dustMsg[4]<<8) + dustMsg[5];
      //when we are done, we can take out the relevant data and calculate value

    }

  }
  Serial.println(dust25);
  String output = String(dust25);
  output.toCharArray(msg, MSG_BUFFER_SIZE);
  delay(1000);

  client.publish("RoboCar/Sensor/Dust", msg);
}

void MQTT_handler(){
  if (!client.connected()) {
    reconnect();
  }

  client.loop();
```

```
}

void setup_wifi() {

 delay(10);
 // We start by connecting to a WiFi network
 Serial.println();
 Serial.print("Connecting to ");
 Serial.println(ssid);

 WiFi.mode(WIFI_STA);
 WiFi.begin(ssid, password);

 while (WiFi.status() != WL_CONNECTED) {
   delay(500);
   Serial.print(".");
 }

 randomSeed(micros());

 Serial.println("");
 Serial.println("WiFi connected");
 Serial.println("IP address: ");
 Serial.println(WiFi.localIP());
}

char getCommand(String message){
  return message[0];
```

```
}


int getParameter(String message){
 String valueString = message.substring(1); //creates a new string starting from
index 1 (after the command letter)
 int value = valueString.toInt(); //converts a string to an integer
 return value;
}


void callback(char* topic, byte* payload, unsigned int length) {
 Serial.print("Message arrived [");
 Serial.print(topic);
 Serial.print("] ");

 String temp = ""; // string to store command
 missionCmdIndex = 0; // variable to count number of commands
 for (int i = 0; i < length; i++) {

   if ((char)payload[i]=='s'){ //stop
     Serial.println("Mission data received");

     runMission();

 //** PRINT OUT MISSION DATA (HERE, INSTEAD, YOU WANT TO CALL A FUNCTION THATRUNS
THE MISSION)
     for (int i = 0;i<missionCmdIndex;i++){

       Serial.print("Command: ");
       Serial.print(getCommand(missionCmd[i]));
```

```
        Serial.print(" ");
        Serial.print("Parameter: ");
        Serial.println(getParameter(missionCmd[i]));
        }
    }
 temp += (char)payload[i]; // add character to mission command

 if ((char)payload[i]==' ') // if we receive next mission command
 {
   missionCmd[missionCmdIndex++]=temp; // store command and increment qeue index
   temp = ""; // clear temp string
 }

 }
}


void reconnect() {
 // Loop until we're reconnected
 while (!client.connected()) {
   Serial.print("Attempting MQTT connection...");
   // Create a random client ID
   String clientId = "ESP8266Client-";
   clientId += String(random(0xffff), HEX);
   // Attempt to connect
   if (client.connect(clientId.c_str())) {
     Serial.println("connected");
     // Once connected, publish an announcement...
```

```
      client.publish("RoboCar/connection", "hello world");
      // ... and resubscribe
      client.subscribe("RoboCar/Mission420/#");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

static esp_err_t index_handler(httpd_req_t *req){
  httpd_resp_set_type(req, "text/html");
  return httpd_resp_send(req, (const char *)INDEX_HTML, strlen(INDEX_HTML));
}

static esp_err_t stream_handler(httpd_req_t *req){
  camera_fb_t * fb = NULL;
  esp_err_t res = ESP_OK;
  size_t _jpg_buf_len = 0;
  uint8_t * _jpg_buf = NULL;
  char * part_buf[64];

  res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);
  if(res != ESP_OK){
    return res;
```

```
}

while(true){
  fb = esp_camera_fb_get();
  if (!fb) {
    Serial.println("Camera capture failed");
    res = ESP_FAIL;
  } else {
    if(fb->width > 400){
      if(fb->format != PIXFORMAT_JPEG){
        bool jpeg_converted = frame2jpg(fb, 80, &_jpg_buf, &_jpg_buf_len);
        esp_camera_fb_return(fb);
        fb = NULL;
        if(!jpeg_converted){
          Serial.println("JPEG compression failed");
          res = ESP_FAIL;
        }
      } else {
        _jpg_buf_len = fb->len;
        _jpg_buf = fb->buf;
      }
    }
  }
  if(res == ESP_OK){
    size_t hlen = snprintf((char *)part_buf, 64, _STREAM_PART, _jpg_buf_len);
    res = httpd_resp_send_chunk(req, (const char *)part_buf, hlen);
  }
  if(res == ESP_OK){
```

```
      res = httpd_resp_send_chunk(req, (const char *)_jpg_buf, _jpg_buf_len);
    }
    if(res == ESP_OK){
      res = httpd_resp_send_chunk(req, _STREAM_BOUNDARY,
strlen(_STREAM_BOUNDARY));
    }
    if(fb){
      esp_camera_fb_return(fb);
      fb = NULL;
      _jpg_buf = NULL;
    } else if(_jpg_buf){
      free(_jpg_buf);
      _jpg_buf = NULL;
    }
    if(res != ESP_OK){
      break;
    }
    //Serial.printf("MJPG: %uB\n",(uint32_t)(_jpg_buf_len));
  }
  return res;
}


static esp_err_t cmd_handler(httpd_req_t *req){
  char*  buf;
  size_t buf_len;
  char variable[32] = {0,};
   buf_len = httpd_req_get_url_query_len(req) + 1;
  if (buf_len > 1) {
    buf = (char*)malloc(buf_len);
```

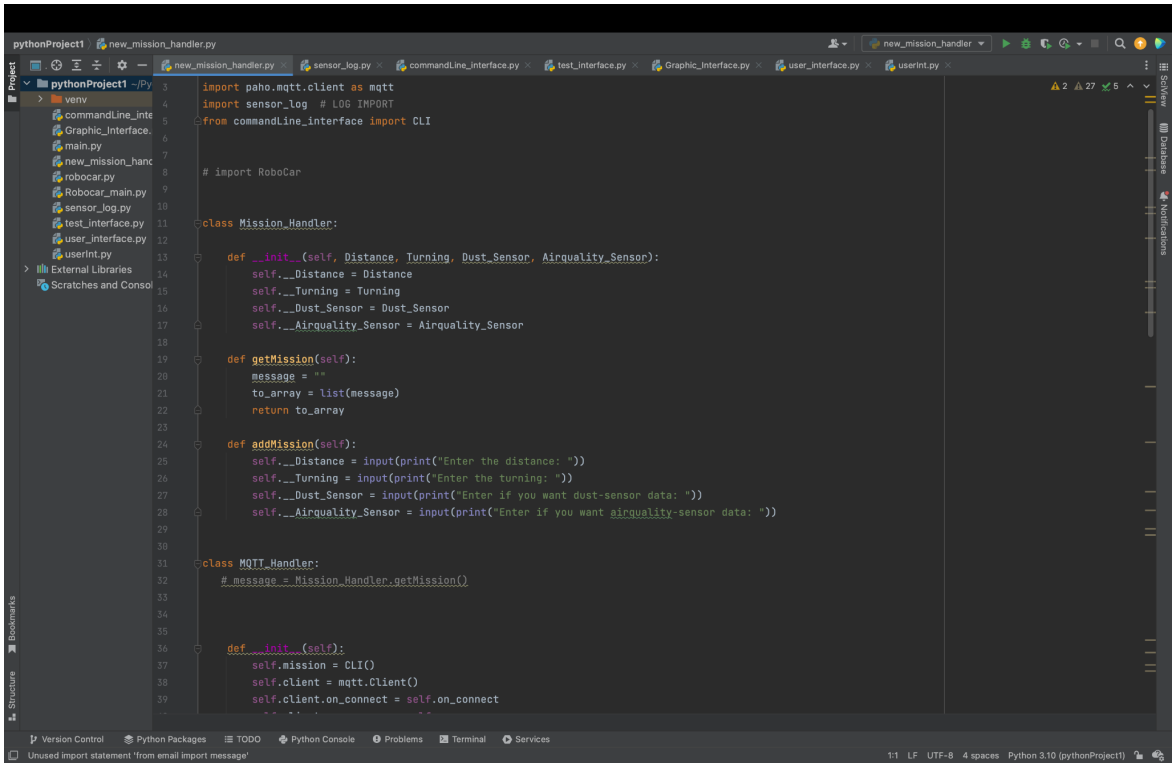
```
    if(!buf){
      httpd_resp_send_500(req);
      return ESP_FAIL;
    }
    if (httpd_req_get_url_query_str(req, buf, buf_len) == ESP_OK) {
      if (httpd_query_key_value(buf, "go", variable, sizeof(variable)) == ESP_OK)
{
      } else {
        free(buf);
        httpd_resp_send_404(req);
        return ESP_FAIL;
      }
    } else {
      free(buf);
      httpd_resp_send_404(req);
      return ESP_FAIL;
    }
    free(buf);
  } else {
    httpd_resp_send_404(req);
    return ESP_FAIL;
  }

  sensor_t * s = esp_camera_sensor_get();
  int res = 0;


  if(res){
    return httpd_resp_send_500(req);
```

```
 }

 httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
 return httpd_resp_send(req, NULL, 0);
}

void startCameraServer(){
 httpd_config_t config = HTTPD_DEFAULT_CONFIG();
 config.server_port = 80;
 httpd_uri_t index_uri = {
   .uri       = "/",
   .method    = HTTP_GET,
   .handler   = index_handler,
   .user_ctx  = NULL
 };

 httpd_uri_t cmd_uri = {
   .uri       = "/action",
   .method    = HTTP_GET,
   .handler   = cmd_handler,
   .user_ctx  = NULL
 };
 httpd_uri_t stream_uri = {
   .uri       = "/stream",
   .method    = HTTP_GET,
   .handler   = stream_handler,
   .user_ctx  = NULL
 };
```

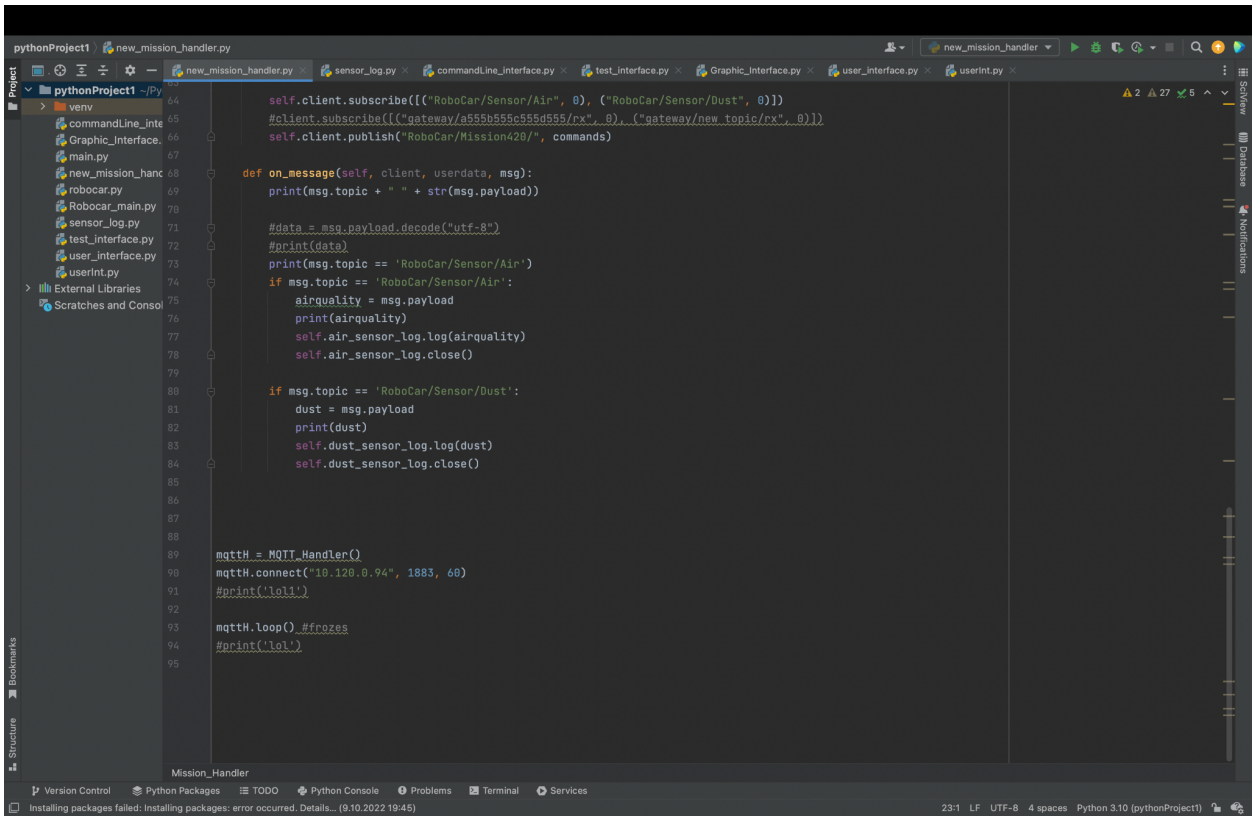
```
  if (httpd_start(&camera_httpd, &config) == ESP_OK) {
    httpd_register_uri_handler(camera_httpd, &index_uri);
    httpd_register_uri_handler(camera_httpd, &cmd_uri);
  }
  config.server_port += 1;
  config.ctrl_port += 1;
  if (httpd_start(&stream_httpd, &config) == ESP_OK) {
    httpd_register_uri_handler(stream_httpd, &stream_uri);
  }
}
```

10.3 Python code:

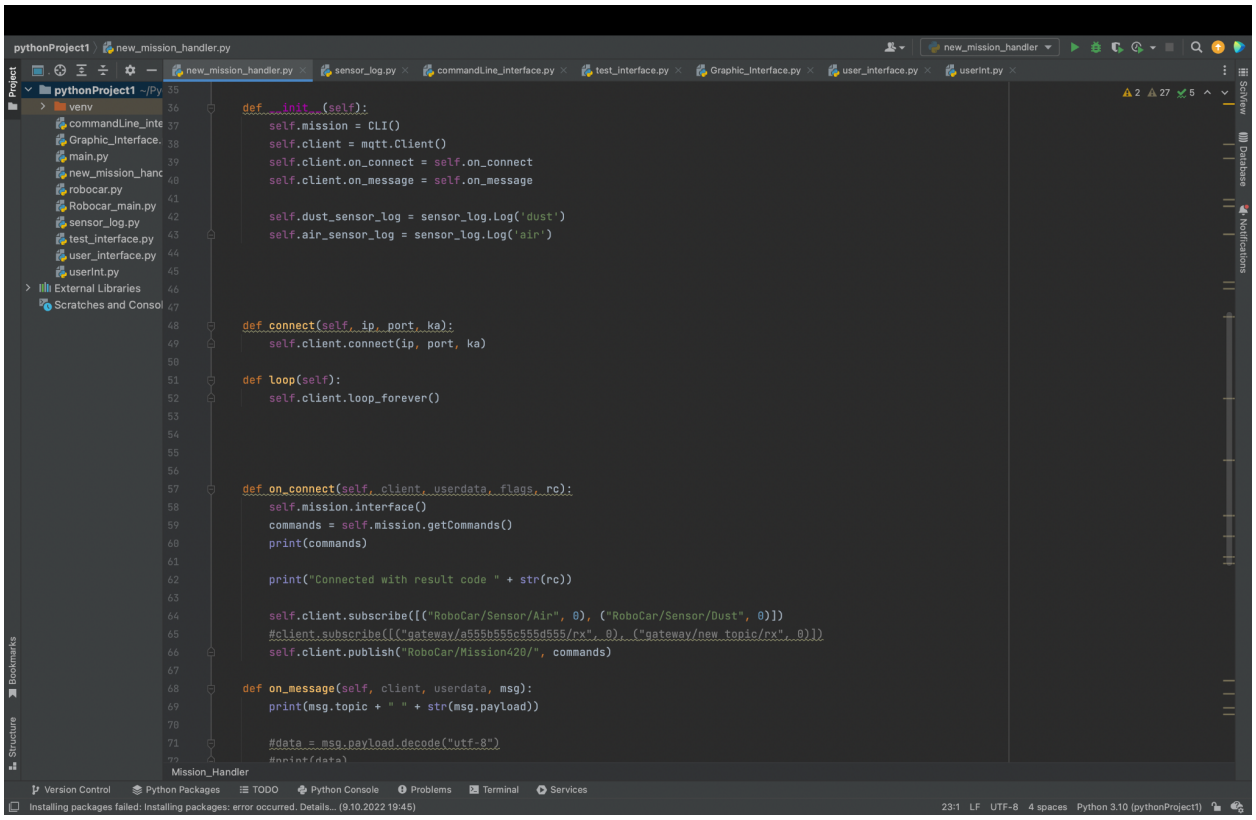
MQTT handler:



```
import paho.mqtt.client as mqtt
import sensor_log  # LOG IMPORT
from commandLine_interface import CLI


# import RoboCar


class Mission_Handler:

    def __init__(self, Distance, Turning, Dust_Sensor, Airquality_Sensor):
        self.__Distance = Distance
        self.__Turning = Turning
        self.__Dust_Sensor = Dust_Sensor
        self.__Airquality_Sensor = Airquality_Sensor

    def getMission(self):
        message = ""
        to_array = list(message)
        return to_array

    def addMission(self):
        self.__Distance = input(print("Enter the distance: "))
        self.__Turning = input(print("Enter the turning: "))
        self.__Dust_Sensor = input(print("Enter if you want dust-sensor data: "))
        self.__Airquality_Sensor = input(print("Enter if you want airquality-sensor data: "))


class MQTT_Handler:
    # message = Mission_Handler.getMission()



    def __init__(self):
        self.mission = CLI()
        self.client = mqtt.Client()
        self.client.on_connect = self.on_connect
```

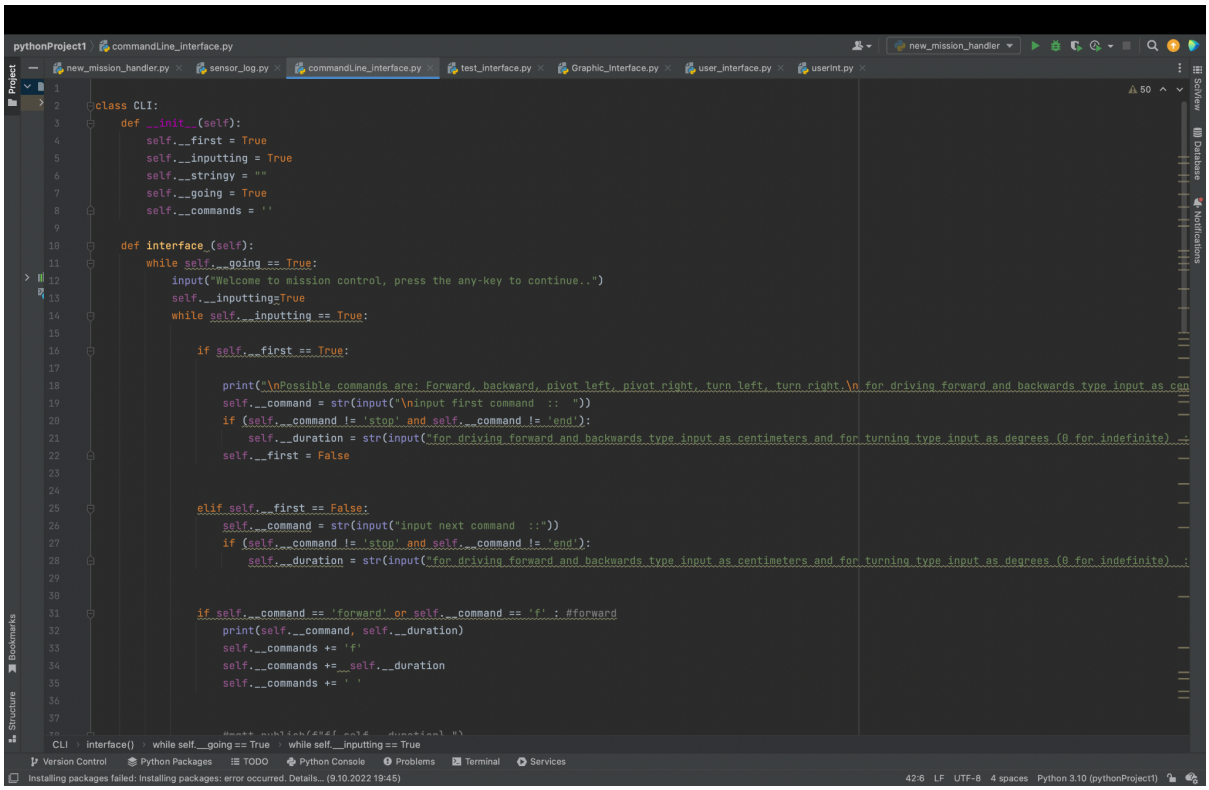


```
            self.client.subscribe([("RoboCar/Sensor/Air", 0), ("RoboCar/Sensor/Dust", 0)])
            #client.subscribe([("gateway/a555b555c555d555/rx", 0), ("gateway/new_topic/rx", 0)])
            self.client.publish("RoboCar/Mission420/", commands)

        def on_message(self, client, userdata, msg):
            print(msg.topic + " " + str(msg.payload))

            #data = msg.payload.decode("utf-8")
            #print(data)
            print(msg.topic == 'RoboCar/Sensor/Air')
            if msg.topic == 'RoboCar/Sensor/Air':
                airquality = msg.payload
                print(airquality)
                self.air_sensor_log.log(airquality)
                self.air_sensor_log.close()

            if msg.topic == 'RoboCar/Sensor/Dust':
                dust = msg.payload
                print(dust)
                self.dust_sensor_log.log(dust)
                self.dust_sensor_log.close()




    mqttH = MQTT_Handler()
    mqttH.connect("10.120.0.94", 1883, 60)
    #print('lol1')

    mqttH.loop() #frozes
    #print('lol')
```

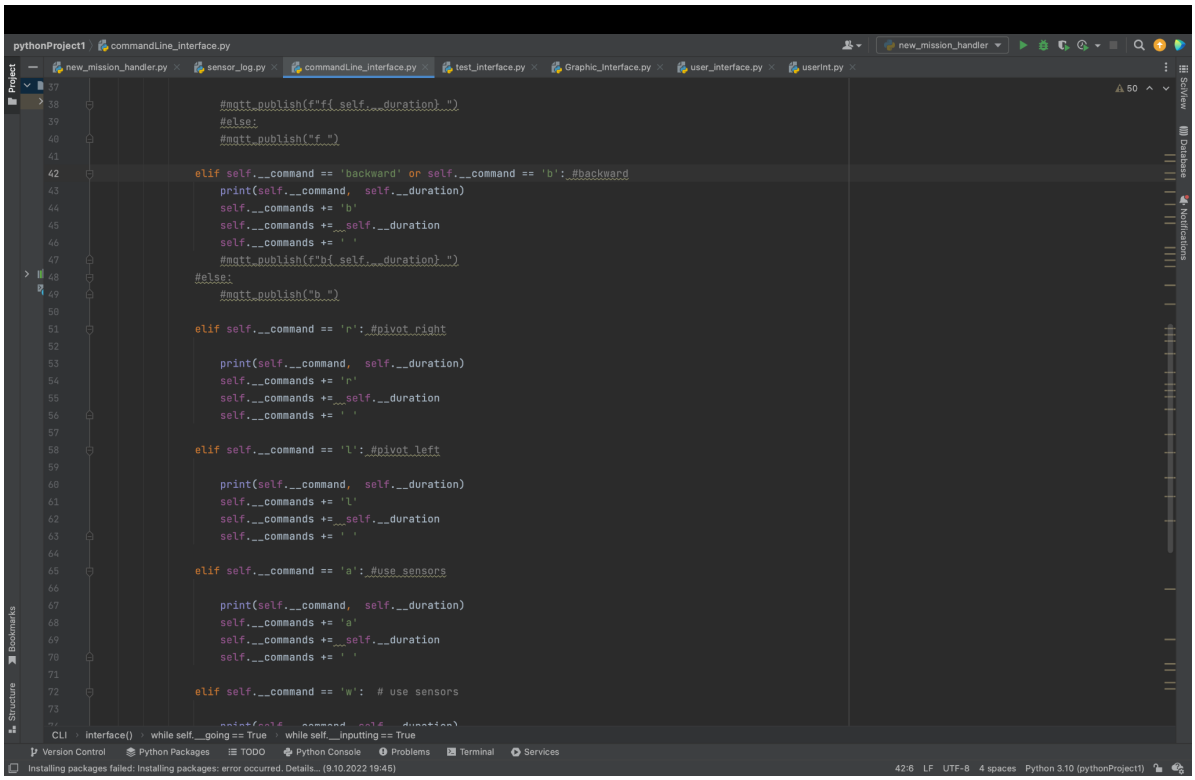


```
        def __init__(self):
            self.mission = CLI()
            self.client = mqtt.Client()
            self.client.on_connect = self.on_connect
            self.client.on_message = self.on_message

            self.dust_sensor_log = sensor_log.Log('dust')
            self.air_sensor_log = sensor_log.Log('air')




        def connect(self, ip, port, ka):
            self.client.connect(ip, port, ka)

        def loop(self):
            self.client.loop_forever()




        def on_connect(self, client, userdata, flags, rc):
            self.mission.interface()
            commands = self.mission.getCommands()
            print(commands)

            print("Connected with result code " + str(rc))

            self.client.subscribe([("RoboCar/Sensor/Air", 0), ("RoboCar/Sensor/Dust", 0)])
            #client.subscribe([("gateway/a555b555c555d555/rx", 0), ("gateway/new_topic/rx", 0)])
            self.client.publish("RoboCar/Mission420/", commands)

        def on_message(self, client, userdata, msg):
            print(msg.topic + " " + str(msg.payload))

            #data = msg.payload.decode("utf-8")
            #print(data)
```

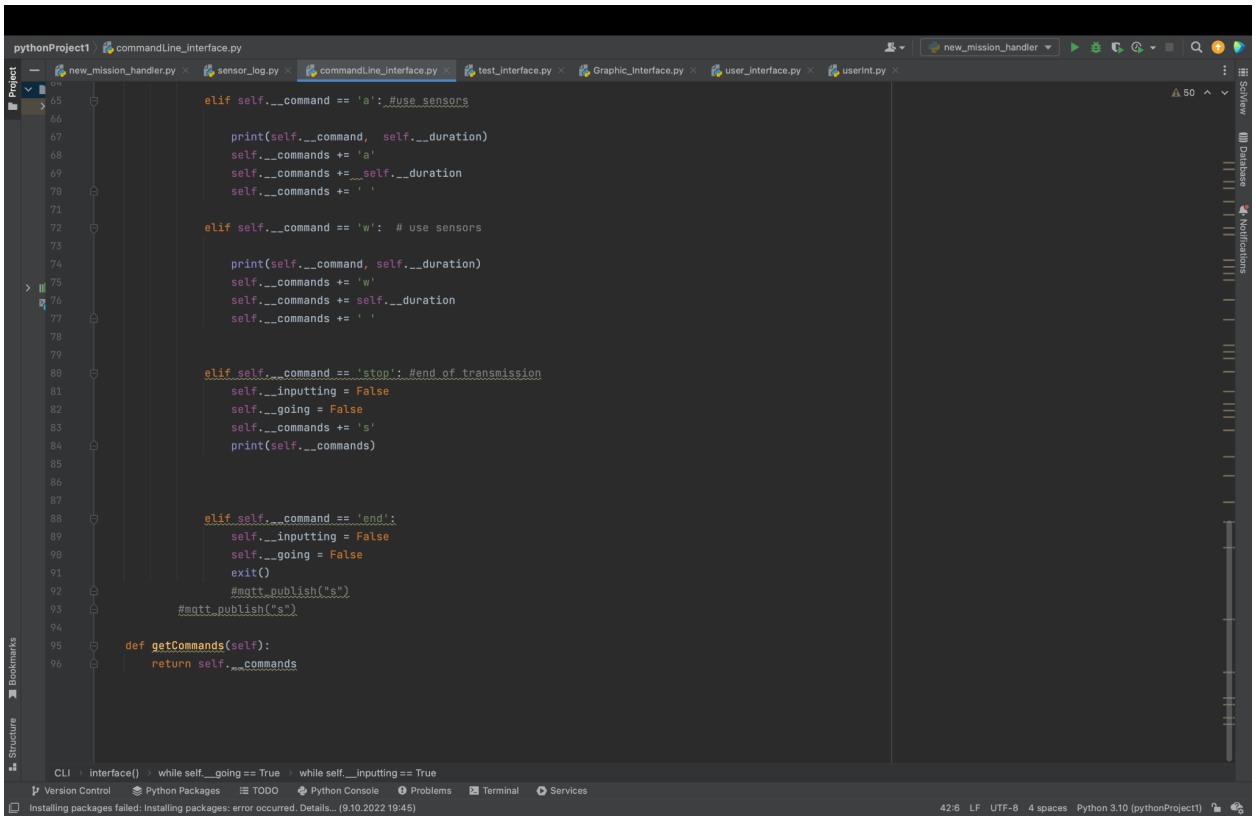
Interface:



```
class CLI:
    def __init__(self):
        self.__first = True
        self.__inputting = True
        self.__stringy = ""
        self.__going = True
        self.__commands = ''

    def interface(self):
        while self.__going == True:
            input("Welcome to mission control, press the any-key to continue..")
            self.__inputting=True
            while self.__inputting == True:

                if self.__first == True:

                    print("\nPossible commands are: Forward, backward, pivot left, pivot right, turn left, turn right,\n for driving forward and backwards type input as cen
                    self.__command = str(input("\ninput first command  ::  "))
                    if (self.__command != 'stop' and self.__command != 'end'):
                        self.__duration = str(input("for driving forward and backwards type input as centimeters and for turning type input as degrees (0 for indefinite) :
                    self.__first = False


                elif self.__first == False:
                    self.__command = str(input("input next command  ::"))
                    if (self.__command != 'stop' and self.__command != 'end'):
                        self.__duration = str(input("for driving forward and backwards type input as centimeters and for turning type input as degrees (0 for indefinite)  :


                if self.__command == 'forward' or self.__command == 'f' : #forward
                    print(self.__command, self.__duration)
                    self.__commands += 'f'
                    self.__commands += self.__duration
                    self.__commands += ' '
```

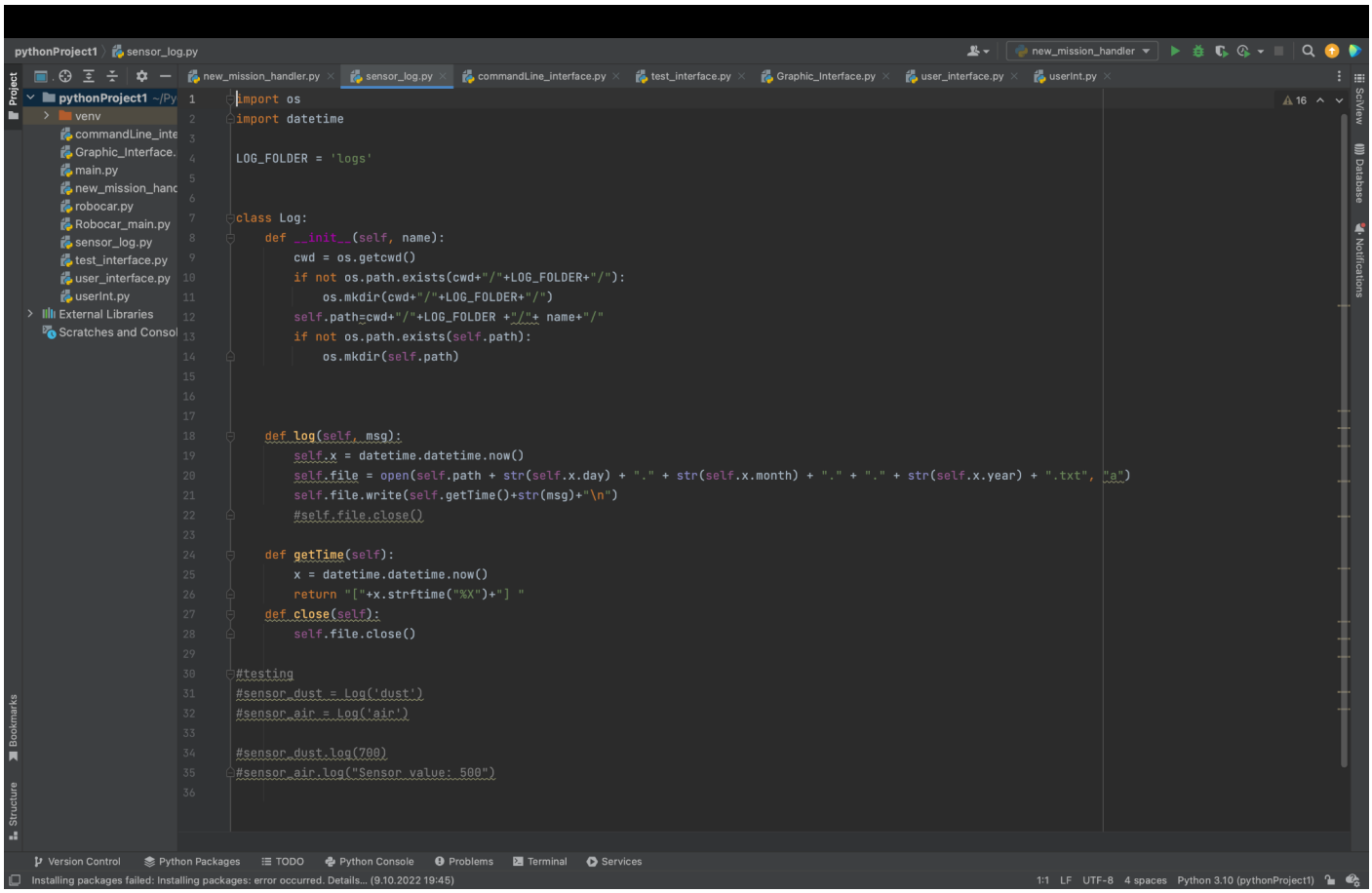


```
                    #mqtt_publish(f"f{ self.__duration} ")
                    #else:
                    #mqtt_publish("f ")

                elif self.__command == 'backward' or self.__command == 'b': #backward
                    print(self.__command,  self.__duration)
                    self.__commands += 'b'
                    self.__commands += self.__duration
                    self.__commands += ' '
                    #mqtt_publish(f"b{ self.__duration} ")
                #else:
                    #mqtt_publish("b ")

                elif self.__command == 'r': #pivot right

                    print(self.__command,  self.__duration)
                    self.__commands += 'r'
                    self.__commands += self.__duration
                    self.__commands += ' '

                elif self.__command == 'l': #pivot left

                    print(self.__command,  self.__duration)
                    self.__commands += 'l'
                    self.__commands += self.__duration
                    self.__commands += ' '

                elif self.__command == 'a': #use sensors

                    print(self.__command,  self.__duration)
                    self.__commands += 'a'
                    self.__commands += self.__duration
                    self.__commands += ' '

                elif self.__command == 'w':  # use sensors
```



```
                elif self.__command == 'a': #use sensors

                    print(self.__command,  self.__duration)
                    self.__commands += 'a'
                    self.__commands += self.__duration
                    self.__commands += ' '

                elif self.__command == 'w':  # use sensors

                    print(self.__command, self.__duration)
                    self.__commands += 'w'
                    self.__commands += self.__duration
                    self.__commands += ' '


                elif self.__command == 'stop': #end of transmission
                    self.__inputting = False
                    self.__going = False
                    self.__commands += 's'
                    print(self.__commands)



                elif self.__command == 'end':
                    self.__inputting = False
                    self.__going = False
                    exit()
                    #mqtt_publish("s")
            #mqtt_publish("s")

    def getCommands(self):
        return self.__commands
```

Sensor Log:


pythonProject1
sensor_log.py
new_mission_handler
new_mission_handler.py
sensor_log.py
commandLine_interface.py
test_interface.py
Graphic_Interface.py
user_interface.py
userInt.py
pythonProject1 ~/Py
venv
commandLine_inte
Graphic_Interface.
main.py
new_mission_hanc
robocar.py
Robocar_main.py
sensor_log.py
test_interface.py
user_interface.py
userInt.py
External Libraries
Scratches and Consol
import os
import datetime
LOG_FOLDER = 'logs'
class Log:
def __init__(self, name):
cwd = os.getcwd()
if not os.path.exists(cwd+"/"+LOG_FOLDER+"/"):
os.mkdir(cwd+"/"+LOG_FOLDER+"/")
self.path=cwd+"/"+LOG_FOLDER +"/"+ name+"/"
if not os.path.exists(self.path):
os.mkdir(self.path)
def log(self, msg):
self.x = datetime.datetime.now()
self.file = open(self.path + str(self.x.day) + "." + str(self.x.month) + "." + str(self.x.year) + ".txt", "a")
self.file.write(self.getTime()+str(msg)+"\n")
#self.file.close()
def getTime(self):
x = datetime.datetime.now()
return "["+x.strftime("%X")+"] "
def close(self):
self.file.close()
#testing
#sensor_dust = Log('dust')
#sensor_air = Log('air')
#sensor_dust.log(700)
#sensor_air.log("Sensor value: 500")
Version Control
Python Packages
TODO
Python Console
Problems
Terminal
Services
Installing packages failed: Installing packages: error occurred. Details... (9.10.2022 19:45)
1:1
LF
UTF-8
4 spaces
Python 3.10 (pythonProject1)