MIDDLE EAST TECHNICAL UNIVERSITY

ELECTRICAL-ELECTRONICS ENGINEERING

EE400-REPORT

COMPANY NAME	OPUS EMBEDDED LTD.
COMPANY ADDRESS	[TEKNOPARK ANKARA] İVEDIKOSB MH. 2224. CD. NO:1 B BLOK, BUZ-20, YENIMAHALLE, ANKARA
STARTING DATE	22.01.2024
ENDING DATE	16.02.2024
SPECIALIZATION AREA	COMPUTER

Bilal Yuksu
2110047
yuksu.bilal@metu.edu.tr
05454531954

Student Signature

Emre Canbolat
emre.canbolat@opusembedded.com
+90 (534) 598 98 84

Supervisor Signature

Contents

About Company

Name of the Company

OPUS-Embedded LTI.

Location of the Company

[TEKNOPARK ANKARA] İVEDIKOSB MH.
2224. CD. NO:1 B BLOK, BUZ-20,
YENIMAHALLE, ANKARA

Contact

Web Address: *https://opusembedded.com*

E-Mail: *technical@opusembedded.com*

Phone: *+90 534 598 98 84*

General Description and Brief History of the Company

OPUS Embedded is a dynamic company specializing in embedded software and digital design. At the forefront of their endeavors is the development of an innovative AHRS (Attitude and Heading Reference System) device, alongside ongoing research into its refinement and diverse applications. Currently comprising a team of three dedicated electrical-electronics engineers, OPUS Embedded operates from its headquarters located in Teknopark Ankara.

Figure 0. Company Logo

Established in 2020 by Emre Canbolat, my internship supervisor and a 2016 graduate from METU Electrical and Electronics Engineering, OPUS Embedded swiftly gained momentum. Beginning as a solo endeavor, Emre's vision soon attracted Ferhat Gölbol, another 2016 METU EE graduate, who assumed the role of head of engineering within the company.

Operating within the embedded systems domain, OPUS Embedded focuses on two primary avenues:

1. Exclusive Design & Production:

As an esteemed solution partner, OPUS excels in custom electronic product design and manufacturing, effectively addressing client needs with cutting-edge technology. OPUS Embedded offers comprehensive design, manufacture, and upgrade responsibility, competitive pricing, engineering expertise, dedicated support, and long-term warranties.

2. Development of Pioneer OPUS Products:

OPUS Embedded is committed to pioneering innovative products, including:

- OPUS-Inertial: An advanced AHRS device and associated evaluation kits.
- OPUS-NINS: A forthcoming GPS/INS solution.
- OPUS-Route: A device dedicated to route control.

Through their dedication to technological advancement and commitment to excellence, OPUS Embedded continues to make significant strides in the field of embedded systems.

ESP32-based Navigation System with GPS and AHRS Integration

Introduction

During my internship, I had the opportunity to work on a fascinating project that involved creating a highly portable and versatile navigation system using ESP32. The main objective of this project is to develop a navigation system that utilizes angles obtained from AHRS and GPS to provide efficient route guidance within predefined tolerances. Such a system is commonly used in outdoor recreational activities, aviation, marine navigation, passenger and goods transportation and even autonomous agriculture (Wieser, 2003) [13]. In particular, the location of agricultural vehicles can be displayed in real-time on a visual route planning interface based on Google maps, obtained via GPS, and used for autonomous farming (Ankaralı, 2020) [9]. It has been stated that it can autonomously guide a robot through crop rows by using a navigation system to direct autonomous vehicles in agricultural areas where crop rows are difficult to detect (Corke, 2014) [12]. Additionally, these navigation systems are employed to integrate GPS orientation data obtained from an ultra-short baseline system with vehicle-grade gyroscopes, enabling pilots to receive instantaneous positioning information within an aircraft's control system (Powell, 1996) [11]. The system was designed to determine a device's position and orientation accurately, utilizing GPS (Global Positioning System) and AHRS (Attitude and Heading Reference System) functionalities. To achieve this goal, I utilized ESP32, Teseo-Liv3f, and OPUS-Inertial, which is the company's own product. I wrote algorithms in C and Python languages to provide precise location and orientation data.

At the request of my supervisor, the navigation system was intended to be user-friendly and easy to use. To meet this requirement, I used various libraries in my Python code to create an interactive map navigation system. The code uses IPython widgets, geopy, gmaps, and threading libraries to enable real-time communication with a server, receive location data, and update markers and paths on a map dynamically. Additionally, the code implements navigation instructions based on received angle, longitude, latitude values and predefined tolerances to provide efficient route guidance. The system has a user-friendly GUI, which offers an intuitive interface for users to interact with the map, add markers, and receive live navigation feedback.

On the other hand, I wrote C code that communicates with Teseo-Liv3f and OPUS-Inertial via I2C communication to take latitude, longitude, and yaw angle data. Moreover, the calibration setting for OPUS-Inertial which can be adjusted by the user is added as an option into the code. Overall, the project provided an exceptional learning opportunity, and the resulting ESP32-based navigation system proved to be quite accurate, reliable, and efficient. Its performance makes it an ideal solution for a wide range of applications, including outdoor activities, aviation, and marine navigation.

Project Overview

In order to determine the user's position and orientation, a practical and effective navigation solution is devised by leveraging accessible hardware components and meticulous design. The project comprised various essential components and functionalities, which are listed below:

ESP32 Microcontroller: The navigation system project relies on the ESP32 microcontroller which is a highly versatile processor that serves as the project's brain. This microcontroller illustrated in Figure 1 provides the necessary power and connectivity to connect with external sensors and communicate with other devices [4]. It features built-in Wi-Fi that allows seamless wireless communication with servers and other devices. By using I2C, the ESP32 microcontroller connects to sensors such as the Teseo-Liv3f GPS module and the OPUS-Inertial AHRS sensor, which enables accurate positioning and orientation determination. All in all, the ESP32 microcontroller is the most essential component in creating a highly functional navigation system.

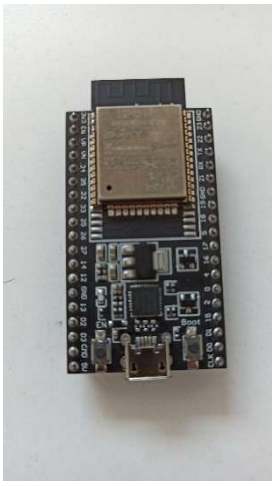


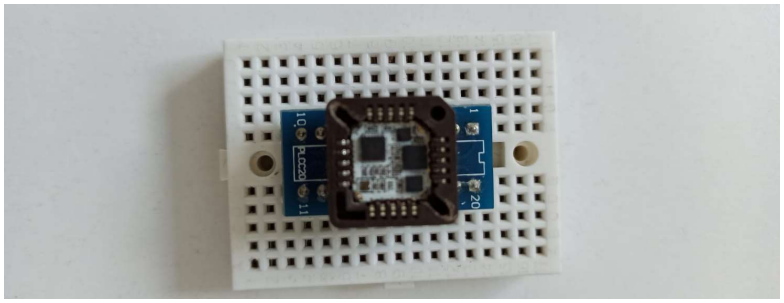
Figure 1. ESP32

GPS Module (Teseo-LIV3F): The Teseo-LIV3F is a module that uses the Global Navigation Satellite System and contains an independent positioning receiver IC. It is shown in Figure 2 [1]. Teseo-LIV3F is a versatile GPS module that can be used in several applications including public transportation, insurance, product tracking, drones, vehicle tracking, emergency calls, fleet management, people and pet location, and anti-theft systems. To obtain latitude and longitude coordinates, the Teseo-LIV3F GPS module was used, and the ESP32 received real-time location data by communicating with the GPS module through I2C.

Figure 2.Teseo Liv3f

Figure 3. OPUS-Inertial

AHRS (Attitude and Heading Reference System): The navigation system includes an AHRS sensor which helps to determine the orientation of the device about the Earth's magnetic north. Integral to this project, OPUS-Inertial, which is shown in Figure 3 boasts several essential features offered by the company to its clients. These encompass the delivery of accurate RAW inertial data and Roll/Pitch/Yaw information, a high data output rate, compact dimensions (around 100 mm²), compatibility with industry-standard interfaces (SPI/I2C/UART), ultra-low power consumption design, and an intuitive firmware interface ensuring effortless device access and control [6]. Roll, pitch and yaw angles are illustrated in Figure 4. The yaw angle we will use for our project will be taken from AHRS. The AHRS data was obtained through I2C communication with the ESP32.



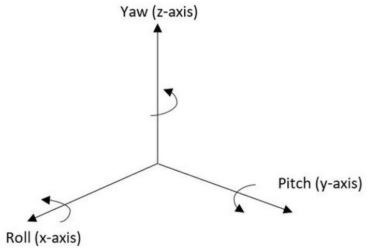


Figure 4. OPUS-Inertial

Integration and Navigation Algorithms: The ESP32 processed the incoming GPS and AHRS data and implemented algorithms to calculate position, orientation, and navigation instructions. These algorithms included distance calculation, angle adjustment, and navigation decision-making based on predefined tolerances.

Project Tasks and Achievements

Hardware Setup

In order to accurately receive yaw values from AHRS and send them to the TCP client-server, first of all, I followed the setup instructions outlined in Figure 5. This involved connecting the ESP32 microcontroller and OPUS-Inertial. Additionally, the pinouts for OPUS-Inertial P-20 can be found in

Figure 6.

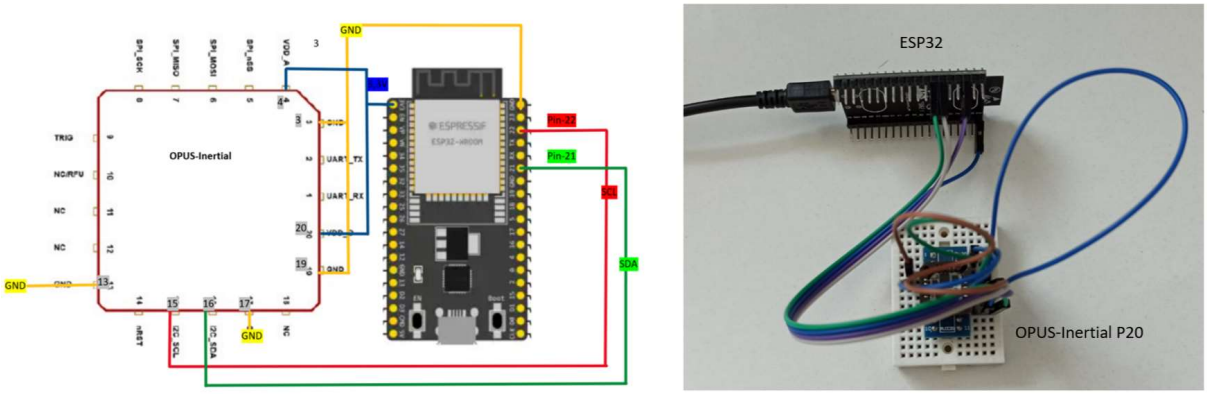


Figure 5. Connection diagram of ESP32 and OPUS-Inertial

#	İsim	Tip	Açıklama
1	UART_RX	DI	UART Receive Data input. 115200 baud rate.[1]
2	UART_TX	DO	UART Transmit Data output. 115200 baud rate.[1]
3	GND	GND	Ground.
4	VDD_A	PWR	Analog power input. 1.71V < VDD_A < 3.63V. 1µF to 10µF bypass capacitor is recommended.
5	SPI_nSS	DI	SPI Slave Select input.
6	SPI_MOSI	DI	SPI MOSI input.
7	SPI_MISO	DO	SPI MISO output.
8	SPI_SCK	DI	SPI Clock input.
9	TRIG	DI	Trigger input.
10	NC/RFU	-	Do not connect to this pin.
11	NC	-	Do not connect to this pin.
12	NC	-	Do not connect to this pin.
13	GND	GND	Ground.
14	nRST	DI	Reset input. Active low (Weak pull-up internally)
15	I2C_SCL	DI	I2C Clock pin. 2.2kΩ to 4.7kΩ pull-up resistor is recommended.
16	I2C_SDA	DIO	I2C Data pin. 2.2kΩ to 4.7kΩ pull-up resistor is recommended.
17	GND	GND	Ground.
18	NC	-	Do not connect to this pin.
19	GND	GND	Ground.
20	VDD_D	PWR	Digital power input. 1.71V < VDD_D < 3.63V. 1µF to 10µF bypass capacitor is recommended. *Note: VDD_A must be equal or greater than VDD_D*

Figure 6. OPUS-Inertial P-20 Pinouts

After reading the yaw angle correctly, using a pre-prepared module, the necessary I2C connections of ESP32, AHRS and GPS were made in this module. It was aimed to get the yaw value from AHRS as well as the latitude and longitude values from GPS.

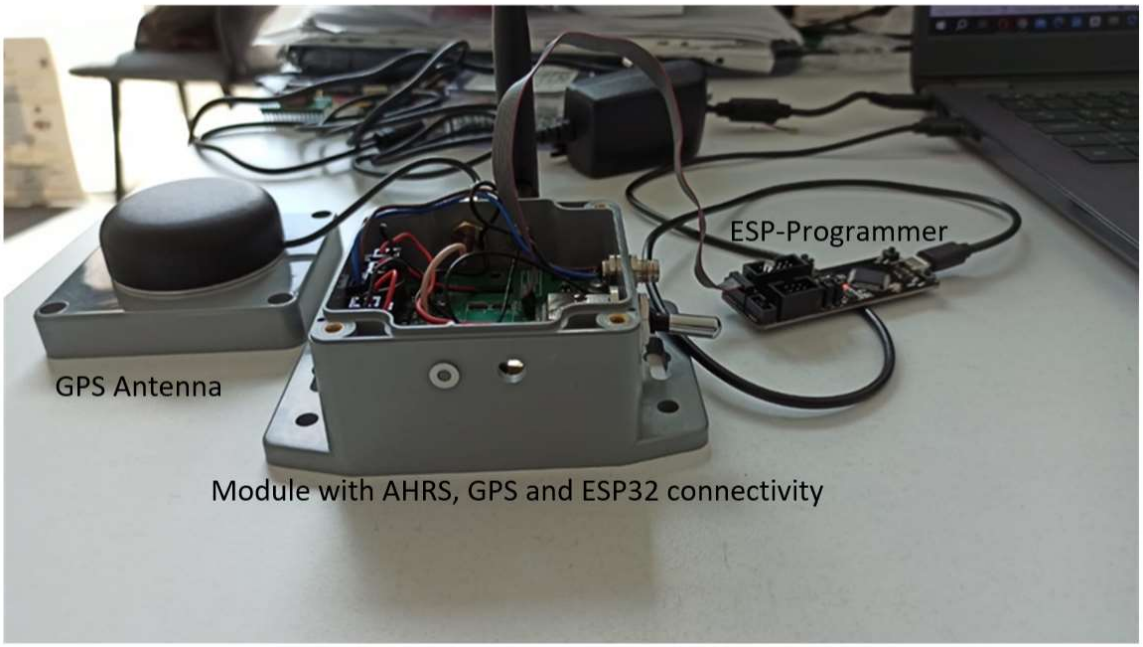


Figure 7. Uploading code via ESP Programmer

After writing the necessary program on Arduino and exporting the compiled binary, our code is uploaded into the microcontroller with the help of ESP-Prog, as shown in Figure 7 and Figure 8.

Figure 8. Module with AHRS, GPS, and ESP32

Software Design - Part 1 (Python Implementation)

First of all, my advisor asked me to write a Python code on Jupyter Notebook to connect to a TCP Client Server. After establishing the TCP connection, I attempted to send and receive data through it. As an example, latitude, longitude and angle as 39.99, 32.74 and 30.00 respectively are sent from TCP Client Server after the connection as shown in Figure 9 and Figure 10.

Figure 9. TCP Client Server

Subsequently, my advisor suggested that I delve into the gmaps library, which is a plugin for Jupyter Notebooks that enables the integration of Google maps. Its primary objective is to facilitate the visualization and manipulation of geographical data. After studying gmaps documentation, I tried to open a map and get the location information of the place which I clicked on the map with the mouse. Then, related locations are marked on the maps and blue lines are drawn between them. *'gmaps.Symbol'* is used for marking a location and *'gmaps.drawing_layer'* is used for drawing line between two locations. Additionally, when you click on the marked location, location information appears as a pop-up thanks to gmaps.symbol_layer(locations, hover_text=names). Also, black dot on the map shows our location and a red line is drawn from the black dot to the first location as shown in Figure 10. The red line and black dot which shows our location is updated on the map whenever data comes from TCP Client-Server.

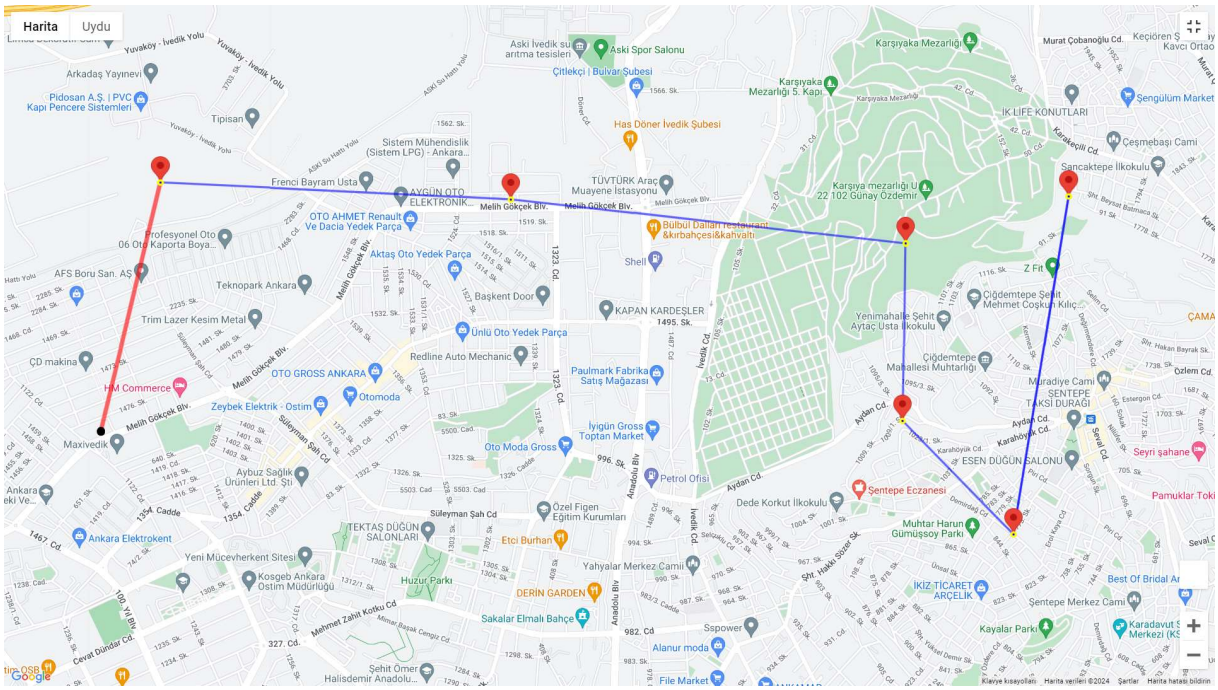


Figure 10. marked locations and lines between them on Google Maps

Furthermore, I calculated the angle between the red line and the north for orientation, and navigation instructions. The related calculation is presented in (1) by using the bearing angle "θ" formula. Latitude, longitude and bearing angle are given in radian in the formula. The difference between the calculated angle and the yaw value obtained from AHRS will give us turning instructions.

$$\Delta_{Lon} = Lon_2 - Lon_1$$

$$x = \sin \Delta_{Lon} \cos Lat_2$$

$$y = \cos Lat_1 \sin Lat_2 - (\sin Lat_1 \cos Lat_2 \cos \Delta_{Lon})$$

$$\theta = \arctan(x, y) \quad (1)$$

In addition, according to the location information received (instant location information), we need to find the possible closest distance to the destination. This distance is calculated and printed on the screen. For the distance calculation Haversine Formula (2) is used, d is the distance between two locations with longitude "ψ" and latitude "φ" and R is the radius of the Earth [10].

$$d = 2.R.\sin^{-1}\left(\sqrt{\sin^2\left(\frac{\varphi_2-\varphi_1}{2}\right) + \cos\varphi_1 \cos\varphi_2 \sin^2\left(\frac{\psi_2-\psi_1}{2}\right)}\right) \quad (2)$$

In addition, the yaw angle from AHRS and the latitude and longitude information from GPS will be printed on the screen and the necessary commands will be given thanks to this information. This printed information will be refreshed on the screen every second. Because the yaw value to be taken has been adjusted according to this time and can be changed.

Figure 11. latitude, longitude and angle are sent from the server

For example, if we examine Figures 11 and 12, our current location is marked with a black dot on the map according to the latitude, longitude and angle information sent from the TCP client-server, and a red line is placed from this black dot to the possible first destination. Additionally, as shown in Figure 12, Google Maps, the total distance, our current location, and instructions are updated and printed on the screen.

```
Connected to the server.
###############################################

'Return LEFT at angle: 39.76168679069413 '

###############################################

'Latitude: 39.9945'

'Longitude: 32.74011'

'Angle coming from AHRS: 45.0'

###############################################

'Minimum distance to the final location: 924.731101984042 m'
```

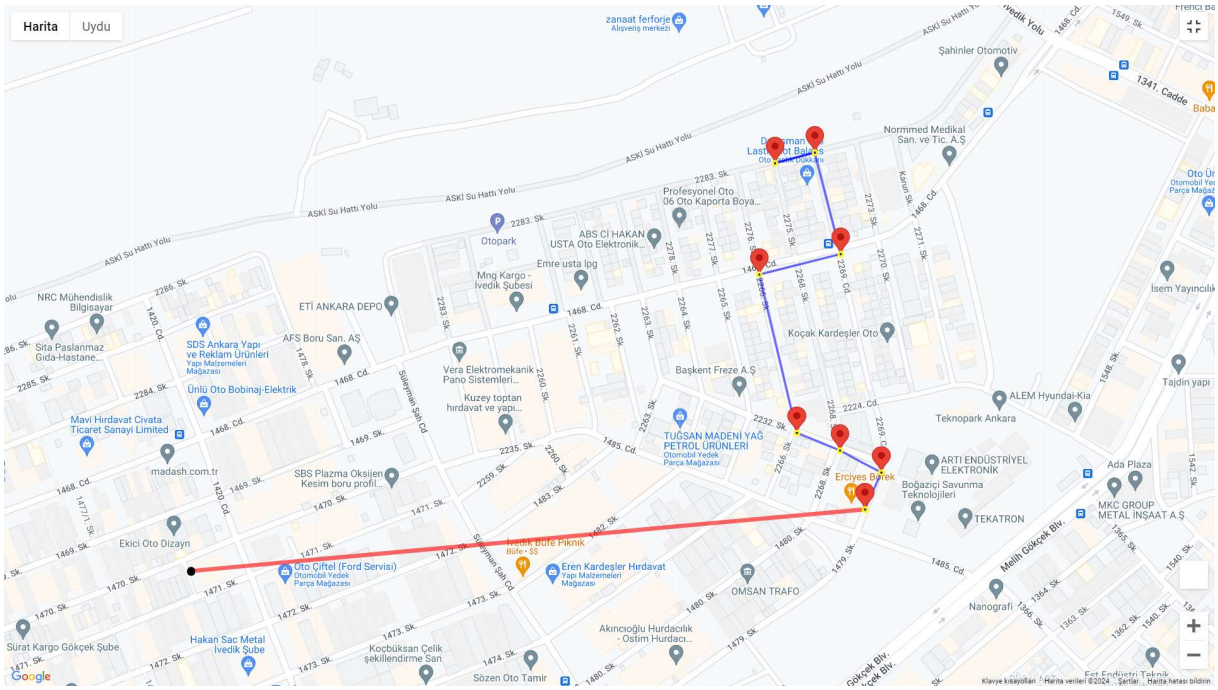


Figure 12. marked locations on google maps, instant location information and instructions.

The flowchart for the explanation mentioned earlier can be found in Figure 13, while the Python code corresponding to the same is available in Appendix B. Please note that the provided code requires an API-KEY for proper functionality.

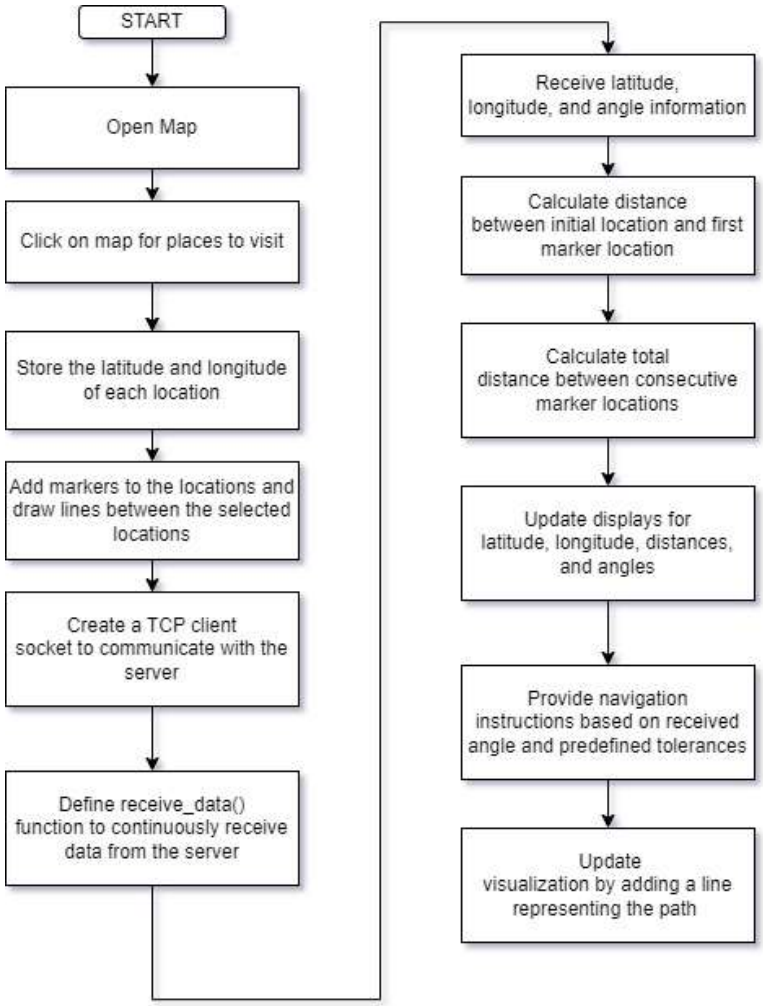


Figure 13. Flowchart of the python code

Software Design - Part 2 (Arduino Implementation)

In this section, the process of establishing communication among the ESP32, Teseo-Liv3f, and OPUS-Inertial devices through the I2C protocol will be elucidated. Subsequently, I will provide an explanation of how the data that has been gathered is transmitted to a server using Wi-Fi. Initially, an examination of pertinent libraries such as "WiFi.h", "Arduino.h", "Wire.h", and "MicroNMEA.h" was conducted, guided by the supervisor's recommendations. Additionally, extensive review of datasheets related to "OPUS-Inertial", "ESP32", and "Teseo-Liv3f" provided essential insights.

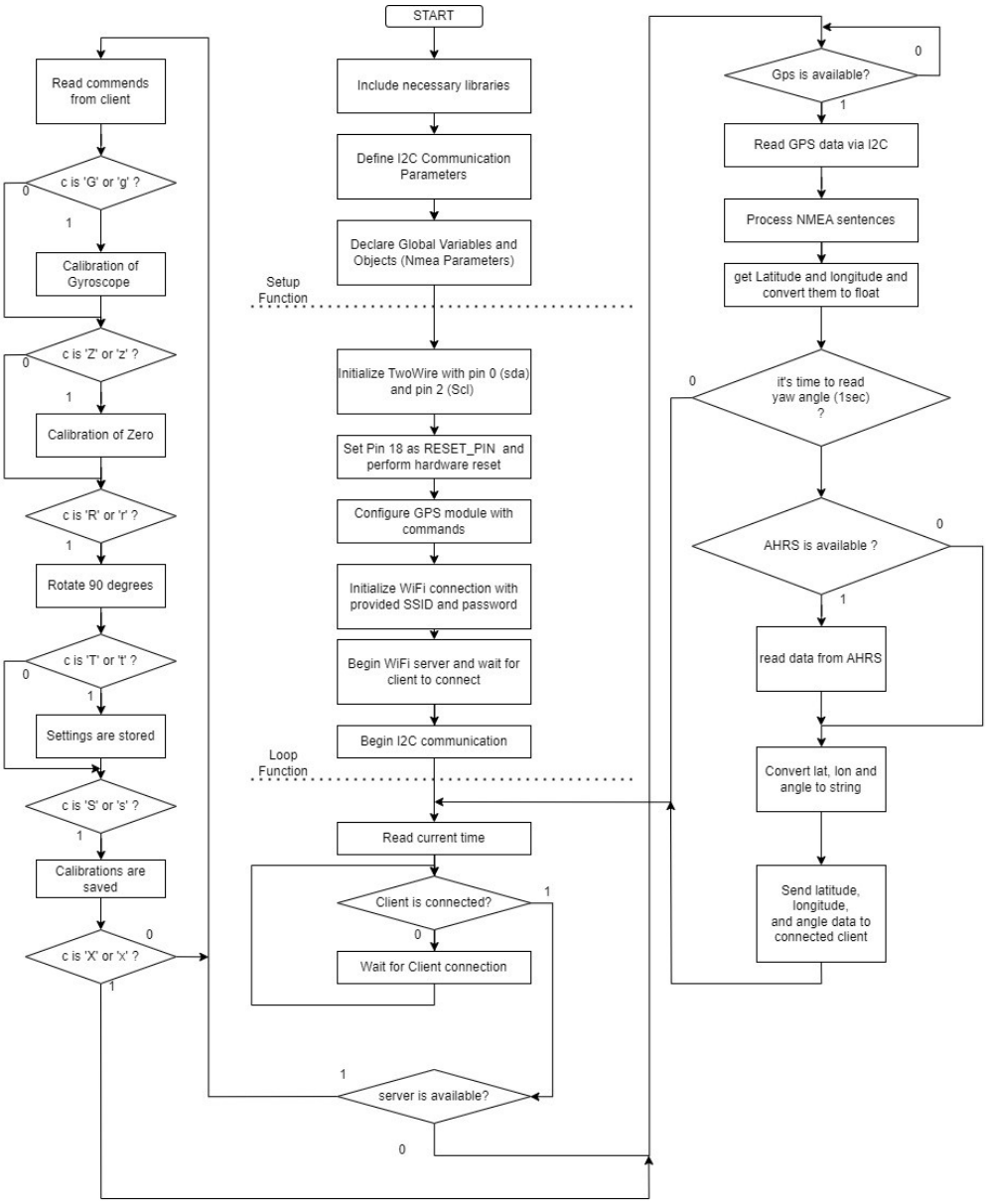


Figure 14. Flowchart of the ESP32 code

To streamline the coding process, a carefully designed flowchart, as illustrated in Figure 14, was formulated. Following this schematic, the code was meticulously crafted in the C language within the Arduino IDE environment. The detailed code is accessible in Appendix A for reference purposes. Following this, I uploaded the compiled code to a module already set up with the "ESP32", "OPUS-Inertial", and "Teseo-Liv3f" components, using the ESP-PROG tool.

Now, let us embark on a step-by-step examination of the developed code:

1. Include necessary libraries such as " WiFi.h", "Arduino.h", " Wire.h", "MicroNMEA.h". MicroNMEA is necessary to read data from TESEO-LIV3F.

2. Define I2C communication parameters: To define the I2C communication parameters for OPUS-Inertial-P20 series, the slave address is 7 bits and set to 0x51, while for Teseo-Liv3f it is 0x3A by default. The RESET PIN is located on pin-18. Figure 15 shows the register address for roll, yaw, and pitch for OPUS-Inertial-P20.

0x18	ROLL0	ROLL (in Float32)	R
0x19	ROLL1		
0x1A	ROLL2		
0x1B	ROLL3		
0x1C	PITCH0	PITCH (in Float32)	R
0x1D	PITCH1		
0x1E	PITCH2		
0x1F	PITCH3		
0x20	YAW0	YAW (in Float32)	R
0x21	YAW1		
0x22	YAW2		
0x23	YAW3		

Figure 15. Register Address of the roll, yaw and pitch

3. Create an instance of the Wire library for liv3f and OPUS-Inertial

4. Define some global variables: To define some global variables, we need to set the time interval to 1000 milliseconds, which means data will be sent every second. Next, we define the following parameters:

- SSID (Service Set Identifier): This is the name of the Wi-Fi network to which our ESP32 will connect. For this project, we have set it as "DESKTOP-1CTQC9F-4593".
- Password: This is a password required to access the Wi-Fi network defined by the SSID. It has been set as "3M45$64p".
- WiFiServer server(48569): This initializes a Wi-Fi server object on port 48569. The ESP32 will listen for incoming connections on this port.
- WiFiClient serverclient: This declares a client object named serverclient which will be used to communicate with the server.

5. Define functions:

- liv3fHardwareReset(): This function resets the hardware of the Teseo-Liv3f device. It does this by briefly setting the RESET_PIN to LOW, causing a reset, and then returning it to HIGH after a short delay. Note that pin-18 was previously defined as the reset pin.
- sendCommand(char *cmd): This function sends a command to the Teseo-Liv3f device via the I2C protocol. It takes a command string (cmd) as input and transmits it to the device. The command is sent using the Wire library, which handles communication over the I2C bus.
- Function readFloatFromWire(byte rawData[4]): This function reads a floating-point value from a 4-byte data stream represented by the rawData array. The function takes an array of bytes as input and returns a float value.

6. Setup function: The setup function initializes various components and establishes connections required for the operation of the system. Firstly, it initializes communication with the Teseo-Liv3f device using the I2C protocol. Then, it configures the RESET_PIN as an output and sets it to HIGH, ensuring proper hardware initialization. Other pin numbers are given as shown in Figure 16.

AHRS	SDA:0
	SCR:2
GPS	SDA:0
	SCL:2
	RST:18

Figure 16. pin numbers of GPS and AHRS for I2C communication

Next, the liv3fHardwareReset function is called to reset the Teseo-Liv3f hardware. Following this, two commands are sent to configure the Teseo-Liv3f device using the sendCommand function. To reset the I2C-MessageList, use "$PSTMCFGMSGL". If you want to disable echo-ing commands, use "$PSTMSETPAR" commands. After a brief delay, Wi-Fi connectivity is initiated, and the ESP32 attempts to connect to the specified network using the provided SSID and password. Finally, the Wi-Fi server is started, and the serverclient object is initialized to handle incoming client connections. Additionally, the Wire library is initialized for I2C communication with given pin numbers.

7. Loop function: The loop function contains the code that runs repeatedly for the duration of the program's execution. In this specific code, the loop function continually performs several tasks in a sequential manner. Firstly, it checks for any incoming data from the client connected to the Wi-Fi server. If there is incoming data, it processes the commands sent by the client, such as initiating gyroscope calibration or adjusting zero values for OPUS-Inertial. First, it starts communicating with the device on the I2C bus at address 0x51, which is the default address for AHRS as mentioned before. Then, by sending the 0x07 command code to the device, it is indicated that this address will be used. Then, the relevant bit is set to 1 for the mode to be set. Figure 17 shows the mode and bit number to be set. Additionally, a short delay is required for the gyroscope calibration process.

Register Address	Name	R/W	Description
0x07	CTL1	R/W	Bit[0]: 1 GyCal
			Bit[1]: 1 MgCalZ
			Bit[2]: 1 MgCalR
			Bit[3]: 1 MgCalS
			Bit[4]: 1 SaveCal
			Bits[5..8]: RSVD

Figure 17. register address of AHRS for calibration

Next, it reads data from the Teseo-Liv3f device using the I2C protocol, specifically requesting latitude and longitude information. It then parses this data and prepares it for transmission.

After that, it checks if it's time to send data to the server. If the predefined interval, which is 1 second there, has elapsed, it reads the angle data from the Teseo-Liv3f device, converts it to a suitable format, and constructs a string containing latitude, longitude, and angle information. This string is then sent to the server client for transmission over Wi-Fi.

Overall, the loop function continuously handles incoming commands from the client, retrieves data from the Teseo-Liv3f device, and transmits relevant information to the server over Wi-Fi, ensuring the smooth operation of the navigation system.

Testing and Validation

Extensive testing was performed to verify the accuracy and reliability of position and orientation calculations. The performance of the outdoor navigation system was simulated and evaluated in real-world scenarios. As a result, accurate results were not always obtained, as Teseo-Liv3f could provide an accuracy of 1-3 meters and factors such as satellite signal quality, environmental conditions and receiver configuration could affect this accuracy. Apart from this, it can be said that the project gave satisfactory results in this short period of my internship. An example test is visualized in Figure 18, Figure 19 and Figure 20 below.

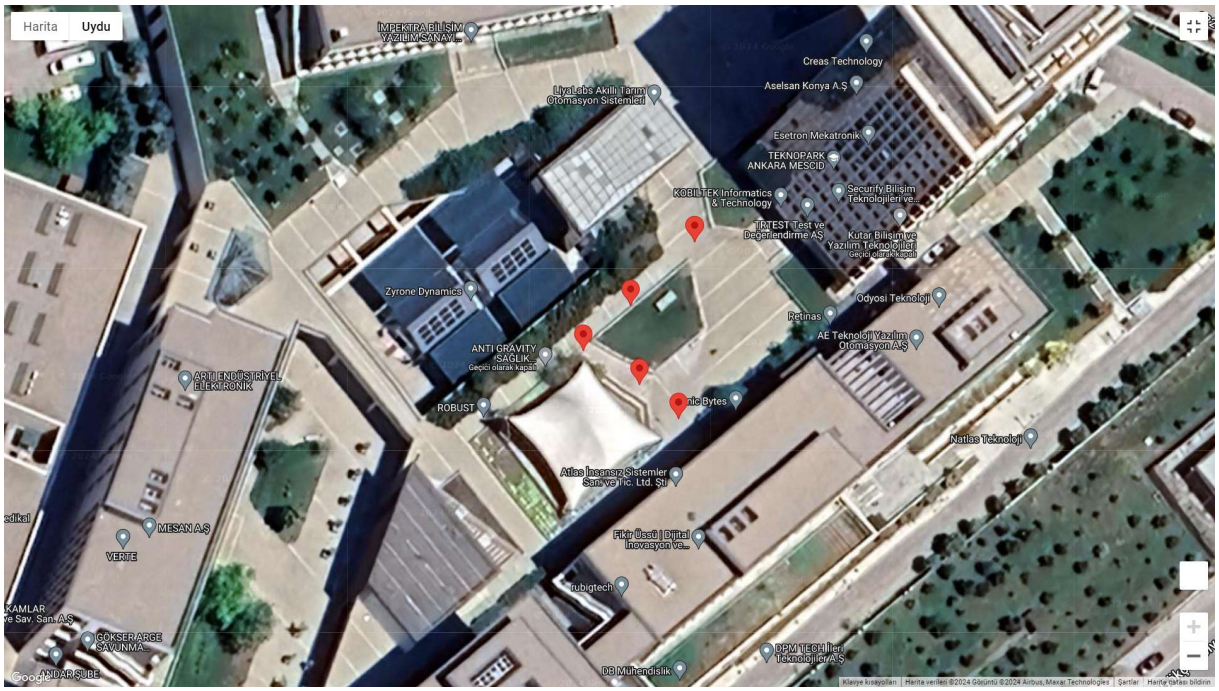


Figure 18. Locations which will be visited are marked.

```
Connected to the server.
##############################################
'Return RIGHT at angle: 88.31180045589306 '
##############################################
'Latitude: 39.99557909199'
'Longitude: 32.752241866126'
'Angle coming from AHRS: 39.4239201428'
##############################################
'Minimum distance to the final location: 22.880456142900094 m'
```

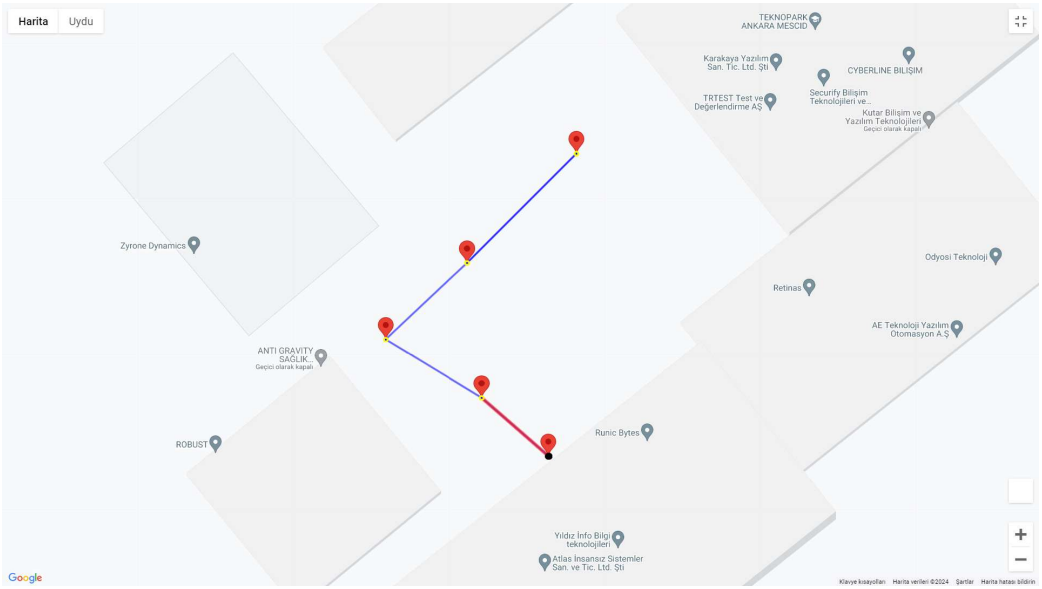


Figure 19. locations are marked and blue for Among the places to visit and red for between the closest point and the current location lines are drawn.

```
Connected to the server.
################################################

'Go straight!'

################################################

'Latitude: 39.99557909199'

'Longitude: 32.752241866126'

'Angle coming from AHRS: -50.4239201428'

################################################

'Minimum distance to the final location: 22.880456142900094 m'
```

Figure 20. marked locations on gmap and instructions.

```
Connected to the server.
###############################################
'Return LEFT at angle: 4.966995072632528 '
###############################################
'Latitude: 39.99575750776558'
'Longitude: 32.75217910537852'
'Angle coming from AHRS: 37.84453279'
###############################################
'Minimum distance to the final location: 26.27410868300521 m'
```

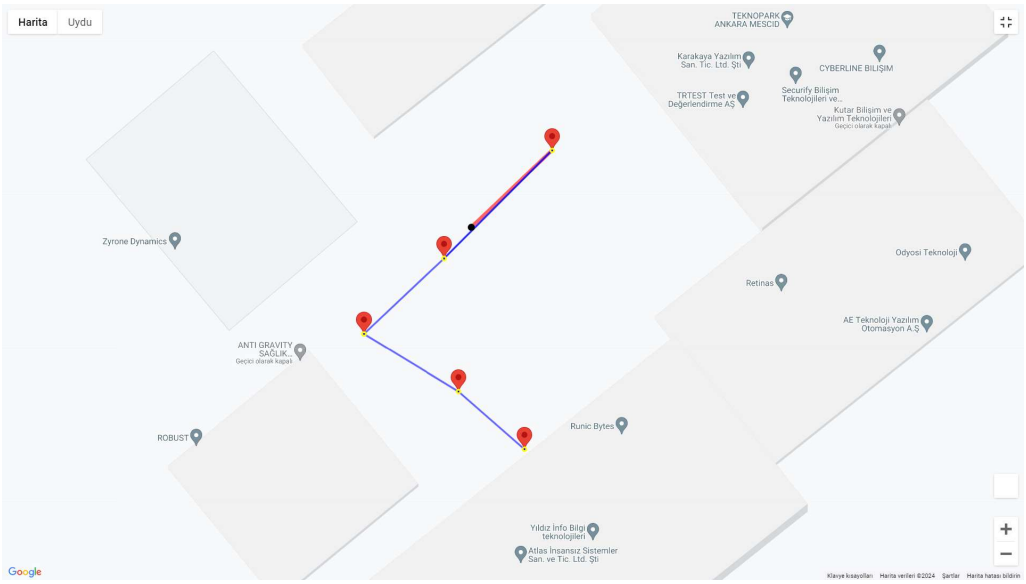


Figure 21. marked locations on gmap and instructions.

```
Connected to the server.
###############################################
'You reached the target!'
###############################################
'Latitude: 39.9958349171'
'Longitude: 32.7522718203'
'Angle coming from AHRS: 57.35455571'
###############################################
'Minimum distance to the final location: 14.616637833580647 m'
```

Figure 22. location information, directions, instructions and maps with relevant markings

Conclusion

In conclusion, the development of the ESP32-based navigation system with GPS and AHRS integration was a major milestone during my internship at OPUS Embedded. This project allowed me to apply theoretical knowledge to real-world applications, improving the skills I acquired in embedded systems, software development and school (in the computer field option).

Working together with my supervisor and using important technologies such as the ESP32 microcontroller, Teseo-Liv3f GPS module and OPUS-Inertial AHRS device, we successfully created a versatile and accurate navigation system. The software design, implemented in both Python and Arduino languages, showcased the integration of TCP client-server communication, real-time data processing, and interactive map visualization using the gmaps library. This user-friendly interface improves usability and provides users with intuitive navigation feedback.

During the testing and validation phase, extensive testing was carried out to ensure the accuracy and reliability of the navigation system. While some difficulties were encountered within the scope of the project, such as changing GPS accuracy due to environmental factors, the overall performance met expectations. In summary, my EE400 internship at OPUS Embedded was very productive, thanks to my supervisors and the opportunities they offered.

References

[1]. https://www.digikey.fi/htmldatasheets/production/3350764/0/0/1/teseo-liv3f.html#pf11

[2]. https://www.st.com/resource/en/datasheet/teseo-liv3f.pdf

[3]. https://www.st.com/resource/en/user_manual/um2229-teseoliv3-gnss-module--software-manual-stmicroelectronics.pdf

[4]. https://www.electronicwings.com/esp32/gpio-of-esp32

[5]. https://micronmea.readthedocs.io/en/latest/

[6]. OPUS-Inertial-P20 (2024). Firmware Interface For I2C (FI_I2C), Revision 1.0. OP020-007-P20

[7]. https://jupyter-gmaps.readthedocs.io/en/latest/index.html

[8]. https://www.arduino.cc/reference/en/libraries/wifi/

[9]. Gölbol, F., Ölmez, H., Hacınecipoğlu, A., & Ankaralı, M. M. (2020, October). Developing a motion controller for autonomous agricultural robot. In *2020 28th Signal Processing and Communications Applications Conference (SIU)* (pp. 1-4). IEEE.

[10]. Chopde, N. R., & Nichat, M. (2013). Landmark based shortest path detection by using A* and Haversine formula. *International Journal of Innovative Research in Computer and Communication Engineering*, *1*(2), 298-302.

[11]. Gebre-Egziabher, D., Hayward, R. C., & Powell, J. D. (1996, April). A low-cost GPS/inertial attitude heading reference system (AHRS) for general aviation applications. In IEEE 1998 Position Location and Navigation Symposium (Cat. No. 98CH36153) (pp. 518-525). IEEE.

[12]. English, A., Ross, P., Ball, D., & Corke, P. (2014, May). Vision based guidance for robot navigation in agriculture. In *2014 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 1693-1698). IEEE.

[13]. Hofmann-Wellenhof, B., Legat, K., & Wieser, M. (2003). Navigation. Springer Science & Business Media.

APPENDIX-A

```
#include <WiFi.h>
#include <Arduino.h>
#include <Wire.h>
#include <MicroNMEA.h>

//I2C communication parameters
#define DEFAULT_DEVICE_ADDRESS 0x3A
#define DEFAULT_DEVICE_PORT 0xFF

#define RESET_PIN 18

TwoWire& liv3f = Wire;

char c;
int idx = 0;

char nmeaBuffer[100];
MicroNMEA nmea(nmeaBuffer, sizeof(nmeaBuffer));

unsigned long previousMillis = 0;
const long interval = 1000;

void liv3fHardwareReset();

void sendCommand(char *cmd);

char *ssid = "DESKTOP-1CTQC9F-4593";
char *password = "3M45$64p";

WiFiServer server(48569);


WiFiClient serverclient;
float readFloatFromWire(byte rawData[4]);

////////////////////////////////////
char latBuffer[20];
char lonBuffer[20];
char angBuffer[20];

float readFloatFromWire(byte rawData[4]);

void liv3fHardwareReset()
{

   digitalWrite(RESET_PIN, LOW);
   delay(50);
   digitalWrite(RESET_PIN, HIGH);
   delay(2000);

}

float readFloatFromWire(byte rawData[4]) {
  float floatValue;
  byte* floatBytes = reinterpret_cast<byte*>(&floatValue);


  for (int i = 0; i < 4; i++) {
    floatBytes[i] = rawData[i];
  }
  return floatValue*57.2958;
}

void sendCommand(char *cmd)
{
   liv3f.beginTransmission(DEFAULT_DEVICE_ADDRESS);
   liv3f.write((uint8_t) DEFAULT_DEVICE_PORT);
   MicroNMEA::sendSentence(liv3f, cmd);
   liv3f.endTransmission(true);
}
void setup() {
  liv3f.begin(0,2);
```

```
  pinMode(RESET_PIN, OUTPUT);
  digitalWrite(RESET_PIN, HIGH);

  liv3fHardwareReset();

  sendCommand("$PSTMCFGMSGL,3,1,0,0");
  sendCommand("$PSTMSETPAR,1227,1,2");

  delay(4000);

  liv3f.begin(0,2);

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }

   server.begin();
  while(!((serverclient)&&(serverclient.connected()))){
    serverclient = server.available();
  }
  Wire.begin(0,2);

}

void loop() {
  unsigned long currentMillis = millis();

  float lat, lon;
  byte rawData[4];
  int x=0;
  if(serverclient && serverclient.connected()){
    if(serverclient.available()){
    char d =serverclient.read();
    if(d=='G'||d=='g'||d=='Z'||d=='z'||d=='R'||d=='r'||d=='T'||d=='t'||d=='S'||d=='s'){

      do{

        switch(d) {
          case 'G':
          case 'g':
              {
                Wire.beginTransmission(0x51);
                Wire.write(0x07);
                Wire.write(0x01);
                Wire.endTransmission();
                serverclient.print("gyroscope calibration is started...");
                delay(5000);
                serverclient.print("Done!!!\n");
                serverclient.print("You can leave by pressing X");
              }
              break;

          case 'Z':
          case 'z':
              {
                Wire.beginTransmission(0x51);
                Wire.write(0x07);
                Wire.write(0x02);
                serverclient.print("Zero is adjusted!\n");
                Wire.endTransmission();
                serverclient.print("You can leave by pressing X");
              }
              break;
          case 'R':
          case 'r':
              {
                Wire.beginTransmission(0x51);
                Wire.write(0x07);
                Wire.write(0x04);
                serverclient.print("Rotate right 90 degrees\n");
                Wire.endTransmission();
                serverclient.print("You can leave by pressing X");
              }
              break;
          case 'T':
          case 't':
```

```
            {
              Wire.beginTransmission(0x51);
              Wire.write(0x07);
              Wire.write(0x08);
              serverclient.print("Results are stored !!!\n");
              Wire.endTransmission();
              serverclient.print("You can leave by pressing X");
            }
            break;
        case 'S':
        case 's':
            {
              Wire.beginTransmission(0x51);
              Wire.write(0x07);
              Wire.write(0x10);
              serverclient.print("Calibration is saved!\n");
              Wire.endTransmission();
              serverclient.print("You can leave by pressing X");
            }
            break;
        case 'X':
        case 'x':
            x=1;
            break;
        default:
            break;
    }
    d='a';
      delay(10);
      if(serverclient.available()){
        d =serverclient.read();
      }

    }while(x==0);
    x=0;

  }


}
  liv3f.beginTransmission(DEFAULT_DEVICE_ADDRESS);
  liv3f.write((uint8_t) DEFAULT_DEVICE_PORT);
  liv3f.endTransmission(false);
  liv3f.requestFrom((uint8_t)DEFAULT_DEVICE_ADDRESS, (uint8_t) 32);

  while(liv3f.available()){
    c=liv3f.read();
    nmea.process(c);
     if (nmea.isValid()){
      long latitude_mdeg, longitude_mdeg;
      latitude_mdeg = nmea.getLatitude();
      longitude_mdeg = nmea.getLongitude();
      // Convert to degrees
      lat = static_cast<float>(latitude_mdeg) / 1000000.0f;
      lon = static_cast<float>(longitude_mdeg) / 1000000.0f;
      nmea.clear();
      break;
   }
  }


  if (currentMillis - previousMillis >= interval){

    previousMillis = currentMillis;
    Wire.beginTransmission(0x51);
    Wire.write(0x20);
    Wire.endTransmission();
    Wire.requestFrom(0x51, 4);
    delay(100);

    if (Wire.available() >= 4) {

      for (int i = 0; i < 4; i++) {
        rawData[i] = Wire.read();
        }
      }
```

```
    float ang = readFloatFromWire(rawData);

    dtostrf(lat, 17, 10, latBuffer);
    dtostrf(lon, 17, 10, lonBuffer);
    dtostrf(ang, 17, 10, angBuffer);

    String data = String(latBuffer) + ", " + String(lonBuffer) + ", " + String(angBuffer);

    serverclient.println(data);

    }

  }
  else{
    while(!((serverclient)&&(serverclient.connected()))){
      serverclient = server.available();
    }
  }

}
```

APPENDIX-B

```
# Cell-1

*********************************************************************************************

import ipywidgets as widgets
import geopy
import gmaps
import time
import socket
import math
import threading
from IPython.display import display, clear_output, update_display

################################  Some Variables  ###################################

gmaps.configure(api_key='Api-Key')  #         Api-Key

server_address = ('192.168.1.25', 48569)  #         Server-Address and Port-Number

initial_location = (39.99515813942635, 32.750430237878376)  #        Initial-Location

length_tolerance = 3

angle_tolerance = 3

r = 6371                        #                          Radius of earth in kilometers

#############################################################################################

fig = gmaps.figure(center=initial_location, zoom_level=20)

marker_locations = []  #                 Initialize the list to store marker locations

def handle_map_click(feature):
    global target_location
    lat, lng = feature.location
    target_location = (lat, lng)
    global marker_locations
    marker_locations.append(target_location)

drawing_layer = gmaps.drawing_layer()

fig.add_layer(drawing_layer)

drawing_layer.on_new_feature(handle_map_click)

display(fig)

*********************************************************************************************

# Cell-2

*********************************************************************************************

fig = gmaps.figure(center=marker_locations[0], zoom_level=20)

line_layer = gmaps.drawing_layer()

lines = []
dots = []
hover_text = []

# Adding dots for each marker location
for i, location in enumerate(marker_locations):
    dot = gmaps.symbol_layer([location], fill_color='green', stroke_color='black', scale=[5])
    dots.append(dot)

    # Adding location info to hover_text
    hover_text.append(f"Location {i+1}: {location}")

# Adding lines between marker locations
for i in range(len(marker_locations)-1):
```

```
    line = gmaps.Line(
        start=marker_locations[i],
        end=marker_locations[i+1],
        stroke_weight=3.0,
        stroke_color="blue"
    )
    lines.append(line)

# Adding the drawing layer, dots, and lines to the figure
fig.add_layer(drawing_layer)
#fig.add_layer(gmaps.drawing_layer(features=lines))  # Note: This line replaces the previous
line_layer
fig.add_layer(gmaps.symbol_layer(marker_locations, fill_color='blue', stroke_color='yellow',
scale=3, hover_text=hover_text))

*******************************************************************************

# Cell-3

*******************************************************************************

# Creating a TCP client socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connecting to the server

try:
    client_socket.connect(server_address)
    print("Connected to the server.")
except Exception as e:
    print(f"Unable to connect to the server: {e}")
    exit()

from math import radians, sin, cos, sqrt, atan2, asin, degrees

# the distance between two locations with following formula:
acos(sin(lat1)*sin(lat2)+cos(lat1)*cos(lat2)*cos(lon2-lon1))*6371

def dist(loc1, loc2):

    # Converting latitude and longitude from degrees to radians

    lat1, lon1 = radians(loc1[0]), radians(loc1[1])
    lat2, lon2 = radians(loc2[0]), radians(loc2[1])

    # Haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2

    c = 2 * asin(sqrt(a))

    return(c * r*1000)

marker_init = gmaps.symbol_layer([(39,32)])

def calculate_angle(point1, point2):
    # Calculation of the bearing angle between the line and the north
    lat1, lon1 = radians(point1[0]), radians(point1[1])
    lat2, lon2 = radians(point2[0]), radians(point2[1])

    delta_longitude = lon2 - lon1

    x = sin(delta_longitude) * cos(lat2)
    y = cos(lat1) * sin(lat2) - (sin(lat1) * cos(lat2) * cos(delta_longitude))

    bearing_angle_rad = atan2(x, y)
    bearing_angle_deg = (degrees(bearing_angle_rad) )

    return bearing_angle_deg

# Create display objects for each piece of text
print("#############################################")
output_display = display("", display_id=True)
print("#############################################")
latitude_display = display("", display_id=True)
longitude_display = display("", display_id=True)
```

```
angle_display = display("", display_id=True)
print("##############################################")
distance_display = display("", display_id=True)
def receive_data():
    while True:
        try:
            # Received data from the server
            data = client_socket.recv(1024).decode('utf-8')

            if not data:
                break  # Break the loop if no data is received

            # Splitting the received data into latitude and longitude
            try:
                latitude, longitude, angle = map(float, data.split(','))
            except ValueError:
                #update_display(f"Invalid data format received from the server.",
display_id=output_display.display_id)

                continue

            # Update latitude, longitude, and angle displays
            update_display(f"Latitude: {latitude}", display_id=latitude_display.display_id)
            update_display(f"Longitude: {longitude}", display_id=longitude_display.display_id)


            # Process of the received data
            # Update the location for the initial_location which comes from the server
            global initial_location
            global marker_locations
            initial_location = (latitude, longitude)
            global distance_first
            distance_first = float('inf')
            global distance_final
            distance_final = float('inf')

            #initial_location = (latitude, longitude)

            # Calculate distances and print the minimum distance to the target and first
location

            if len(marker_locations) > 0:
                distance_first = dist(initial_location, marker_locations[0])
                for l in range(len(marker_locations)-1):
                    distances=[dist(marker_locations[l],marker_locations[l+1])]
                distance_final_sum=sum(distances)+distance_first
                update_display(f"Distance to the first location: {distance_first} m",
display_id=distance_display.display_id)
                update_display(f"Minimum distance to the final location: {distance_final_sum}
m", display_id=distance_display.display_id)

            marker_init.markers = [gmaps.Symbol(location=initial_location)]

            if(distance_final<length_tolerance) or len(marker_locations)==0:
                    update_display(f"You reached the target!",
display_id=output_display.display_id)
                    break

            if distance_first<3 and len(marker_locations)!=0:
                marker_locations.pop(0)

            if len(marker_locations)>=1:
                line = gmaps.Line(
                    start=initial_location,
                    end=marker_locations[0],
                    stroke_weight=6.0,
                    stroke_color="red"
                )
                line_layer.features = [line]

                #angle to the north which comes from map - line from our loc to the first loc

                turning_angle=calculate_angle(initial_location,marker_locations[0])
                update_display(f"Angle coming from AHRS: {angle}",
display_id=angle_display.display_id)
```

```
                #angle we need to turn to reach the target

                turning_angle=turning_angle-angle

                if(angle_tolerance>abs(turning_angle)):
                    update_display(f"Go straight!", display_id=output_display.display_id)
                elif (turning_angle >= angle_tolerance) and (turning_angle <= (180-
angle_tolerance)):
                    update_display(f"Return LEFT at angle: {turning_angle} ",
display_id=output_display.display_id)
                elif (turning_angle > 180):
                    update_display(f"Return RIGHT at angle: {360-turning_angle} ",
display_id=output_display.display_id)
                elif (turning_angle >=(-180)) and (turning_angle < -abs(angle_tolerance)):
                    update_display(f"Return RIGHT at angle: {abs(turning_angle)} ",
display_id=output_display.display_id)
                elif turning_angle < -180:
                    update_display(f"Return LEFT at angle: {360 + turning_angle} ",
display_id=output_display.display_id)
                else:
                    continue


        except Exception as e:
            print(f"Error receiving data from server: {e}")
            break

        time.sleep(1)


# Start the receive_data function in a separate thread

receive_thread = threading.Thread(target=receive_data)
receive_thread.start()
# Add the drawing layer

lines.append(line)

fig.add_layer(line_layer)
fig.add_layer(marker_init)
fig.add_layer(gmaps.drawing_layer(features=lines))
fig
```