UPPSALA
UNIVERSITET

UPTEC F 22029
Examensarbete 30 hp
Juni 2022

GPS tracking device

Ivet Shami Jamil

Civilingenjörsprogrammet i teknisk fysik

UPPSALA
UNIVERSITET

GPS tracking device

Ivet Shami Jamil

Abstract

The project was performed in collaboration with the company Prevas AB. The aim of the project was to implement a GPS tracking device, or a GPS tracker, that has a hardware setup retrieving the GPS position of an object and then sends the position through a mobile communication module to a smartphone. A software application running on a smartphone simulator was developed to visualize the position with a Google map widget. The hardware unit consists of a GPS/GSM/GPRS module, a microcontroller, and a smartphone simulator that runs the software application made for iOS, Android and Linux operating systems using Flutter - an UI multi-platform software development kit.

Implementation of the setup started with a Telit GM862 GSM/GPRS/3G-GPS module (together with a SmartGM862 development board), an Arduino UNO microcontroller board and a logical level converter. This first setup was not successful because a problem with the serial communication between the Telit GSM-GPS module and the Arduino could not be solved.

The second setup was implemented in which an Arduino GSM/GPRS/GPS shield was used with the Arduino board. The serial communication between the shield and the Arduino was functioning properly and a GPS position could be retrieved. The GPS position is published (sent) using a lightweight messaging protocol - MQTT (messaging queuing telemetry transport) through GPRS functions to the smartphone of the destination user.

To visualize the GPS position in a Google map, an Object Tracker app was implemented on the smartphone simulator app (an iPhone in this project) that subscribes and retrieves the GPS position, and then displays it on the map with a red marker indicating the position.

The GPS tracker with the second setup and the app has been tested and proved to work properly. It has, however, much room for improvement and further development, e.g., making the app more user-friendly, and designing and making a PCB so all the components are mounted on one board. It has been noticed in an initiated test that the GPS antenna placement could affect positioning accuracy. This, thus, needs to be considered for high accuracy.

Teknisk-naturvetenskapliga fakulteten

Uppsala universitet, Uppsala

Handledare: Anders Jansson

Ämnesgranskare: Ping Wu

Examinator: Tomas Nyberg

ISSN: 1401-5757, UPTEC F 22029

Populärvetenskaplig Sammanfattning

Mycket teknik är idag implementerad i vardagliga produkter och telefoner. IoT-teknik har fått ett stort genomslag och finns i många hem och ger stöd i människors vardag. En sådan teknik är GPS-spårning och återfinns i mobiltelefoner, appar som Uber samt så kallade "airtags" som kan placeras på valfria objekt. Inspirationen till detta projekt har kommit till från de växande cykelstölderna i Uppsala stad och där polisens resurser i dagsläget inte räcker för att lösa och motverka dessa stölder. För att motverka dessa stölder och ge stöd i Polisens arbete kan IoT-tekniken komma väl till hands genom att installera en GPS-spårare på en cykel men även på andra fordon.

Projektet som har framtagits hos Prevas AB är utfört som ett "proof of concept" projekt och är därför inte fokuserat på en optimal GPS-spårare utseendemässigt. Funktionalitet för detta projekt ligger i fokus. Målet för projektet är att tillverka en GPS-spårare som erhåller en position i latitude och longitude. Denna position presenteras sedan på en app med en implementerad "Google map" widget. Positionen skall visas tydligt med en röd markering. Vid en förändrad position skall kartan förflyttas och den nya positionen presenteras.

Systemet för en GPS-spårare kan beskrivas med tre byggnadsstenar. Den första byggstenen består av hårdvaran som är ansvarig för att erhålla en GPS position och formattera om positionen till lämplig form för överföring till nästa byggsten. Nästa byggsten är en server som agerar förmedlare mellan hårdvara och app. Meddelandet skickas från hårdvaran till servern med kommunikationsprotokollet MQTT. Därmed kallas denna server för MQTT broker. Till en broker kan ett flertal utgivare och abonnenter vara anslutna till ett ämne som är skapat i brokern. För detta projekt agerar hårdvaran som utgivare, därifrån positionen publiceras och appen agerar som en abonnent som tar emot dessa meddelanden innehållande positionen. Den tredje byggstenen för systemet är appen. Appen abonnerar på ämnet och positionen inhämtas. Positionen omformateras till en datatyp som en Google karta tillåter och därmed kan en position presenteras med en röd markering.

Initialt konstruerades en GPS-spårare med en konfiguration som innehöll en Arduino Uno, modulen Telit GM862-GPS som är en GSM/GPRS/GPS modul och en nivåomvandlare som omvandlar de höga spänningsnivåerna från Arduinon till de låga nivåerna som GPS-modulen tolererar. GPS-modulens funktioner är tillgängliga genom AT-kommandon som skickas från Arduinon genom seriell kommunikation. Ingen respons från GPS-modulen erhölls när AT-kommandon skickades. Resultat av felsökning gav att den nämnda GPS-modulen var defekt och därmed kunde inte en slutgiltig prototyp erhållas. Tillverkaren av modulen hävdade att modulen är uråldrig och har inte varit till försäljning under 10 år och därmed kunde inte support erbjudas. Av den anledningen konstruerades en sekundär konfiguration innehållande en Arduino Uno samt en Arduino-skärm innehållande GSM/GPRS/GPS-modulen SIM808. Med den sekundära konfigurationen uppnås en slutgiltig hårdvaruprototyp som sedan placerades i en 3D-utskriven låda anpassad för hårdvaran. Tester av GPS-modulens noggrannhet samt känslighet utfördes vilket påvisade att placeringen av GPS-antennen påverkar nog-

grannheten till stor del. Bäst noggrannhet erhölls när GPS-antennen var placerad utomhus, det andra när GPS-antennen är placerad inomhus i närheten av ett fönster. Det sämsta resultatet erhölls när GPS-antennen var placerad utomhus i närheten av metall vilket skapar interferens och därmed undermålig noggrannhet av positionsbestämningen.

Appen utvecklades med plattformen Flutter. Flutter som är tillverkat av Google är plattformsoberoende och därmed kan appar med samma kodbas brukas av operativsystemen Android och iOS. Den fullständiga appen har möjlighet att inhämta en position, formatera den till lämplig datatyp och presentera positionen på en karta. I appen har en meny implementerats med alternativen "Hem", "Inställningar" samt "Parkering". I appen är även en uppdateringsikon implementerad för att uppdatera position av det spårade objektet.

Eftersom projektet är ett "proof of concept" förekommer det många utvecklingsmöjligheter. Förslagsvis implementering av en funktion där användaren notifieras om en förändrad position av det spårade objektet upptäcks, implementera funktionerna "Inställningar" och "Parkering" fullständigt och optimera appens användarvänlighet. Förbättringar inom hårdvaruaspekten måste utföras i form av reducering av spårarens storlek samt optimering av spårarens nuvarande energikonsumtion som är hög.

Acknowledgements

I am extremely grateful for the support, guidance and valuable knowledge that Mikael Hamberg and Anders Fagerström have provided throughout this thesis. I would also like to express my gratitude for my supervisor Anders Jansson for his support and my subject reader Ping Wu for his advice and suggestions. To my family Rifat, Susan and Mari, you always inspire me to do better and this would not have been possible without your love and support throughout my life and education. Finally, my husband Ramis. Your encouragements and your love means the world to me and I will be forever grateful for you and what you make me achieve for each day that passes by.

Acronyms

API - Application Programming Interface
BSC - Base Station Controller
BTS - Base Transceiver Station
GDOP - Geometric Dilution Of Precision
GGSN - Gateway GPRS support node
GPRS - General Packet Radio Service
GSM - Global System for Mobile Communications
HDOP - Horizontal Dilution Of Precision
IMSI - International Mobile Subscriber Identity
IoT - Internet of Things
JSON - JavaScript Object Notation
MQTT - Messaging Queuing Telemetry Transport
MS - Mobile Station
NMEA - National Marine Electronics Association
OS - Operating System
QoS - Quality of Service
RMS - Root Mean Square
SDK - Software Development Kit
SGSN - Serving GPRS support node
SIM - Subscriber Identity Module
UI - User Interface
VDOP - Vertical Dilution Of Precision
VSWR - Voltage Standing Wave Ratio

Contents

1

Introduction

1.1 Background

On the 3rd of January 2022, an article was published in www.svt.se [1]. The police department has reported new numbers for bike theft in Uppsala, where an increase of 63% between 2016 and 2020 was shown . For each year, the police department receives an approximate 4000 reports and equally many thefts are left unreported. To counteract and discourage to eventual theft, a GPS tracking device can be installed on a bike. Unfortunately, other vehicles such as electric scooters and motorbikes are equally exposed to theft and therefore it is also suggested to implement such a device on other vehicles.

The project is preformed in collaboration with Prevas AB which is a consulting company that provides technical solutions and technical expertise. Similar projects as this master thesis project but for different applications have been performed at Prevas. One example is the alcohol meter tripleA and its corresponding app from Kontigo Care that is developed together with Prevas.

1.2 Purpose and Goals

This master thesis project is mainly a "proof of concept" project with numerous possible future developments. The aim is to implement a device to track an optional vehicle. If the vehicle is moved from its original position, the new position is provided to the user via the app on a smartphone. The new location is shown using Google maps services to achieve a user friendly app. The device should preferably be placed in a case to protect the device from different weather conditions and to protect the electronics from external damages. The case will also provide the user to attach the device easily on an optional vehicle.

1.3 Tasks and scope

The main project goal is to achieve a GPS tracker that retrieves a position and visualize the position on a Google map widget in an app. This is may be achieved by

- Correctly connect an Arduino Uno to a GPS/GPRS/GSM-module.
- Develop an Arduino code that retrieves a GPS position and publishes it to a server in a correct format.
- Retrieve the position from the server and convert the position to a desired format.
- Present the position on a Google map widget on a developed app.
- Investigate the performance of the tracking device.

1.4 Outline

The report is arranged as follows. Chapter 2 includes theoretical and technical background about GPS and trilateration, the geographical coordinate system, wireless mobile communications and GPRS, and information about the MQTT protocol. In chapter 3, the methods and implementation of the project is presented, covering one unsuccessful configuration and one successful configuration of the GPS tracker. Information about the hardware, MQTT broker and software is also included. At the end of the chapter the app development, its methods and background are presented. In chapter 4 the final prototype and its application and functions are presented together with a discussion. In chapter 5 the conclusion is drawn and improvements are suggested.

2

Theoretical Background

2.1 Global Positioning System and Trilateration

GPS, or NAVSTAR GPS, is a navigation system that was developed by the United States Department of Defense in 1973 [9]. Via preposition of satellites, a user with a GPS module with a receiver is able to obtain the current positioning in terms of longitude, latitude and altitude. The positioning is determined as the distance of the GPS receiver is calculated relatively to other satellites. In satellites, atomic clocks are implemented, which grants very precise and accurate time and synchronization with other satellites. Data of the satellite's position is also recorded. Signals are transmitted from the satellites and propagated with a velocity of light and received at the ground level. When signals are received, two fundamental units are obtained which is an accurate time and velocity. Here, two things are stated. The velocity of the transmitted signal is constant and the time obtained is the time delay from signal transmission to receiving at the earth's surface. Further,the distance to each satellite is calculated, and thereby a position to the GPS receiver is determined via trilateration. For GPS position determination, an amount of at least 4 satellites are required [10]. With trilateration the relative distances are measured. Assume figure 2.1. The GPS receiver receives broadcasting signals from different satellites. Eventually the broadcasted satellite signal L_1 will be received by the GPS receiver. As the signal is received, the distance to the satellite will be known, implying that a GPS receiver must be positioned on the circumference of a circle with radius d_1. Receiving an additional broadcasted signal from satellite L_2, a new distance to the GPS receiver will be determined and a new circle with the radius d_2 will be obtained. It implies two circles that intersect each other in two points. For each point, there is a possibility that is the position of the GPS receiver. Therefore, an additional satellite is sufficient to determine the final position of the GPS receiver. Again, a circle with radius d_3 will be obtained and the circle will intersect the other circles in a certain point. As seen in Figure 2.1, the three circles intersect in one common point. In the intersecting point the GPS receiver exists, hence a position of the antenna is determined. This type of trilateration would have been perfect in a 2D world. For a 3D world, the mentioned circles are spheres and for that reason a minimum of 4 satellites are required to determine a position of an object in a 3D

world.

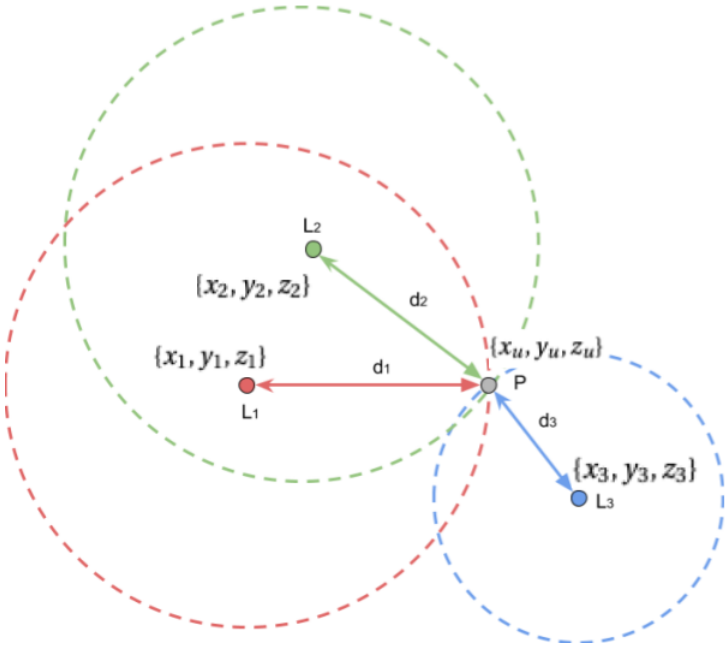


Figure 2.1: The trilateration method to determine a position of an object in a 2D world.

The unknown position of a GPS receiver can be calculated mathematically as

$$\begin{cases} d_1 = \sqrt{(x_1 - x_u)^2 + (y_1 - y_u)^2 + (z_1 - z_u)^2} \\ d_2 = \sqrt{(x_2 - x_u)^2 + (y_2 - y_u)^2 + (z_2 - z_u)^2} \\ d_3 = \sqrt{(x_3 - x_u)^2 + (y_3 - y_u)^2 + (z_3 - z_u)^2} \end{cases} \tag{2.1}$$

,where $\{d_1, d_2, d_3\}$ is the measured distance to each satellite, $\{x_1, x_2, x_3\}$, $\{y_1, y_2, y_3\}$ and $\{z_1, z_2, z_3\}$ is the known position of each satellite in the xyz direction, and $\{x_u, y_u, z_u\}$ is the unknown GPS receiver position. As three equations are obtained and three unknown exists, the equation is solvable.

Many elements contribute to the accuracy of a GPS position such as the amount of available satellites, refractions and multipath effects. Three important terms within GPS accuracy are geometric dilution of precision (GDOP), vertical dilution of precision (VDOP) and horizontal dilution of precision (HDOP) [9]. A good GDOP value is obtained if the visible satellites in the sky are spread apart. If the satellites are not spread apart, the circles mentioned previously will align close to each other and a poor position estimation will be obtained. The GDOP value is defined as:

$$GDOP = \frac{1}{\sigma}\sqrt{\sigma_x^2 + \sigma_y^2 + \sigma_z^2 + \sigma_b^2} \tag{2.2}$$

, where σ is the root means square (RMS) error of the psuedorange, which is the distance

between the user position and satellite position, $\{\sigma_x, \sigma_y, \sigma_z\}$ is the RMS error in the $\{x, y, z\}$ direction of the position and σ_b is the RMS clock error between the satellite clock and GPS user clock. A bad HDOP value is obtained when the position on the horizontal axis is poorly estimated and defined as

$$HDOP = \frac{1}{\sigma}\sqrt{\sigma_x^2 + \sigma_y^2} \tag{2.3}$$

A bad VDOP value is obtained as the position on the vertical axis is poorly estimated and defined as

$$VDOP = \frac{\sigma_z}{\sigma} \tag{2.4}$$

All DOP values are kept small if a good accuracy of the position is obtained.

The mentioned factors all together do affect the accuracy of the GPS position which is highly affected by the satellites position in the atmosphere. A GPS user is able to prevent the GPS receiver from exposure to multipath effects and nearby objects. Multipath effects are avoided as a GPS receiver is stationed far from buildings and structures. A GPS receiver should preferably not be placed inside a building due to signals that are impacted by the structure material. It is highly advised to not station a GPS receiver on or near metal to avoid signal interference which entails poor position estimation.

2.2 Geographical coordinate system

To obtain a GPS position, a coordinate system for the world has been interpreted. Any position in the world can be presented by a point given in latitude and longitude coordinates. The earth is approximated to a sphere with parallel and meridian lines. The parallel lines, the lines of latitude, present the lines drawn across the axis that goes through the north and south pole. The latitude value is in the range [-90°,90°] and presents every position in the north/south direction. The meridian lines that present the longitude values are instead of the range [-180°,180°] and present any position in the east/west direction, as shown in Figure 2.2.

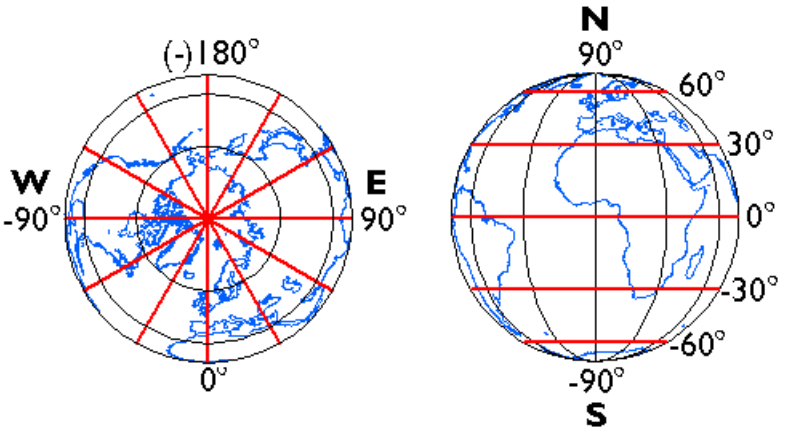


Figure 2.2: The geographic coordinate system where to the left the meridian lines are presented and to the right the parallel lines are presented.

Mathematically a position in latitude and longitude is expressed as follows

$$latitude = tan^{-1}\left(\frac{z_u}{\sqrt{x_u^2 + y_u^2}}\right) \tag{2.5}$$

$$longitude = tan^{-1}\left(\frac{y_u}{x_u}\right) \tag{2.6}$$

2.3 Wireless Mobile Communications and General Packet Radio Service

Wireless mobile communication was introduced year 1979 with its first generation 1G and has been developed to its fifth generation 5G year 2019. 1G is a fully analog telecommunication system. The communication was often terminated, the voice quality was bad, the maintained communication was not secured and the data rate was up to 2 kbps. The second generation 2G, also referred as GSM, was introduced 1991 and provided users with more secure and reliable communication. GSM is a digital network and new data services as SMS and MMS were introduced. The introduction of GSM pioneered for the line up of IoT and machine to machine communication. GSM provides with a data throughput up to 64 kbps. A further development of 2G is 2.5G, also referred to as GPRS and is a main block for the communication of this project. As GPRS was introduced year 2000, users were able to e-mail and web browse. The third generation 3G was introduced year 2001 and had a major success during the IPhone release, year 2008. The main development of 3G is increased bandwidth and higher data rates which implies to the ability to video stream and video call with fast and high quality communication. Data rates up to 2 Mbps is achieved with 3G data services. 4G, which was introduced year 2000 and developed until 2010, has data throughput which is up to 500 times higher than 3G data services. Higher resolution videos and

video calls are achieved with a data rate up to 1 Gbps. As 4G was developed, newer and higher technology possibilities could be achieved which gave the IoT market a huge breakthrough. Considering 4G provided with higher bandwidth and data rates, IoT devices were able to communicate with good QoS. The fifth generation 5G was introduced recently (2020) and is still under development. It can achieve data rates up to 35 Gbps. The goal of the 5G development is to reduce communication delay, more devices connected concurrently and higher connectivity so that 5G is suited for IoT.

As mentioned, the GPRS architecture is a further development of the GSM architecture at which the packet switched network is utilised rather than the circuit switched network that GSM utilises. Rather than occupying the lines fully as done for circuit switched networks, data is sent in packets within packet switched networks. What is achieved is higher throughput data rates and a constant internet connection with payments for only the amount of data that is utilised.

From the GSM architecture the block mobile station (MS) , base transceiver station (BTS) and base station controller (BSC) are implemented [2]. Different equipment can be used as a MS. A device alone is not defined as a MS but becomes one when used with a SIM-card. In this case, the MS is a microcontroller board together with a GPS/GSM/GPRS module. Communication is maintained as a MS is carrying information which is sent over the radio interface to the BTS. In the GSM architecture, several BTS are connected to one base station controller (BSC) which together provides the radio access network. The services that BSC provides are the different GSM protocols, setups and channel allocation.

To further develop the GSM architecture, GPRS support nodes have been implemented in the architecture. The support nodes are implemented to deliver packets and route packets between MSs and the public data network [3]. The first implemented node is the serving GPRS support node (SGSN). The main function of the SGSN is to forward the communication by routing and sending/delivering packets to the MS within its service area. The SGSN is in addition responsible for the authentication of the MS and logical link control as error checking, flow control and synchronization. In the SGSN information about the GPRS users is created and stored, e.g. their address and IMSI. An additional support node that is implemented in the GSM architecture is the gateway GPRS support node (GGSN). The node is an interface to external networks where GPRS data packets are converted from the SGSN into the desired protocol. Since an internet connection to publish messages on a MQTT broker is desired for this project, the selected protocol that the data packets are converted to is the IP.

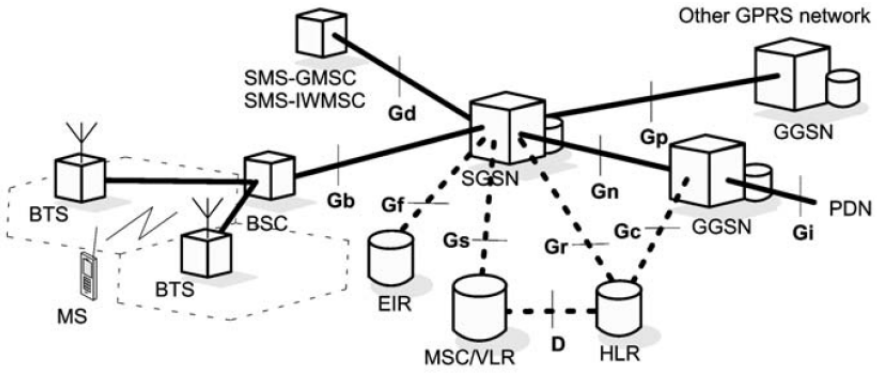


Figure 2.3: Overview of GPRS architecture with its main subsystems GSN, SGSN and GGSN.

2.4 Messaging Queuing Telemetry Transport

MQTT is a lightweight messaging protocol that is suited for IoT projects and Machine to Machine communication [18]. The benefits of MQTT are simple implementations, lightweight, energy and bandwidth efficiency and high data reliability. The MQTT protocol consists of three main elements which are the broker, the subscriber and the publisher. A publisher or a subscriber are not restricted to a specific type of component. A publisher/subscriber may be any device, e.g. a microcontroller, an app, a server etc. One aspect is necessary and common for all publishing/subscribing devices and that is the ability to connect to a network. Using MQTT, there are no restrictions to the amount of devices that can publish and subscribe to a topic/subject on a MQTT broker.

MQTT is built upon a publish/subscribe model where the broker decouples the publisher and subscriber and shares a mutual communication endpoint that is the broker. The broker's task is to receive the message from the publisher, to filter incoming messages and to send the correct message to the correct subscriber. The message filtering is managed in three manners. A subject-based filtering, content-based filtering and type-based filtering [19]. A subject-based filtering is based upon a publisher that publishes the messages to specific topics. As a subscriber subscribes to the specific topic it will receive all the messages that are uploaded to that specific topic by the publisher. A content-based filtering is defined by a subscriber that receives messages with specific contents that the subscriber accepts. A type-based filtering is based upon the type or class of message that the subscriber is expecting to receive. The data type of a common published message on a MQTT topic is a string in a JSON format, more detail of which is presented in section 3.3.2.

The MQTT protocol is built upon the TCP/IP stack and the connection is maintained through handshakes. The publisher performs a three-way handshake with the MQTT broker and the TCP connection is established when the publisher receives an acknowledgment from the server that the MQTT broker exists on. To establish an MQTT connection, a handshake is performed where a connect packet is sent from the client/publisher side to the broker/server [20]. The connect packet contains a connect request with sufficient information that provides the ability to connect to the broker. The contained information in a

connect packet is clientID , cleanSession, username, password, lastWilltopic, lastWillQoS, lastWillMessage, lastWillRetain and keepAlivePeriod. Some of the mentioned parameters are optional while other parameters need to be provided to be able to connect. As a connect packet is sent and all sufficient information is provided, the broker sends an acknowledgment and a connect acknowledgement, as shown in Figure 2.4. As an acknowledgment is obtained, the publisher is able to publish messages to the broker.

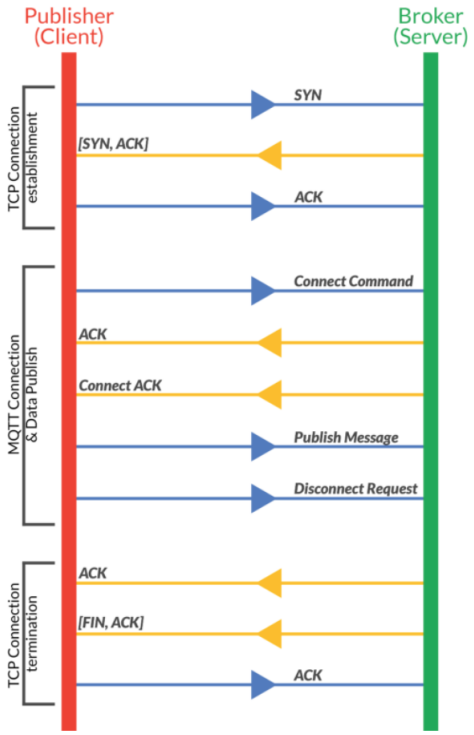


Figure 2.4: TCP and MQTT connection establishment and termination between a publisher and broker.

Packets sent from a publisher to a broker or from a broker to a subscriber are delivered with a certain QoS level. The QoS level is optional and set dependent on network reliability and applied application of the device. In MQTT, 3 QoS levels are defined:

- At most once (0)
- At least once (1)
- Exactly once (2)

If QoS level 0 is selected, packet deliveries cannot be guaranteed and the message receiver cannot send an acknowledgement that a message has been delivered. For QoS level 1, the message receiver will receive the message at least once. As the message is delivered, the receiver sends a puback. The message is stored at the sender side until the puback is received. If a puback is not received for a specific time interval, the sender will resend the message until a puback is received. Messages that are sent twice or more will be assigned a duplicate

flag. If a message is sent with QoS level 2, the sender assures that each sent message is delivered exactly once to the recipient. To assure this, a four part handshake is performed between the sender and receiver. As the mentioned handshake is completed, both the sender and receiver will be assured that the message has been delivered. The QoS level is defined at publisher and subscriber side. The broker that is an intermediary will define the same QoS level that is defined at each endpoint. Hence, when the broker is a receiver, it defines the same QoS level that is defined at the publisher side. While the broker is the sender, it defines the same QoS level that is defined at the subscriber side.

3

Methods and Implementation

3.1 Hardware system with Telit GM862-GPS Module

3.1.1 System Overview

The first setup of a GPS tracking device consists of 6 main building blocks which are the Arduino Uno board, Telit GM862 module, GPS antenna, GSM antenna, logical level converter and two power sources that are 9V alkaline batteries. The GPS/GSM module, Telit GM682 provides the user with the positioning of the vehicle and mobile communication. Through a 50-pin board to board Molex connector, the GPS/GSM module is connected to a Smart GM862 development board to access the different pins. To the GPS/GSM module an internal GSM antenna is connected with an MMCX connector. Considering that the GPS antenna holds a SMA connector, the antenna is connected to the Smart GM862 development board via a MMCX-SMA adapter cable. Here, it is important that at each MMCX connector, which is for each antenna output, an MMCX angle plug crimp is attached, which is performed to maintain a 50 Ω impedance [5]. This is the standard for the majority of coaxial cables. From the Smart GM862 development board, the module is further connected to a logical level converter that converts the high voltage signals from the Arduino board to the low voltage signals that are accepted by the GPS/GSM module. From the Smart GM862 development RX is connected to pin LV1 on the logical converter. Further, RX from the logical converter continues from pin HV1 and further to the TX pin on the Arduino board. The same is performed for the TX line on the development board, but connected to pin LV2 on the logical level converter, continues from pin HV2 and further to the Arduino board. On each side of the logical level converter, the development board and the Arduino board are connected to ground. On the Arduino side of the logical level converter, a 5V line from the Arduino is connected to pin HV on the converter. On the development board side of the converter a 3.3V line from the 3.3V pin on the Arduino is connected to pin LV on the converter. From the same 3.3V node, a line is connected to VCC on the development board. Finally, on each side of the logical level converter, each board is connected to the GND pins. A pull-up resistor of the value 47 kΩ is connected to DTR on the development board and then grounded. RTS

on the development board is further connected to GND. For this setup, two power sources are needed, hence a 9V battery is connected to the pins GND and Vin on the Arduino board. The second battery is connected to the cable that is connected to the CN2 connector on the development board. The wires in the cable of the CN2 connector are exposed and further connected to the wires of the battery. A wiring summary for the Telit module is presented in table 3.1.

Component	Arduino	Level converter HV	Level converter LV	SmartGM862
Battery 1	GND , Vin			
Battery 2				CN2 connector
	3.3V, 5V	HV	LV	VCC
	GND	GND	GND	GND
	RX	HV2	LV2	TXD
	TX	HV3	LV3	RXD
				RTS,GND
Resistor 47kΩ				DTR,VCC

Table 3.1: A wiring summary for the Telit GM862-GPS module. This table complements Figure 3.1.

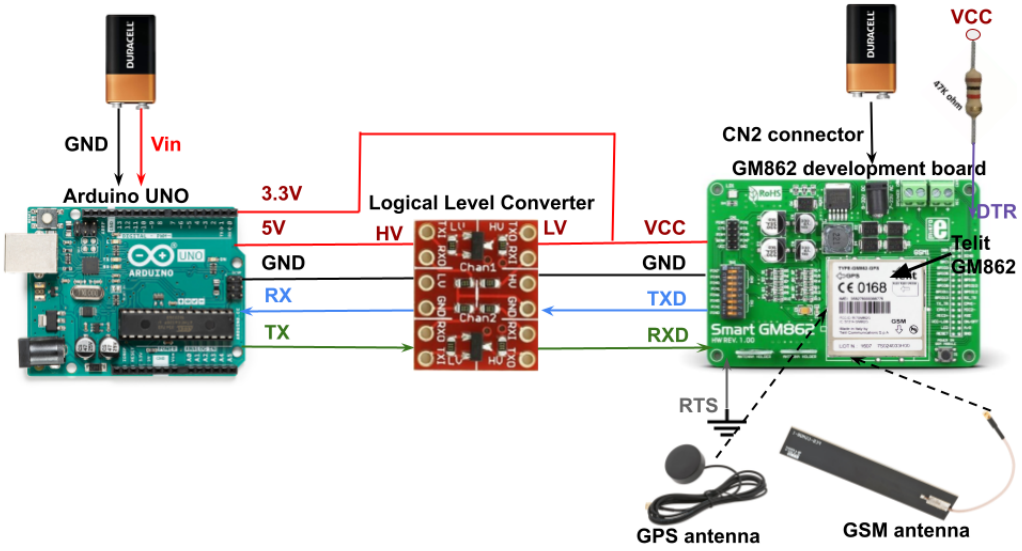


Figure 3.1: The setup of the GPS device that includes the Telit GM862 GPS module. The device consists of an Arduino Uno, GM862 development board, GSM/GPS module, logical level converter, two 9V batteries, GPS and GSM antenna.

3.1.2 Arduino Uno Rev3

For this project a microcontroller is essential, hence the Arduino Uno Rev3 is selected. The board is based on the ATmega328P microchip [11]. The recommended input DC supply to

the board is in the range 7-12V. Hence, for this project, the board is supplied with an alkaline 9V battery that is connected to the CN2 connector on the board, as shown in Figure 3.2.

The serial interface is crucial considering the communication with the GPS/GSM module. Arduino Uno has UART TTL serial communication where the interface is found on pin 0, RX, where the incoming communication is received, and pin 1, TX, where the outgoing communication is transmitted. In Arduino Software (IDE), the RX & TX pins can be reprogrammed to be utilised by other input/output digital pins. The Arduino board has in total 14 digital pins that can be programmed to be used as input/output. The RX & TX signals are modulated as signals with an amplitude of 0-5V.

Connection between the board and a computer is established when an USB 2.0 cable type A/B is used.

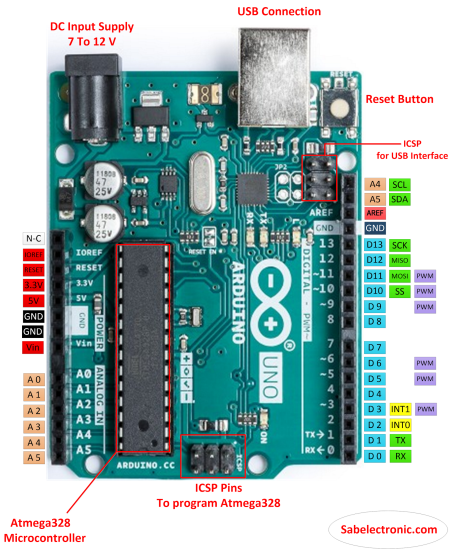


Figure 3.2: Arduino Uno Rev3 and its pinouts and connectors.

3.1.3 Telit GM862 GSM/GPRS/3G-module with GPS

For this project, two hardware setups were constructed. The first setup included the Telit module, as shown in Figure 3.1 but entailed a GPS tracker that could not be put into use due to internal hardware issues.

The module is a combined GSM, GPS and GPRS 3G module. The module is controlled through AT-commands which makes the implementation and communication with a microcontroller effortless and effective. AT-commands are sent from a microcontroller as the Arduino Uno to access the different GPS-module functions as sending SMS, acquiring GPS positions and setting up internet connection. Basic commands of the module include powering the module on or off, initializing the module with the accurate baud rate, GSM signal quality, etc.

The module has a 50 pin vertical SMD Molex board to board connector to connect the mod-

ule to a development board and access all pins.To access the GSM services a SIM card is placed in a SIM card adapter and inserted in the SIM card holder. From the module, GPS and GSM antenna connectors are accessible. On the module side, two 50 Ω MMCX coaxial female RF connectors are attached which gives requirements for the antenna and antenna connectors on the application side. On the applications side a MMCX angle plug crimp has to be attached to the antenna connectors to maintain the 50 Ω impedance. In the hardware user guide of Telit GM862-GPS [5] the following antenna requirements in table 3.2 and 3.3 are listed and have to be fulfilled.

GSM Antenna Requirements	
Frequency range	Depending by frequency band(s) provided by the network operator, the customer shall use the most suitable antenna for that/those band(s)
Bandwidth	80 MHz in EGSM 900, 70 MHz if GSM 850, 170 MHz in DCS, 140 MHz PCS band
Gain	Gain < 3dBi
Impedance	50 Ω
Input power	> 2 W peak power
VSWR absolute max	<= 10:1
VSWR recommended	<= 2:1

Table 3.2: The GSM antenna requirements for Telit G862-GPS module.

GPS Antenna Requirements	
Frequency range	1575.42 MHz (GPS L1)
Bandwidth	+ - 1.023 MHz
Gain	1.5 dBi < Gain < 4.5 dBi
Impedance	50 Ω
Amplification	Typical 25dB (max 27dB)
Supply voltage	Must accept from 3 to 5 V DC
Current consumption	Typical 20 mA (40 mA max)

Table 3.3: The GPS antenna requirements for Telit G862-GPS module.

The selected GSM antenna for the Telit module is an ANT-GSMQB-MMCX PCB antenna with gain 1 dBi, with an included angle plug crimp which provides with 50 Ω of impedance, a VSWR of <= 2:1 and is operating in the frequency bands 850/900/1800/1900 MHz. The selected GPS antenna is operating in the frequency bands 1560 – 1572MHz, have an impedance of 50 Ω, gain 4 dBi, VSWR of <= 1.5, amplification gain 28 dBi and accepts 3-5 V DC.

In the hardware user guide [5], it is clearly instructed that the RXD and TXD pins need a CMOS 2.8 V supply voltage but voltages up to 3.3 V are accepted.

Figure 3.3: GPS/GSM module, model type GM862-GPS from the manufacturer Telit.

3.1.4 SmartGM862 Development System

SmartGM862 by Mikroelektronika is a development tool developed for Telit GM862-GPS modules. As the module is connected to the board via its SMD molex board to board connector, all its pins are accessed. For this project, RXD and TXD pins are necessary to maintain communication between the module and the Arduino. These pins are accessed by selecting the correct pins on the DIP switch as seen in Figure 3.4. For the project, pins 6 and 7 were switched to the right on the DIP switch and two female jumper cables were connected to pin 6 and 7 on the 2x5 male header. Two additional female jumper cables are connected to the 2x5 male header to pin GND and VCC. According to the Telit hardware user guide [5], the RTS pin on the 2x5 male header should be connected to GND if not used, and therefore the pin is connected to an arbitrary GND pin on the Arduino. It is also stated that the DTR pin is not pulled up internally and therefore a 47 kΩ resistor is required.

The required power supply is 7-23V AC or 9-32V DC power supply voltage. The module is power supplied with 12 V through a CN2 connector on the board and the whole module is powered on when the power key is pressed and held for a minimum of 1 second.

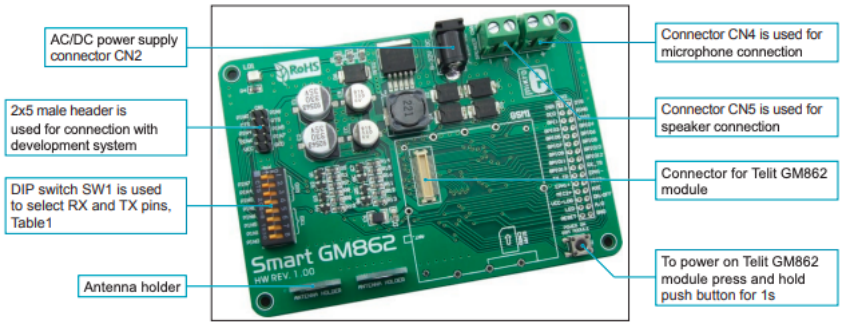


Figure 3.4: The GSM/GPS module Telit GM862-GPS is connected to the development system SmartGM862 to obtain accessibility of the pins.

3.2 Hardware system with SIM808 Module

3.2.1 System Overview

The second system setup is constructed with a SIM808 GSM/GPRS/GPS Arduino Shield by Waveshare. The shield is connected to the Arduino by attaching the pins of the shield into the pins of the Arduino. Shown in Figure 3.5, the following hardware setup is presented:

1. USB cable, type B connected between a computer and the Arduino.
2. A micro USB cable connected to the shield, mainly used for UART and PuTTY applications, more detail of which is presented in section 3.3.4.
3. CN2 connector to power the Arduino shield. Supplied by DC power, 6-9V.
4. The shield is powered on or off via the switch.
5. To select the appropriate VREF, the 2-pin jumper is shifted to another pin.
6. The GPS antenna is connected here.
7. Power the SIM808 GPS module by holding the PWRKEY button for 1 second.
8. The Bluetooth antenna is connected here, but not applied for this project.
9. The GSM antenna is connected here.
10. UART configuration for SIM808. In the Arduino code, the RX and TX pins are selected and the 2-pin jumpers are connected according to the selection.
11. Select the UART communication port via the switch. When desiring communication via the CP2102, "USB" is selected, and when Arduino communication is desirable, "D0/D1" is selected.

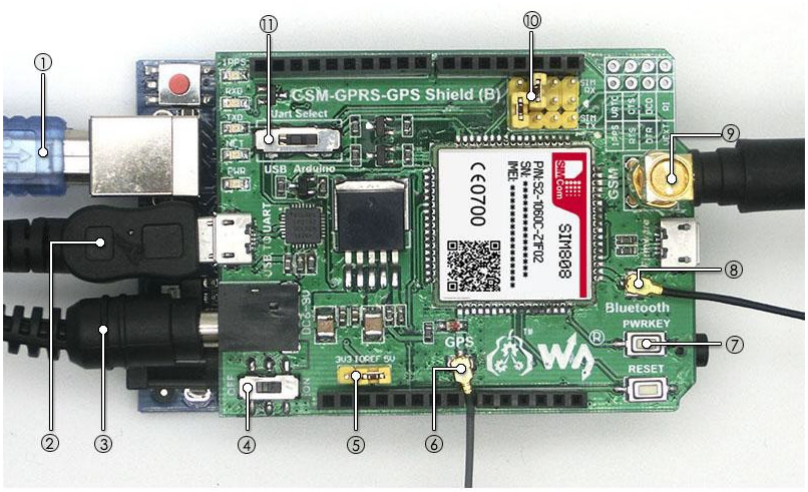


Figure 3.5: Hardware system overview with SIM808 Arduino shield. The shield is connected on the Arduino Uno.

For a remotely connected device the module is connected according to figure 3.5 with a 9 V battery as power source that is connected to Vin and GND on the shield.

3.2.2 SIM808 GSM/GPRS/GPS Arduino Shield

The Arduino shield is based on the SIM808 GPS-module and is compatible with Arduino Uno. No jumper wires are required. The shield is directly attached to the Arduino board via its pins and connected immediately. On the backside of the shield, as shown in Figure 3.6, the SIM-card is inserted if functions as GSM and GPRS are necessary. For this project, a 3D-printed SIM-card adapter together with a nano SIM card was used.

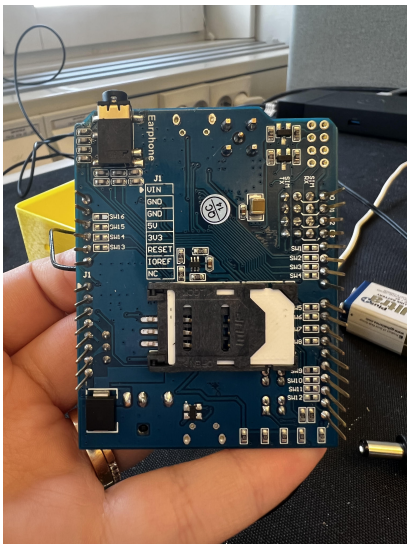


Figure 3.6: The backside of the Arduino shield with an inserted SIM-card adapter

The shield was ordered from Waveshare, a retailer that offers electronics, development boards, etc. The obtained package included the shield, a Bluetooth antenna, GSM antenna, a GPS antenna, an EU standard adapter and a USB cable type A to micro. Unfortunately, no information could be retrieved from the retailer about the characteristics of the accompanying GSM and GPS antennas except the information that is printed on the GPS antenna and presented in Appendix A.4 Component List.

3.3 Software and Installations

3.3.1 Communication Overview

The main goal of this project is to obtain GPS locations of the tracked object and present it on the developed mobile application on Google maps. The location is retrieved from the GPS module that is implemented on the Arduino shield. Later, the obtained location is decoded from National Marine Electronics Association (NMEA) format to a JSON format and published on a MQTT broker with GPRS services, more detail of which is presented in section 2.4. To the broker, multiple devices are able to subscribe. For this project the subscriber is a mobile phone with the installed Object Tracker app. As a position is retrieved in the app, the location is decoded from JSON format to a LocationData format that the Google maps widget acquires. The widget takes a position in latitude and longitude and presents it on a Google map with a red marker.

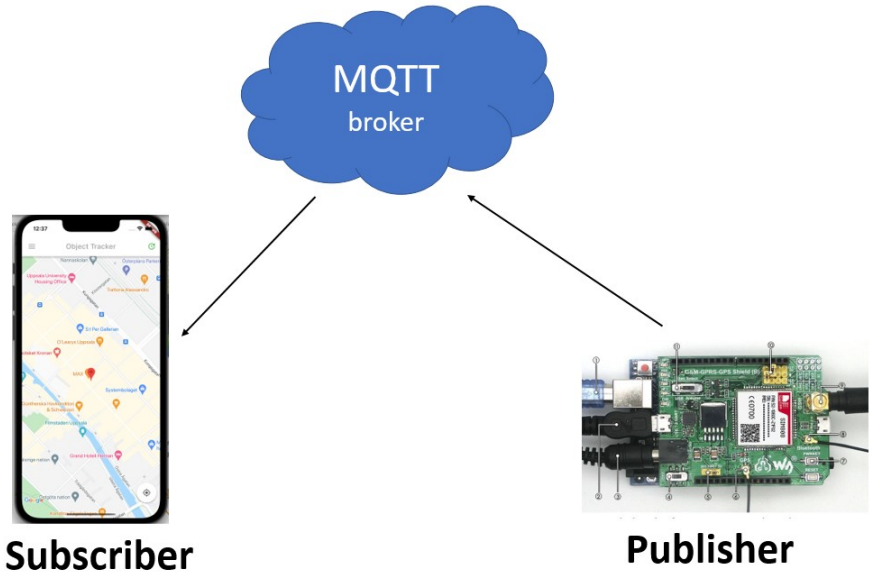


Figure 3.7: Communication overview between the hardware Arduino and Arduino shield, MQTT broker and Object Tracker app.

3.3.2 Arduino Software and Libraries

The Arduino code was developed in the Arduino environment, the Arduino Software IDE. The software contains a code editor, a window with message output, a menu for different tools and buttons for code verification and uploading to the board. The applied programming language for Arduino is C++. The code was written with the three aiding libraries TinyGSM, ArduinoJson and PubSubClient.

The TinyGSM library [13] supports different GPS modules, including SIM808 that is used for this project. A predominant amount of GPS/GPRS/GSM modules uses AT-commands when the user desires to access the module functionalities. The mentioned AT-commands are implemented into functions in the TinyGSM library. Hence, an alternative to sending multiple commands to achieve a GPRS connection e.g., a simple function may be applied. Hence, a more neat and reader-friendly code is obtained.

As data is transmitted from hardware to a MQTT broker and from the broker to the app, it is desirable if the data is compact and lightweight, which is achievable with JSON formatting. JSON format is applied for data interchange within servers and web applications [16] and is suitable when transmitting to and from the MQTT broker. The ArduinoJson library [14] aids with serialization and creating a JSON formatted object that is later sent to the MQTT broker. An example of a JSON formatted object message containing latitude and longitude information is:

{"latitude":59.85453,"longitude":17.65196}

The PubSubClient library [15] is an MQTT messaging library that provides functions with the possibility to connect and publish or subscribe to a MQTT broker. The library supports the latest MQTT protocol version 3.1.1. The library is able to publish messages to a topic with QoS level 0 and is able to subscribe to topics with QoS level 1 and 2. At the subscriber side (the app), the set QoS level is 1. more detail of which is presented in section 2.4.

3.3.3 Arduino Code

The code that is developed for the Arduino board is responsible for retrieving the GPS position, JSON formatting the message and thereafter publishing the message to the MQTT broker, Adafruit IO. Initially library variables are set. A very crucial function is the command function that sends any AT-command from the Arduino through the serial port, to the serial port of the shield. Later, a desired baud rate is set for the Arduino and the shield. The serial communication is set to baud rate 38400 bps due to communication issues that arose when the baud rate was set to 115200 bps, which was to high for the Arduino to process. Using the command function and the functions that are implemented in the TinyGSM library, the GPS and GPRS functions are set to on. Thereafter a loop is initiated where a GPS position is retrieved each 20 seconds and the latitude and longitude values are stored in variables. These variables are later imported to a build JSON function that converts latitude and longitude values to a JSON formatted message. Thereafter a connection to the broker is set up

and the message containing the latitude and longitude values are published. Thereafter the loop restarts. During the loop the connection to the broker is always checked and if no connection is upheld, a new connection is initiated. In Figure 3.8 a flowchart of the described code is presented.

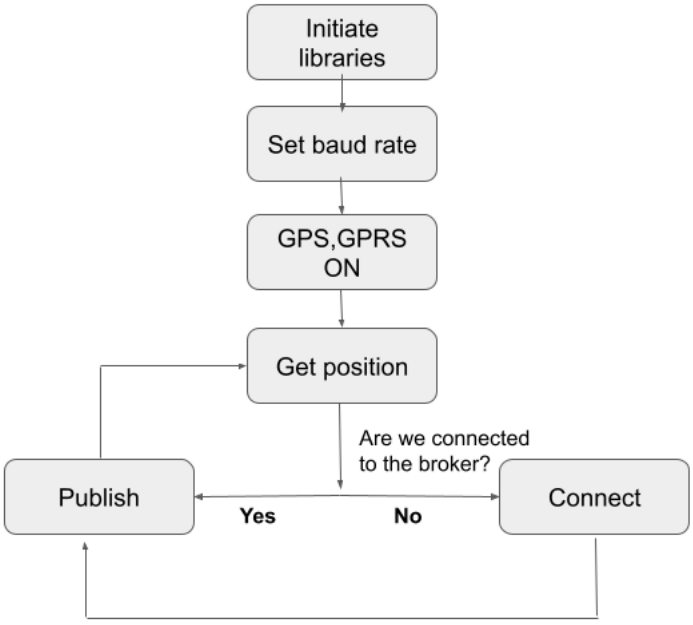


Figure 3.8: The flowchart of the Arduino code.

The Arduino code for this project is written with the aid of the examples published in the GitHub pages for the TinyGSM library [13], the ArduinoJson library [14] and the PubSubClient library [15].

3.3.4 PuTTY

PuTTY is a terminal emulator that supports different network protocols as SCP, SSH, Telnet, rlogin but is also applied for serial communication [17] which is the prime user area of this project. On the Arduino shield, the UART communication was selected via a switch, which is presented in section 3.2.1. On the switch, "USB" was selected to initiate the serial communication with the PuTTY emulator. A PuTTY emulator was downloaded to examine the communication of the Arduino shield, mainly to assure that the correct information is obtained from the Arduino shield while an AT-command is sent. Some coding was in addition based upon the incoming answer from the GPS module, thus the answer was examined initially through PuTTY. The PuTTY emulator was also applied to assure that the correct baud rate was set for the serial communication when communication issues arose between the shield and the Arduino Uno board. When initiating serial communication with PuTTY, the following settings were applied, in addition to selecting the correct COM port:

- Baud Rate - 38400
- Parity - None
- Stop bits - 1
- Data bits - 8

Inserting the correct settings should initiate a PuTTY emulator session and serial communication with a device. According to the software user guide for the Telit module [5], the baud rate can be selected from the range 0,300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 bps. The same is valid for the Arduino shield.

3.3.5 Adafruit IO

Many brokers exist, some are free, some are free to a certain limit and some contain monthly charging plans. Adafruit IO brokers include a free plan and a billing plan. In the free plan the following is included:

- 30 data points per minute
- 30 days of data storage
- 10 topics/feeds

, which fulfills the requirements of the project. For the project, one topic (called feed in Adafruit IO) where the latitude and longitude values are published from the Arduino, is required. Messages from the Arduino are sent to the broker and feed every 50 seconds, which is set in the Arduino code. There are no requirements of storage since only the last incoming latitude and longitude value is applied to the app.

Adafruit supports only QoS level 0 and 1 [21] and for that reason the QoS level is not set to a higher level in the app. To connect to the Adafruit IO broker the user is able to select among 3 socket ports depending on the application. If the application includes a webbsocket client, socket port 443 is recommended. For other applications one is recommended socket port 1883 which is an insecure socket port and 8883 which is a secure SSL socket port. Further sufficient information that must be provided is

- **Host** - io.adafruit.com
- **Username** - Your Adafruit IO username
- **Key** - Your Adafruit IO key

The Adafruit server limits the amount of publish actions that can be made to a topic to prevent from service overload. The limit for a free account is 30 points per minute and for a billing account is 60 points per minute. If the publish action rate is exceeded the user account may be banned.

3.4 App Development

3.4.1 Flutter Software and Packages

Flutter is an UI multi-platform software development kit. It is used as a cross-platform for development of apps for Android, Linux and iOS. Flutter provides development for different OS's while using one single codebase, which makes Flutter a popular cross platform. Flutter is developed by Google and uses the programming language Dart.

For this project different Flutter packages were applied to the code.

- **material** - The material.dart package [22] contains multiple icons and widgets. It is the most important package for visualizing content on the app and is the main building block for the interface.
- **location** - From the location.dart package [23] the most important class for this project is the LocationData class which takes the latitude and longitude parameters. The plugin is also able to retrieve an app user's current location and handle callbacks for changed locations.
- **mqtt_client** - From the mqtt_client.dart package [24] is a v.3 MQTT plugin. From this plugin, the app user is able to connect, publish and subscribe to a broker. The plugin is able to subscribe and publish on all QoS levels and supports secure and non secure socket ports. From the package source a GitHub repository is linked and examples are found and used as a template.
- **convert** - The convert.dart package [25] is able to encode and decode between different data types. For this project, the convert package is applied to convert a JSON-formatted message to a LocationData format.
- **google_maps_flutter** - To display a Google maps widget on the app, the google_maps_flutter.dart plugin [26] is implemented. It is necessary to retrieve a Google maps API key to make this plugin useful, more detail of which is presented in section 3.4.3.

3.4.2 Installations and Editor

The app development and installations are performed on a MacBook Air from early 2015 with macOS Monterey. The installation of Flutter is performed with the aid of online guidelines [7]. Prior to the installations, the user must ensure that an amount of 40 GB of free disk space is available. For an iOS setup, Xcode and iOS simulator is installed. To deploy the future developed app with Flutter plugins on a real device, Cocoapods is installed. For an Android setup, Android Studio and an Android emulator is installed. The installations are all performed with a Homebrew terminal. Adding the path of the Flutter SDK to the resource file of the shell, the user is able to use flutter commands permanently on any terminal window. The resource file is edited using the following command:

```
open  /.bash_profile
```

An example of a path of a Flutter SDK inserted in a resource file is:

```
export PATH="$PATH:/Users/ivetshami/Downloads/flutter/bin"
```

It is recommended to send the command `Flutter doctor` in the terminal to detect any missing installations. The installed editor for this project is Visual Studio code where all code for the app was written. A Flutter project is selected as an application from the command palette. From VS code, the user is able to select a device on a simulator that the app is presented on throughout the code developing process. This is also possible to execute on a terminal window by sending the command:

```
open -a Simulator
```

The user is able to run the code from VS code or from the terminal by sending the command:

```
Flutter run
```

As the code is developed further, the user is able to select "Hot Restart" to restart the app or "Hot Reload" to reload the app. Also this is possible in the terminal by sending the command `R` + enter for Hot Restart or `r` + enter for Hot Reload following the `Flutter run` command.

3.4.3 Google Maps API

A sufficient part of the project and app is to present a certain position on a Google map. This is only possible if a Google maps API key is retrieved from the Google Cloud Platform. On the Google Cloud Platform page, create credentials is selected and later API key selected and created. On the same page it is also possible to restrict the API key to certain API's and applications. For this project, no application restrictions are performed and the API key is selected to be restricted. On Google Cloud Platform page, API's are selected in the navigation bar and thereby the user is able to enable different API's. For this project the enabled API's are Maps embed API, Maps JavaScript API, Maps SDK for Android and Maps SDK for iOS.

For each created Flutter project, a folder is created. The folder contains subfolders with Android files, iOS files, lib files where the code exists and etc. The obtained key is specified in the Android and iOS files that exist in their corresponding folders. Selecting the AndroidManifest file in the path `android/app/src/main/AndroidManifest.xml`, the API key is inserted as follows

```
<manifest ...
  <application ...
    <meta-data android:name="com.google.android.geo.API_KEY"
               android:value="INSERT API KEY HERE"/>
```

. For the iOS setup the file in the path `iOS/Runner/AppDelegate.swift` is edited as follows:

```
import UIKit
import Flutter
import GoogleMaps

@UIApplicationMain
@objc class AppDelegate: FlutterAppDelegate {
  override func application(
    _ application: UIApplication,
    didFinishLaunchingWithOptions launchOptions:
        [UIApplication.LaunchOptionsKey: Any]?
  ) -> Bool {
    GMSServices.provideAPIKey("INSERT YOUR API KEY HERE")
    GeneratedPluginRegistrant.register(with: self)
    return super.application(application, didFinishLaunchingWithOptions:
        launchOptions)
  }
}
```

A detailed description of the mentioned steps are also available on the Google maps Flutter package page [26].

3.4.4 Object Tracker Application Code

The application code was developed following examples published for each GitHub page of each Flutter package. Similar projects have been investigated and the Flutter code has been developed according to the desired widgets and functions. The MQTT functions was developed following a similar IoT tracking project [27]. Connection to the Adafruit broker is accomplished by following the Adafruit MQTT example functions [28]. Implementing a google map widget is performed following the guidelines for how to add a google map to a Flutter app [29]. The challenging part of this project is obtaining a location that later is visualized on the map with a marker. For every new position, the marker is added to the position and the map of the new location is shown. A software engineer of the name Angga Dwi Arifandi has developed a similar tracking project [30]. The aim of his project is to develop an Uber-like app where the app user location is visualized on a Google map. The code contains three necessary functions that are implemented in the Object Tracker application. The first function, gotNewLocation acquires the obtained position and calls the second function animateCameraToNewLocation and navigates the user to the new position on the Google map, and the third function that converts a message in JSON format to a LocationData format that the Google map widget acquires.

During the course of the code development an MQTT connection error was obtained for each time the code was run. The problem was solved in two steps. The first solution to the problem is adding internet permission to the AndroidManifest file as follows:

```
<manifest xmlns:android...>
```

```
 ...
 <uses-permission android:name="android.permission.INTERNET" />
 <application ...
</manifest>
```

The second solution to the problem is to connect to a different network. The first network connection was the guest network of the company Prevas that this project was developed at. The network contained a firewall that prohibited the app from connecting to the broker. Connecting to a private network solved the issue.

The Object Tracker code has throughout the development been simulated on an iOS simulator. The code has not been simulated on an Android emulator. Hence, it cannot be guaranteed that the code is compatible with an Android device but should be regarding Flutters characteristics.

4

Results and Discussion

4.1 Final Prototype

In Figure 4.1 the final prototype of the GPS tracker is presented. The prototype consists of an Arduino Uno and Arduino shield as hardware mounted in a 3D printed case. On the computer screen the app simulator together with the Adafruit broker is presented where the tracker has provided the position of Prevas AB in Uppsala to the broker. On the broker page it can be noted that a position has been published to the topic `IvetSJ96/Feeds/object_tracker` in a JSON format. The app has subscribed to the topic and acquired the position and presented the acquired position on the Google map widget. In the following sections the GPS tracking device, the app and the unsuccessful configuration containing the Telit GM862-GPS module is presented and discussed in detail.

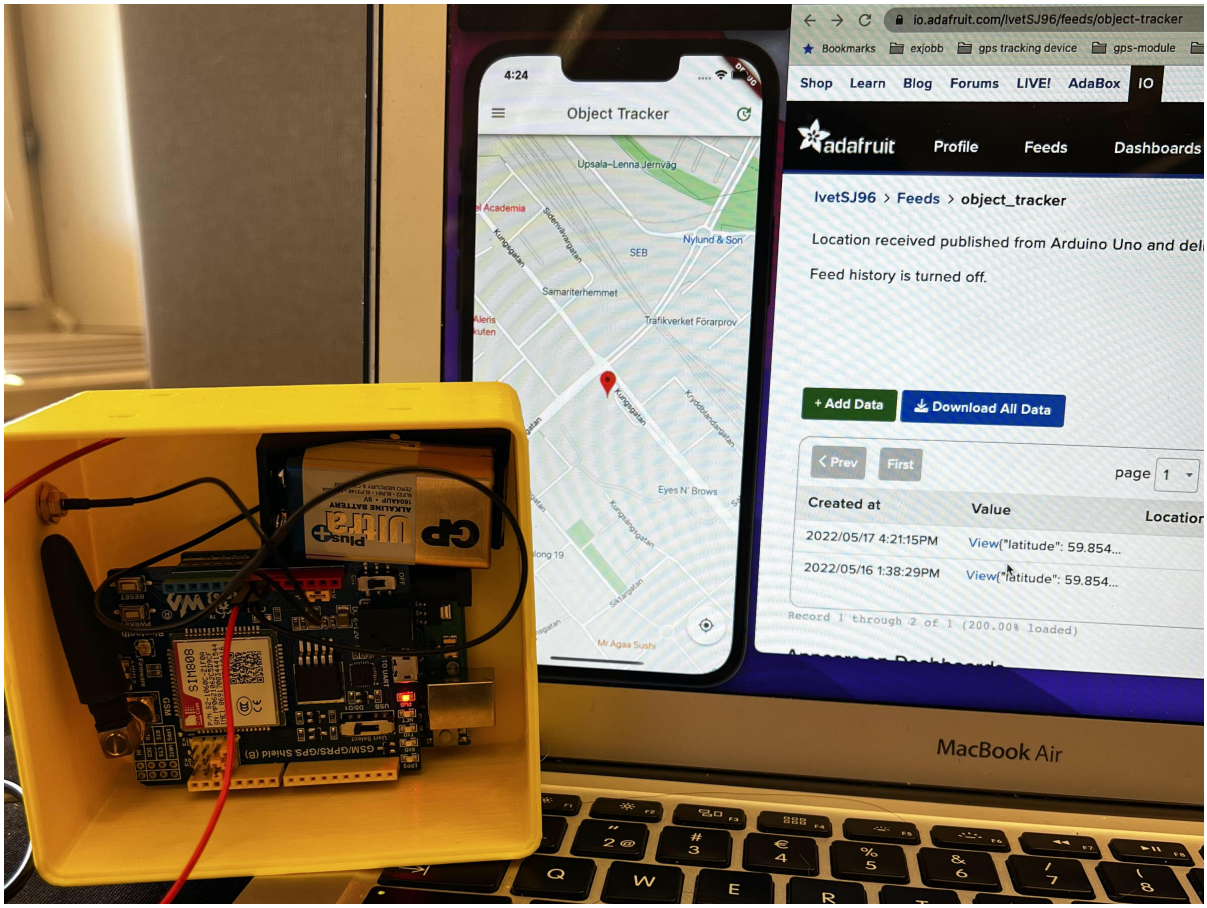


Figure 4.1: The GPS tracker system including the hardware, Adafruit broker and app simulator.

4.2 GPS Tracking device

No results were achieved with the Telit GM862-GPS module due to defective hardware. Due to that reason, the module was changed to the Arduino shield with the SIM808 GPS module and a prototype was obtained. The shield and Arduino Uno are connected according to section 3.2.1. Further the hardware is mounted in a custom made box, as shown in Figure 4.2, 4.3 and 4.4. The box is developed in the software Creo by a Prevas colleague and later 3D printed. The device is mounted on the box with three screws which prohibit the electronics from moving freely in the box. On the side of the box the SMA connector of the antenna is attached. On the backside of the box, 4 holes are constructed to be able to insert cable ties and make it possible to attach the box onto an optional vehicle. The PWRKEY is pressed for a second on the Arduino shield and the program that is uploaded on the Arduino Uno is executed. The program is executed correctly if the 1PPS LED flashes every second, the NET LED flashes fast and continuously (when connection to the network has succeeded), and the PWR LED light is on continuously.

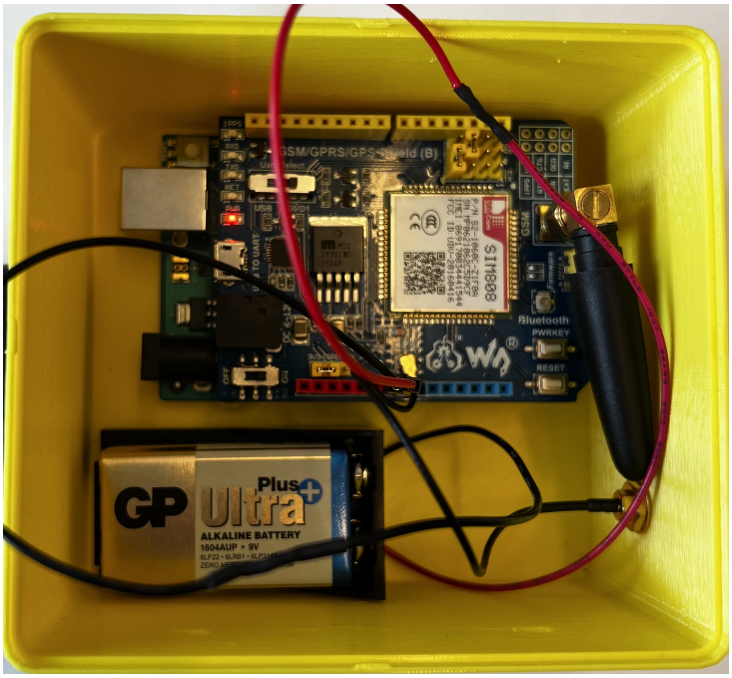


Figure 4.2: Final proof of concept GPS tracker prototype seen from above.

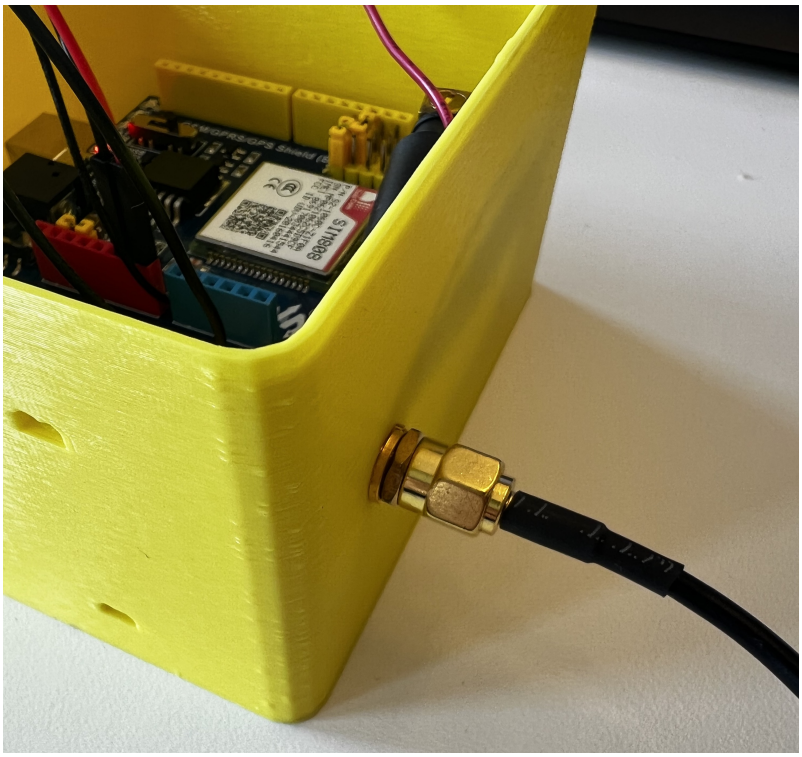


Figure 4.3: SMA connector of the GPS antenna attached to the surface of the box.

Figure 4.4: The GPS antenna and holes on the backside of the box to be able to attach the box onto a vehicle with cable ties.

Throughout the project development memory issues occurred as this message was displayed:

```
Sketch uses 22810 bytes(70%) of program storage space.  Maximum is 32256 bytes.
Global variables use 1301 bytes (63%) of dynamic memory, leaving 747 bytes
for local variables.  Maximum is 2048 bytes.
```

The message is not a warning message but a regular message that is displayed for each time that the code is compiled. In the serial monitor of the Arduino environment, outgoing serial communication is displayed but no serial communication from the Arduino shield was obtained. As a piece of code was commented, the occupied program storage space and the dynamic memory was diminished and the serial communication was processed appropriately. The code was optimized and variables were reduced to reduce the occupied storage space. The optimization resulted in a reduction of a few percent that did not affect the serial communication issue. As the issue was investigated further, it was shown that the lines

```
Serial.println(...);
```

,did occupy the RAM memory of the Arduino which implied serial communication issues. The issue was solved by editing the Serial.prinln lines to

```
Serial.println(F(...));
```

, which assures that the string is not occupying the RAM memory and is occupying the program memory instead.

4.3 Object Tracker Application

The object tracker app does fulfill the project specifications. To the app, a Google map widget is shown. On the right of the app bar, a green update icon is implemented and on the left of the app bar a menu drawer is implemented, as shown in Figure 4.5. When the green

update icon is pressed, the app is connected to the Adafruit broker and listens to incoming subscribe messages. Messages are obtained and reformatted from JSON format to LocationData format. The position is saved in a variable that is presented with a red marker on the app. For each incoming message, the variable is updated and presented on the map widget. If the marker is pressed, an information box is shown with the text "Your vehicle is here!".

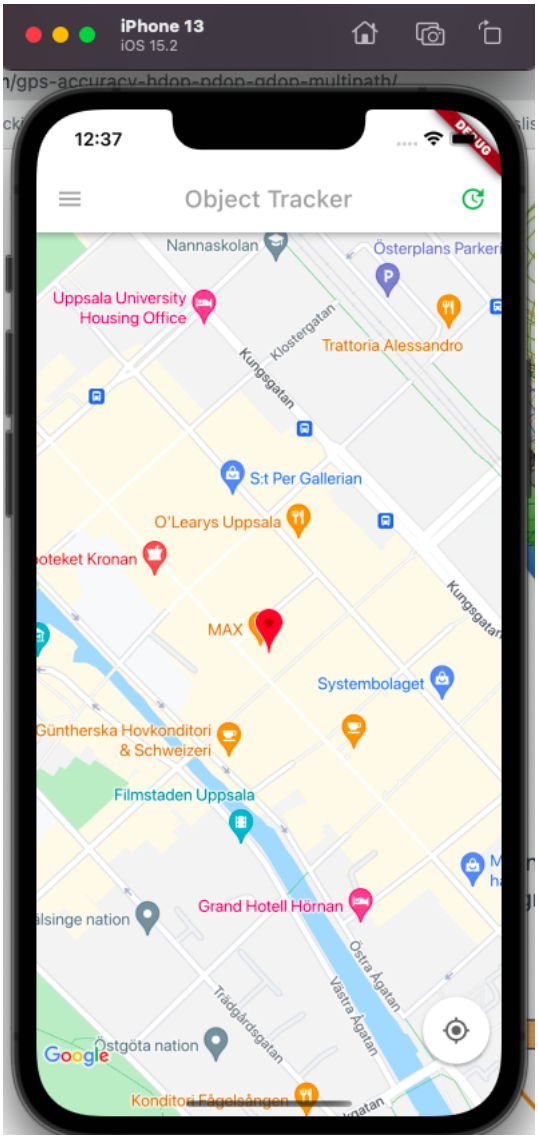


Figure 4.5: A screenshot of the Object Tracker app run on an iOS 15.2 iPhone 13 simulator. A position has been obtained, reformatted to LocationData format and visualized on the map.

If the menu drawer icon is pressed the options "Home", "Settings" and "Parking" are shown, as shown in Figure 4.6. If the "Home" option is pressed, the user is navigated back to the intro screen with the Google map. Currently if the user presses "Settings" and "Parking", nothing will appear due to unimplemented functions.

For this project Flutter is selected as a development software considering its great usage among developers and its multi-functionality for different OS's. The software is considered

to be the future of development platforms. The second basis for the software selection is its Google libraries. The aim is to develop an app that provides the location of the tracking device by displaying it on Google maps. Therefore, using Flutter as an application is beneficial due to its great synchronization with Google products.

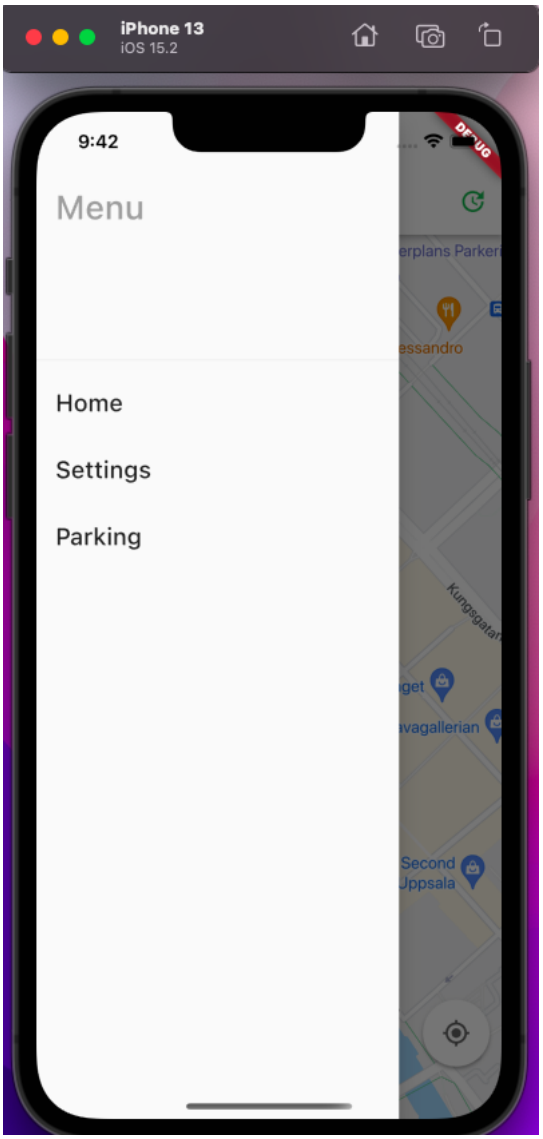


Figure 4.6: A screenshot of the implemented menu drawer of the app.

4.4 Performance

To examine the performance and limitations of the GPS tracking device, a test of how well a GPS receiver can estimate the actual position was conducted. The GPS tracking device was positioned in three different environments. An outdoor environment, an indoor environment and an outdoor environment where the antenna was placed on a piece of metal (a windowsill). A total of 5 measurements were noted for each environment and for each ob-

tained position a marker was placed. The exact measurement position was estimated and noted with a red marker. The markers are placed in a Google map to visualize the results, as shown in Figure 4.7.

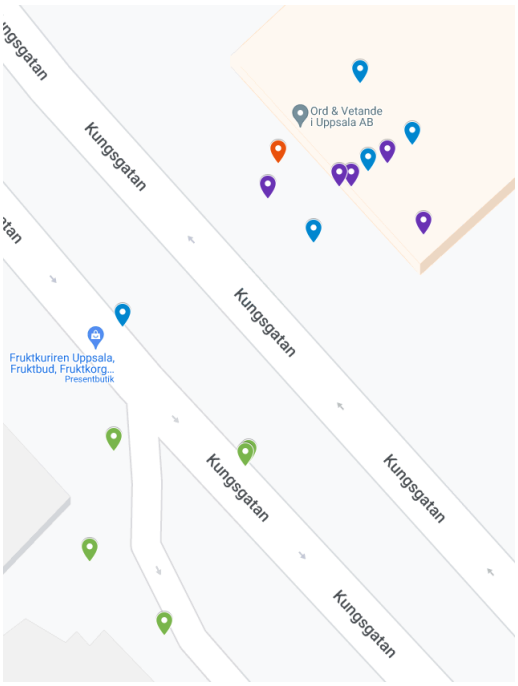


Figure 4.7: A visible presentation of the performance of the GPS receiver of the SIM808 module. The red marker represents the exact measurement location, the blue markers represent the indoor measurements, the purple markers represent the outdoor measurements and the green markers represent the outdoor-metal measurements.

From the measurements it can be stated that the best estimation of the exact position (red marker) is given by the outdoor measurements. Using the measurement tool in Google maps, the best estimated position is estimated to be 2 m from the actual measurement position. The poorest estimation of the outdoor measurements is 15 m from the actual position. The second best measurements were obtained by the indoor measurements. The best estimated position was 8 m from the actual measured position. The poorest estimation was 21 m from the actual position. It is worth mentioning that the antenna for the indoor measurements was placed near a window and therefore poorer results would have been obtained if the antenna was placed in the middle of the room. The poorest estimation was obtained by the outdoor-metal measurements. The best estimated position of the outdoor-metal measurements was 27 m from the actual position and the poorest measurement is 45 m from the actual position. A summary of the estimations are presented in table 4.1.

Environment	Best estimated distance [m]	Poorest estimated distance [m]
Outdoor	2	15
Indoor	8	21
Outdoor-metal	27	45

Table 4.1: An overview of the best and poorest estimation for each environment that the measurements where conducted in.

The results implies that the placement of the antenna should strongly be considered when further developing the tracking device. As a suggestion, if the tracking device is aimed to be assembled on a bike, the tracking device should preferably be placed inside the handlebar with the GPS antenna facing out. According to the examined GPS receiver performance in section 4.4, the receiver is mostly affected if metal is placed nearby, hence the accuracy of the GPS position may be affected if too much metal is surrounding the antenna on the same plane that the antenna is placed on . If the GPS tracking device is developed for a car, the device needs to be placed in the middle of the car to avoid signal reflections and multipath effects to the furthest extent.

4.5 Telit GM862-GPS

Primarily the project is aimed for the use of Telit GM862-GPS as a tracker and communication device with a broker, which was suggested by the subject reader as the module has been succesful in other projects. The module is connected to a SmartGM862 development board and connected according to the Telit GM862-GPS hardware guide [5]. The final tracker is presented in Figure 3.1. As the Arduino code was uploaded, no serial communication was obtained in the serial monitor. Troubleshooting was performed on the SmartGM862 board using the provided circuit diagram, presented in Appendix A.3 SmartGM862 Circuit Diagram, and an oscilloscope. One probe of the oscilloscope was connected to ground and the second probe was moved around to detect the incoming signals from the Arduino. The Arduino sends the communication from its TX pin, which is connected to a level converter and further connected to pin 7, that is selected as RXD on the DIP switch on the development board. Referring back to Appendix A.3 SmartGM862 Circuit Diagram, it can be noted that the RXD signal is measured at R7 and R5, which is where the signals are measured. The TXD signals, the module response are instead measured at R8. The conclusion of the troubleshooting is that incoming signals to the module, measured at R7 and R5 could be detected, as shown in Figure 4.8, but no TXD signals could be detected at R8, which concludes in that the hardware does not reply to the AT-commands it receives. The serial communication was set to different baud rates but no response from the module was obtained.

Figure 4.8: Measurement of RXD signals at R5 and R7 to assure that the correct AT-command is sent. Here the AT-command "AT+CREG?" is sent to the module to check the network availability.

In Figure 4.9 an example for a correct response of a module is presented. The presented response is obtained from the SIM808 module when the command "AT+GNSINF" is sent which require the GPS position of the GPS tracker. The oscilloscope demodulates the waveform to ASCII, which later is demodulated to characters and can be read in human form.

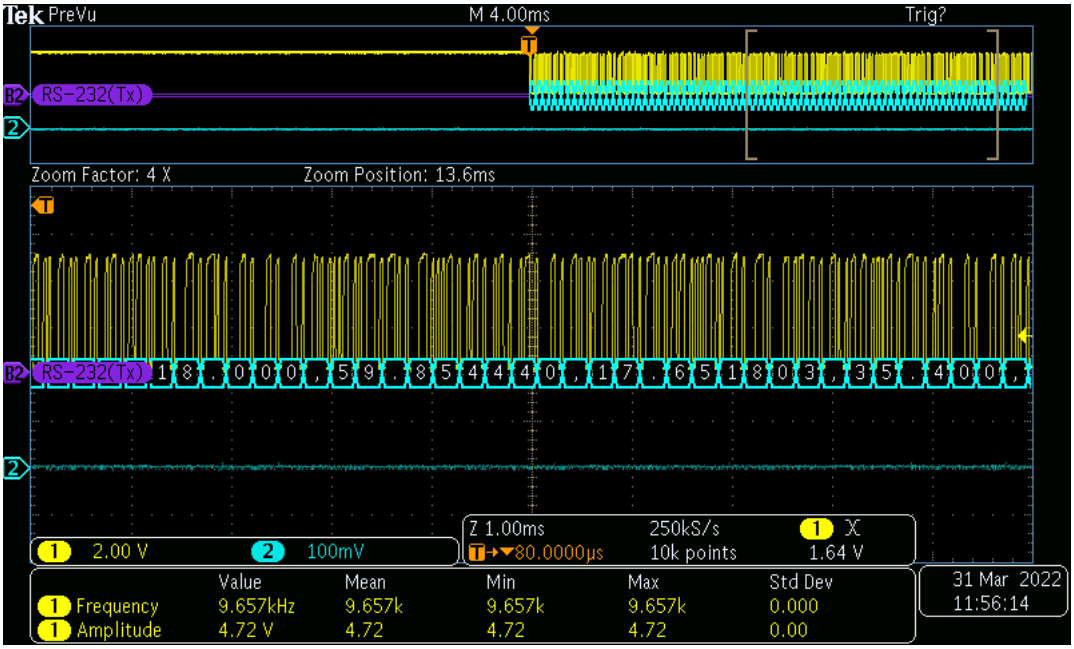


Figure 4.9: Examined serial communication on the oscilloscope. Here, a successful GPS location reply is obtained from the Arduino shield when the correct baud rate is set.

In the hardware user guide it is mentioned that the Telit module acquires 2.8 V RX and TX signals but signals up to 3.3 V are tolerated, which is the voltage level of the logical level converter that converts the RX and TX signals from 5 V to 3.3 V. To assure that the signals are transmitted with the correct voltage level, a voltage divider is applied to the 3.3 V node to obtain a voltage level that is 2.8 V, as shown in Figure 3.1. It resulted in no change and no response was obtained from the module. According to the hardware user guide pin VCC_LOG is set to high when the module is on and set to low when the module is off. This is examined and is correct for when the power button is pressed for a few seconds, which concludes in that the module does receive the needed supply voltage. From additional troubleshooting tests, it could be concluded that the hardware had some internal defection, which was also confirmed when the manufacturer Telit was contacted for eventual support.

It is not recommended to implement the Telit GM862-GPS module for different reasons. The module is discontinued and the manufacturer does not provide any support for the module. The tracking device that consists of the Arduino Uno, the logical level converter and the SmartGM862 development board is bulky in comparison to the tracking device that consists of the Arduino Uno and Arduino shield. Another disadvantage with the Telit module is that it is rather complicated to connect to the Arduino board where many things need to be taken into account but is favorable for learning purposes. Implementing the Arduino shield rather than the Telit module entails less complicated connections and code due to the available libraries for the SIM808 module.

5

Conclusion and Future Work

5.1 Conclusion

Throughout this project two hardware setups were constructed to act as a GPS tracker device. The first setup, which contained a Telit GM862-GPS module, had internal hardware issues and could not be implemented for the project. It is not recommended to implement the module for future projects due to the discontinuation of it and for that reason the manufacturer can not provide any support. The second successful setup contains an Arduino and an Arduino shield that retrieves a GPS position in NMEA formatted string. From the string the latitude and longitude values were set and formatted to a JSON formatted message. The message of the position is published through GPRS services to an Adafruit broker. From the Adafruit broker, the position was retrieved by the Object Tracker app and the position was visualized on a Google map with a red marker. All of the desired functionalities, listed in section 1.3, were achieved but the tracker requires sizing improvements, Arduino code improvements to save power and the Object Tracker application code usability. GPS tracker performance tests entail that the accuracy of the position is very much affected by the antenna placement and should strongly be considered during the further development of the tracker.

The challenging part of this project was the developing of the Object Tracker application. The programming language Dart was unfamiliar and difficult to understand. However, a lot of guidelines could be found online and many examples to follow from to build an app after specific requirements. Dart has also many implemented functions for respective Flutter package which makes the code development fast and productive.

5.2 Future Work

A final prototype and proof of concept was obtained with the Arduino shield and the Object Tracker App. The GPS position is retrieved from the SIM808 module and published with its

GPRS function to the Adafruit broker. Later the position is retrieved from the broker and visualized on the Google map that is implemented on the Flutter application. However, considering it being a proof of concept there is much room for improvements. The application has the basal functions but is not user friendly. The first suggested improvement is to notify the user if the GPS position of the vehicle has changed. For the moment, if the user presses the update icon, the app is connected to the broker and listens for incoming messages. This could preferably be a managed backend where the app connects to the broker, retrieves the position from the broker and compares the new position to the old position every two minutes. If the new position does differ from the old position, the app user should be notified. However, this will result in a user that constantly will receive notifications about a position change when the user is the one that is utilizing the vehicle. To prohibit this, a parking function has been implemented in the app but not developed, presented in section 4.3 and shown in Figure 4.6. The parking function is selected from the menu navigation and the user is able to select to park or unpark the vehicle with a toggle switch. The backend function of the toggle switch is to select if the notifications are on or off. When the vehicle is parked, the toggle switch will be set to on, which implies that notifications will be on and the user will be notified if there is a position change. Additional unimplemented function on the menu navigation is "Settings". On the settings page general information about the user should be provided as name, mobile number, email address and living address. From the settings page the user can also select to be notified through email or SMS if a position change has occurred. The user should also be able to select if notifications should be provided for a specific time frame of the day or always.

From the Arduino side, locations are published to the broker every 50 seconds but the loop is set to be repeated every 20 seconds, and hence a location should be published to the broker every 20 seconds but that is not the case. The GPSPosition function is built upon the incoming answer from the SIM808 module. The function acquires the incoming answer that is a string and assigns the latitude and longitude variables the values that are presented on a specific position in the string. Every other time that the code moves into the loop the obtained answer is not what is expected. For that reason the latitude and longitude variables are assigned the incorrect values due to a change of position in the string. A fast solution to this problem was to implement a line in the GPSPosition function that only considers the answer that was expected initially. Hence, the code is published each 50 seconds. A further improved solution to this problem is to implement a line that acquires both responses that are obtained by the module and assign the correct latitude and longitude values if response A or B is obtained. The code of the Arduino can also be improved overall by optimizing the code memory-wise and remove the Serial.print lines which are only necessary for debugging and not for the actual production code.

Developing the GPS tracker further different hardware is needed. The current prototype is big in size and non discrete and such a device needs to be discrete to prevent it from being damaged by others. The power source of the current tracking device should also be reconsidered. The module was powered on and the code was run for an hour. An hour later the voltage of the 9V battery was dropped to 7.89V and the module did not receive enough power. A GPS module is high in energy consumption which was not taken into account when the code was developed. In the code setup, the GPRS and GPS functions are powered

on and never powered off. A suggested improvement is to power on the GPS and GPRS functions, retrieve a GPS position, publish it to the broker and later power down the GPS and GPRS functions. A disadvantage to this method is the accuracy of the GPS position. As GPS is powered on, a few minutes are necessary for the setup of the GPS. Hence, the published messages will contain zeros or an inaccurate position.

Appendices

A.1 Arduino code

```
// Select modem:
#define TINY_GSM_MODEM_SIM808

//Set serial debug monitor
#define SerialMon Serial

// Set serial for AT commands
#ifndef __AVR_ATmega328P__
#define SerialAT Serial1

// or Software Serial on Uno, Nano
#else
#include <SoftwareSerial.h>
SoftwareSerial SerialAT(2, 3); // RX, TX
#endif

#define TINY_GSM_DEBUG SerialMon

//  imported libraries
#include <TinyGsmClient.h>
#include <ArduinoJson.h>
#include <PubSubClient.h>

/************************* Adafruit.io Setup *********************************/

#define AIO_SERVER    "io.adafruit.com"
#define AIO_SERVERPORT 1883
#define IO_USERNAME "IvetSJ96"
#define IO_KEY     "PRIVATE IO KEY"
#define IO_TOPIC "IvetSJ96/feeds/object_tracker"

/*****************************INITIALIZE************************************/
const char apn[] = "internet.tele2.se";

// Initiate libraries
TinyGsm modem(SerialAT);
TinyGsmClient client(modem);
PubSubClient mqtt(client);

//  Global variables
```

```
String content;
String response;
String Latitude;
String Longitude;
uint32_t lastReconnectAttempt = 0;

// Send command function. The command is sent and set for a duration of time.
// Incoming chars are read, concated and stored in content variable.
String command(String Command, unsigned long milliseconds) {
  String content = "";
  SerialMon.print(F("Sending: "));
  SerialMon.println(Command);
  SerialAT.println(Command);
  unsigned long startTime = millis();
  SerialMon.print(F("Received: "));
  while (millis() - startTime < milliseconds) {
    if (SerialAT.available()) {
      char c = SerialAT.read();
      content.concat(c);
    }
  }
  return content;
}

/*****************************GPS FUNCTIONS*******************************/

// Retrieve GPS position and assign Latitude and Longitude values in string
   format.
// GPS NMEA format returned.
String GPSPosition() {
  SerialMon.println(F("Getting GPS position..."));
  String ans = command(F("AT+CGNSINF"), 5000);
  if (ans.length() > 0) {
    SerialMon.println(ans + '\n');
    SerialMon.println(F("Got location\n"));
    Latitude = ans.substring(48, 57);
    SerialMon.print(F("Latitude:"));
    SerialMon.println(Latitude + '\n');
    Longitude = ans.substring(58, 67);
    SerialMon.print(F("Longitude:"));
    Serial.println(Longitude + '\n');
    return ans;
  }
  else {
    SerialMon.println(F("Could not get position"));
    return ans;
  }
}
```

```
// Set accurate buad rate. If "ERROR" is received, resend command.
void SetBuadRate() {
  String ans = command(F("AT"), 1000);
  while (ans == F("ERROR")) {
    command(F("AT"), 1000);
  }
  SerialMon.println(ans + '\n');
  ans = command(F("AT+IPR=38400"), 1000);
  SerialMon.println(ans + '\n');
}

// Power on GPS module.
void GPSOn() {
  String ans = command(F("AT+CGNSPWR=1"), 5000);
  SerialMon.println(ans + '\n');
}

/************************GPRS FUNCTIONS*****************************/

// Connect to network.
void GPRSInit() {
  SerialMon.print(F("Waiting for network..."));
  if (!modem.waitForNetwork()) {
    SerialMon.println(F(" fail"));
    while (true);
  }
  SerialMon.println(F(" OK"));

  SerialMon.print(F("Connecting to "));
  SerialMon.print(F("internet.tele2.se"));
  if (!modem.gprsConnect("internet.tele2.se")) {
    SerialMon.println(F(" fail"));
    while (true);
  }
  SerialMon.println(F(" OK"));
}

/*************************MQTT FUNCTIONS***************************/

// Prepare MQTT message in JSON format
String buildJson() {
  StaticJsonDocument<16> doc;

  doc["latitude"] = Latitude.toDouble();
  doc["longitude"] = Longitude.toDouble();

  String output;
  serializeJson(doc, output);
  return output;
```

```
}

// Connect to MQTT broker, subscribe to topic and publish message in JSON
    format.
// If not connected, reconnect. If failed to connect, show error code.
void reconnect() {
  while (!mqtt.connected()) {
    Serial.print(F("Attempting MQTT connection..."));
    if (mqtt.connect("", IO_USERNAME, IO_KEY)) {
      Serial.println(F("connected"));
      mqtt.publish(IO_TOPIC, (char*) buildJson().c_str());
      mqtt.subscribe(IO_TOPIC);
    } else {
      Serial.print(F("failed, rc="));
      Serial.print(mqtt.state());
      Serial.println(F(" try again in 5 seconds"));
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}

/**********************START PROGRAM****************************/
void setup() {
  // Set console baud rate
  SerialMon.begin(38400);
  SerialAT.begin(38400);
  delay(6000);
  SetBuadRate();
  delay(3000);
  GPRSInit();
  delay(3000);
  GPSOn();
  delay(3000);

  //MQTT setup
  mqtt.setServer(AIO_SERVER, AIO_SERVERPORT);
}

// GPS position retrieved every 20 seconds.
void loop() {
  if (GPSPosition().length() > 0) {
    SerialMon.print(Latitude + ":" + Longitude);
    mqtt.loop();
    if (!mqtt.connected()) {
      reconnect();
    }
  }
  delay(20000);
```

```
}
```

A.2 Object Tracker App code

```
//The main program where the code is executed. A widget is built with the
//intro screen as a home screen.

//file name: main.dart

import 'package:flutter/material.dart';
import 'screens/intro_screen.dart';

void main() {
  runApp(const GlobeApp());
}

class GlobeApp extends StatelessWidget {
  const GlobeApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      theme: ThemeData(primaryColor: Colors.white70),
      home: const IntroScreen(),
    );
  }
}
```

```dart
//The intro screen code is built here. Here the Google map widget is created
//and a marker is set for a change of position.

//file name: intro_screen.dart

import 'package:flutter/material.dart';
import 'package:google_maps_flutter/google_maps_flutter.dart';
import '../MQTT/mqtt_wrapper.dart';
import '../shared/menu_drawer.dart';
import 'package:location/location.dart';

class IntroScreen extends StatefulWidget {
  const IntroScreen({Key? key}) : super(key: key);

  @override
  _IntroScreenState createState() => _IntroScreenState();
}

class _IntroScreenState extends State<IntroScreen> {
  String topicName = 'IvetSJ96/feeds/object_tracker';
  late MqttClientWrapper mqttClientWrapper;
  LocationData? currentLocation;
  late GoogleMapController mapController;
  Map<MarkerId, Marker> markers = <MarkerId, Marker>{};
  final LatLng _center = const LatLng(59.85882, 17.63889);

  //Setting up the MQTT connection and listening for new locations.
  void setup() {
    mqttClientWrapper = MqttClientWrapper(
        (newLocationJson) => gotNewLocation(newLocationJson!));
    mqttClientWrapper.prepareMqttClient();
  }

  //If a new location is obtained, navigate the user to the new location by
     calling the function animateCameraToNewLocation.
  void gotNewLocation(LocationData newLocationData) {
    setState(() {
      currentLocation = newLocationData;
    });
    animateCameraToNewLocation(newLocationData);
  }

  //Connecting to the MQTT broker
  void connect() {
    mqttClientWrapper.prepareMqttClient();
  }

  //A function that navigates the user to the new location by setting the
```

```
    correct latitude and longitude values.
void animateCameraToNewLocation(LocationData newLocation) {
  mapController.animateCamera(CameraUpdate.newCameraPosition(CameraPosition(
      target: LatLng(newLocation.latitude!, newLocation.longitude!),
      zoom: 16.0)));
}

@override
void initState() {
  super.initState();

  setup();
}

@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        actions: <Widget>[
          IconButton(
            icon: const Icon(
              Icons.update,
              color: Colors.green,
            ),
            onPressed: () {
              connect(); //if the update button is pressed, connect to the
                  broker
            },
          )
        ],
        title: const Text('Object Tracker'),
        backgroundColor: Colors.white,
        foregroundColor: Colors.grey,
      ),
      drawer: const MenuDrawer(),
      body: GoogleMap(
          initialCameraPosition: CameraPosition(
            target: _center,
            zoom: 16.0,
          ),
          markers: currentLocation == null
              ? Set()
              : {
                  Marker(
                      markerId: const MarkerId('Vehicle Position'),
                      infoWindow:
                          const InfoWindow(title: 'Your vehicle is here!'),
                      position: LatLng(currentLocation!.latitude!,
```

```
                        currentLocation!.longitude!))
                }.toSet(),
            onMapCreated: (GoogleMapController mapController) {
              setState(() {
                this.mapController = mapController;
              });
            }),
      ),
    );
  }
}
```

```dart
//An MQTT client is connected and subscribed to the broker and listens for
//incoming messages.

//file name: mqtt_wrapper.dart

import 'dart:developer';
import 'package:mqtt_client/mqtt_client.dart';
import 'package:mqtt_client/mqtt_server_client.dart';
import 'package:object_tracker/MQTT/Adafruit_feed.dart';
import 'package:object_tracker/MQTT/models.dart';
import 'package:location/location.dart';
import '../MQTT/converter.dart';

class MqttClientWrapper {
  String username = 'IvetSJ96';
  String password = 'PRIVATE IO KEY';
  String server = 'io.adafruit.com';
  String topic = 'IvetSJ96/feeds/object_tracker';
  String clientIdentifier = 'object_tracker_app';

  late MqttClient client;
  var connectionState = MqttClientConnectionState.Idle;
  var subscriptionStatus = MqttClientSubcriptionState.Idle;

  final Function(LocationData?) onDataReceivedCallback;

  JsonToLocationConverter jsonToLocationConverter = JsonToLocationConverter();

  MqttClientWrapper(this.onDataReceivedCallback);

  void prepareMqttClient() async {
    _setupMqttClient();
    await _connectClient();
    subscribeToTopic();
  }

  //Subscribe to topic and listen for incoming messages
  Future subscribeToTopic() async {
    print('[MQTT client] Subscribing to the $topic topic');
    client.subscribe(topic, MqttQos.atMostOnce);

    client.updates!.listen((List<MqttReceivedMessage<MqttMessage>> c) {
      final receivedMsg = c[0].payload as MqttPublishMessage;
      final String data =
          MqttPublishPayload.bytesToStringAsString(receivedMsg.payload.message);

      AdafruitFeed.add(data);
      log('[MQTT Client] Got a message $data');
```

```
    LocationData? newLocationData = _convertJsonToLocation(data);
    onDataReceivedCallback.call(newLocationData);
  });
}

//Set up the MQTT client with accurate authentication information
void _setupMqttClient() {
  final MqttConnectMessage connectMessage = MqttConnectMessage()
      .withClientIdentifier('object_tracker_app')
      .startClean()
      .authenticateAs(username, password)
      .withWillQos(MqttQos.atMostOnce)
      .withProtocolName("MQTT");

  client = MqttServerClient.withPort(server, clientIdentifier, 1883);
  client.logging(on: true);
  client.keepAlivePeriod = 1000;
  client.connectionMessage = connectMessage;
  client.onDisconnected = _onDisconnected;
  client.onConnected = _onConnected;
  client.onSubscribed = _onSubscribed;
}

//Connect the client
Future<MqttClientConnectionStatus?> _connectClient() async {
  MqttClientConnectionStatus? status;
  try {
    log('[MQTT Client] Adafruit client connectiing...');
    connectionState = MqttClientConnectionState.Connecting;
    status = await client.connect(username, password);
  } catch (e) {
    log('[MQTT Client] exception - $e');
    connectionState = MqttClientConnectionState.ErrorWhenConnecting;
    client.disconnect();
    throw Error();
  }

  if (client.connectionStatus!.state == MqttConnectionState.connected) {
    connectionState = MqttClientConnectionState.Connected;
    log('[MQTT client] client connected]');
  } else {
    log('[MQTT Client] Client ${client.connectionStatus}, disconnecting...');
    connectionState = MqttClientConnectionState.ErrorWhenConnecting;
    client.disconnect();
  }
  return status;
}

void _onSubscribed(String topic) {
```

```
    log('[MQTT Client] Subscription confirmed for topic $topic');
    subscriptionStatus = MqttClientSubcriptionState.Idle;
  }

  void _onDisconnected() {
    log('[MQTT Client] onDiscconected client callback - Client disconnected');
    connectionState = MqttClientConnectionState.Disconnected;
  }

  void _onConnected() {
    connectionState = MqttClientConnectionState.Connected;
    log('[MQTT Client] OnConnected client callback - Client connected!');
    subscribeToTopic();
  }

  //Convert the incoming messages from JSON format to LocationData format
  LocationData? _convertJsonToLocation(String newLocationJson) {
    try {
      return jsonToLocationConverter.convert(newLocationJson);
    } catch (exception) {
      print("Json can't be formatted ${exception.toString()}");
    }
    return null;
  }
}
```

```
//The menu is created here. Contains the options "Home", "Settings" and
    "Parking".
//The code is working but is incomplete.

//file name: menu_drawer.dart


import 'package:flutter/material.dart';
import '../screens/settings_screen.dart';
import '../screens/intro_screen.dart';

class MenuDrawer extends StatelessWidget {
  const MenuDrawer({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Drawer(
        child: ListView(
      children: buildMenuItems(context),
    ));
  }

  List<Widget> buildMenuItems(BuildContext context) {
    final List<String> menuTitles = ['Home', 'Settings', 'Parking'];
    List<Widget> menuItems = [];
    menuItems.add(const DrawerHeader(
      child: Text('Menu', style: TextStyle(color: Colors.grey, fontSize: 28)),
    ));
    menuTitles.forEach((String element) {
      Widget screen = Container();
      menuItems.add(ListTile(
        title: Text(element, style: const TextStyle(fontSize: 20)),
        onTap: () {
          switch (element) {
            case 'Home':
              screen = const IntroScreen();
              break;
            // case 'Settings':
            //   screen = SettingsScreen();
            //   break;
//            case 'Parking':
//             screen = ParkingScreen();
//              break;
          }
          Navigator.of(context).pop();
          Navigator.of(context)
              .push(MaterialPageRoute(builder: (context) => screen));
        },
```

```
      ));
    });
    return menuItems;
  }
}
```

```
//A class that takes in a JSON message, parses it and converts it to a Location-
//Data formatted message

//file name: converter.dart

import 'dart:convert';

import 'package:location/location.dart';

class JsonToLocationConverter {
  LocationData convert(String input) {
    Map<String, dynamic> jsonInput = jsonDecode(input);
    return LocationData.fromMap({
      'latitude': jsonInput["latitude"].toDouble(),
      'longitude': jsonInput["longitude"].toDouble(),
    });
  }
}
```

```
//The different connection states of the MQTT client.

//Code used from thuyhoang-hvtt GitHub repository:
//https://github.com/thuyhoang-hvtt/IoT-tracking/blob/master/lib/models/enums.dart

//file name: models.dart

enum ViewState { Idle, Busy }
enum Power { Off, On }
enum MqttClientConnectionState {
  Idle,
  Connecting,
  Connected,
  Disconnected,
  ErrorWhenConnecting,
}

enum MqttClientSubcriptionState { Idle, Subcripting }
```

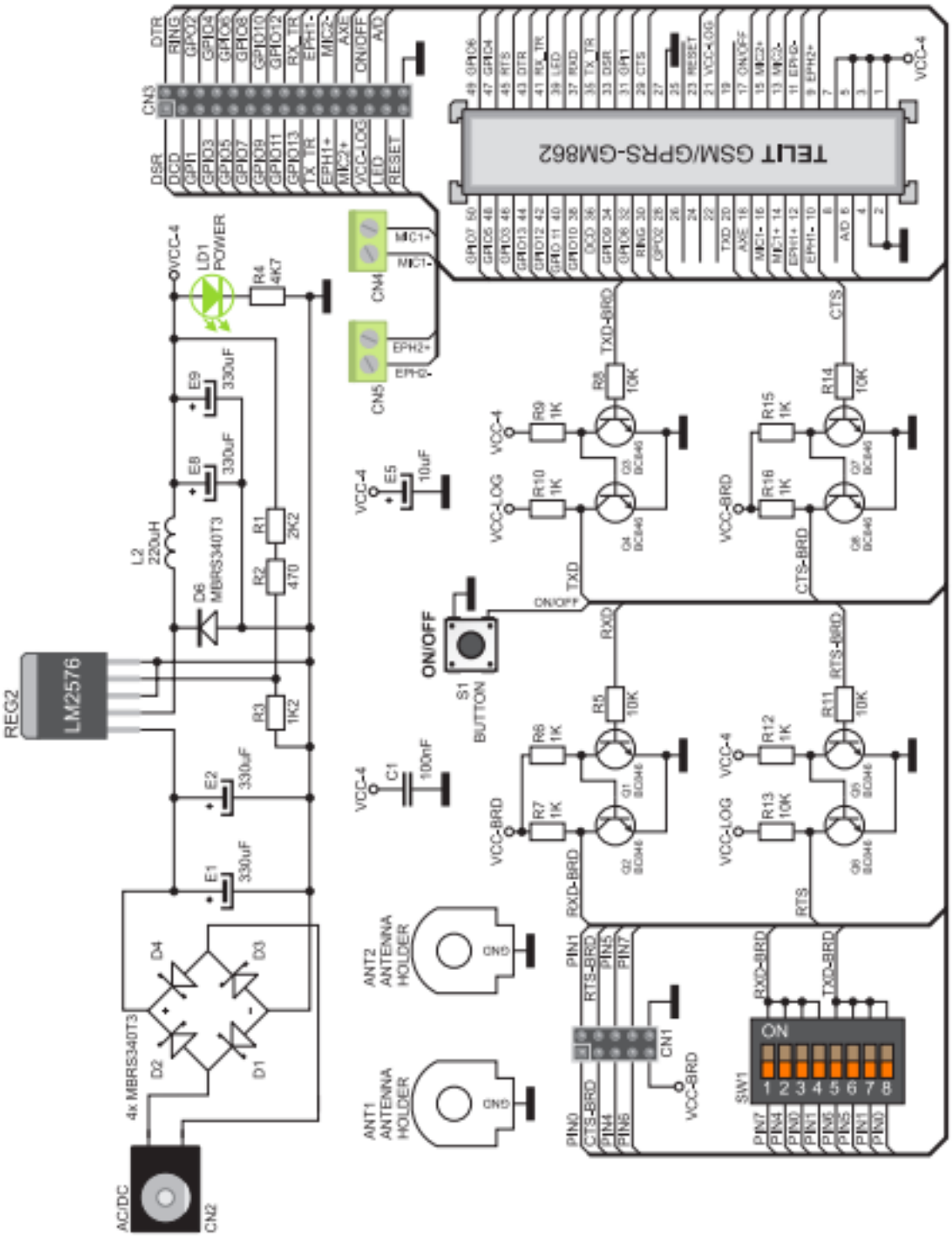
```dart
//code used from BitKnittings GitHub repository:
//https://github.com/BitKnitting/flutter_adafruit_mqtt/blob/master/lib
// /Adafruit_feed.dart

//file name: Adafruit_feed.dart

import 'dart:async';
import 'package:logging/logging.dart';

class AdafruitFeed {
  //
  // Both the StreamController and Stream are defined as static. This
  // means they both belong to the class and not to an instance.
  // It was my way of getting to what I was used to via Singleton
  // functionality in some other languages.
  //
  // A Stream controller alerts the stream when new data is available.
  // The controller should be private.
  static final _feedController = StreamController<String>();
  // Expose the stream so a StreamBuilder and use it.
  static Stream<String> get sensorStream => _feedController.stream;
  static void add(String value) {
    Logger log = Logger('Adafruit_feed.dart');
    try {
      _feedController.add(value);
      log.info('---> added value to the Stream... the value is: $value');
    } catch (e) {
      log.severe(
          '$value was published to the feed. Error adding to the Stream: $e');
    }
  }
}
```

A.3 SmartGM862 Circuit Diagram



A.4 Component List

To reproduce this project and achieve a complete GPS tracking device, the following components are essential for the hardware:

- Arduino Uno Rev3
- GSM / GPRS / GPS Arduino Shield (B) (for EU) by Waveshare
- 9V alkaline battery
- 9V battery contact / holder with contact
- GPS antenna, frequency 1575.42MHz and voltage 2.7 - 5V
- GSM antenna
- USB cable Type-A to Micro (for PuTTY debugging)
- USB 2.0 Cable Type A/B (for Arduino Uno)
- EU-standard adapter (Power source during development)
- SIM card adapter
- Registered SIM card
- 2 breadboard jumper wires
- Shrinkable tubing
- Optional: device case

References

[1] SVT, Fler cyklar stjäls i Uppsala – så ska polisen stoppa stölderna [Online]. Available: https://www.svt.se/nyheter/lokalt/uppsala/cykelstolderna-okar-i-uppsala [Accessed:2022-01-03]

[2] J. Eberspächer, H.Vögel, C.Bettstetter, C.Hartmann. GSM-Architecture, Protocols and Services. 3rd ed. Chichester, U.K.: Wiley; 2009,, U.K.: Wiley; 2009, pp. 43-55

[3] J. Eberspächer, H.Vögel, C.Bettstetter, C.Hartmann. GSM-Architecture, Protocols and Services. 3rd ed. Chichester, U.K.: Wiley; 2009, pp. 235-236

[4] R. Langley, Dilution of Precision. University of New Brunswick, 1999.

[5] Electrokit, GM862 Family Hardware User Guide [Online]. Available: https://www.electrokit.com/uploads/productfile/41017/GM862-GPS_Hardware_User_Guide.pdf [Accessed: 2022-01-17]

[6] Yumpu, Telit GM862-GPS Software User Guide [Online]. Available: https://www.yumpu.com/en/document/read/49729568/gm862-gps-software-user-guide-semiconductorstorecom [Accessed: 2022-02-04]

[7] Flutter Documentation, Installation of Flutter on MacOS [Online]. Available: https://docs.flutter.dev/get-started/install/macos [Accessed: 2022-02-13]

[8] Mikroe, SmartGM862 development system user manual [Online]. Available: https://download.mikroe.com/documents/add-on-boards/other/wireless-connectivity/smartgm862/smartgm862-manual-v100.pdf [Accessed: 2022-02-15]

[9] Federal Aviation Administration, Satellite Navigation - Global Positioning System (GPS) [Online]. Available: https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gnss/gps [Accessed:2022-02-25]

[10] Federal Aviation Administration, Satellite Navigation - GPS - How It Works [Online]. Available: https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gnss/gps/howitworks [Accessed: 2022-03-04]

[11] Arduino, Arduino Uno Rev3 [Online]. Available: https://store.arduino.cc/products/arduino-Uno-rev3 [Accessed: 2022-03-04]

[12] Waveshare, GSM/GPRS/GPS Shield (B), Arduino Shield Based on SIM808, How To Use [Online]. Available: https://www.waveshare.com/wiki/GSM/GPRS/GPS_Shield_(B) [Accessed: 2022-04-04]

[13] TinyGSM library for Arduino - GitHub Repository [Online]. Available: https://github.com/vshymanskyy/TinyGSM [Accessed: 2022-04-15]

[14] ArduinoJson library for Arduino - GitHub Repository [Online]. Available: https://github.com/bblanchon/ArduinoJson [Accessed: 2022-04-15]

[15] PubSubClient library for Arduino - GitHub Repository [Online]. Available: https://github.com/knolleary/pubsubclient [Accessed: 2022-04-15]

[16] JSON, Introducing JSON [Online]. Available: https://www.json.org/json-en.html [Accessed: 2022-04-26]

[17] Wikipedia, PuTTY emulator [Online]. Available: https://en.wikipedia.org/wiki/PuTTY [Accessed: 2022-04-26]

[18] HiveMQ, Introducing the MQTT Protocol [Online]. Available: https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/ [Accessed: 2022-04-27]

[19] HiveMQ, Publish & Subscribe - MQTT Essentials. [Online] Available: https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/ [Accessed: 2022-04-28]

[20] R.Vahidnia, F.Dian . Cellular Internet of Things for practitioners. British Columbia Institute of Technology; 2021. Chapter 4.2.

[21] Adafruit, Adafruit IO MQTT API [Online]. Available: https://io.adafruit.com/api/docs/mqtt.html#adafruit-io-mqtt-api [Accessed: 2022-05-01]

[22] Flutter Documentation, Flutter material.dart package [Online]. Available: https://pub.dev/packages/material [Accessed: 2022-05-04]

[23] Flutter Documentation, Flutter location.dart package [Online]. Available: https://pub.dev/packages/location [Accessed: 2022-05-04]

[24] Flutter Documentation, Flutter mqtt_client.dart package [Online]. Available: https://pub.dev/packages/mqtt_client [Accessed: 2022-05-04]

[25] Flutter Documentation, Flutter convert.dart package [Online]. Available: https://pub.dev/packages/convert [Accessed: 2022-05-04]

[26] Flutter Documentation, Flutter google_maps_flutter.dart package. [Online]. Available: https://pub.dev/packages/google_maps_flutter [Accessed: 2022-05-04]

[27] GitHub repository - IoT Tracking Flutter project [Online]. Available: https://github.com/thuyhoang-hvtt/IoT-tracking [Accessed: 2022-05-04]

[28] GitHub repository - How to publish to an Adafruit broker [Online]. Available: https://github.com/BitKnitting/flutter_adafruit_mqtt [Accessed: 2022-05-04]

[29] Google Codelabs, Adding Google Maps to a Flutter app [Online]. Available: https://codelabs.developers.google.com/codelabs/google-maps-in-flutter#0 [Accessed: 2022-05-05]

[30] Medium, Using MQTT With Flutter to Build a Location Sharing App [Online]. Available: https://medium.com/swlh/using-mqtt-with-flutter-to-build-a-location-sharing-app-24e7307b21d3 [Accessed: 2022-05-05]