MCO015-340112

Data Acquisition Technologies and Sensor Networks
Fall 2022

Prof. Ph.D. Fangning Hu

Group Project Work

Group Members

1	ABHISHEK MATHUR	30006043
2	MOHAMMAD TANZIL ALAM	30006050
3	CHANDRA SHEKAR SRINIVASAIAH	20332555

Contents

Introduction

As new technology improvements emerge, users prefer automobile systems to manually operated ones. Accidents and thefts have increased along with the number of car users. The use of the Internet in networking has expanded because of the convergence of several technologies, enabling remote sensing and control of things.

The vehicle is tracked, monitored, and controlled via an internet-based embedded systems using a mobile or computing device. It can offer intercity transportation vehicles a telemonitoring system. The vehicular module locates the accident site, investigates, and notifies the monitoring station. The suggested design offers real-time information on the identity, speed, and location of the vehicle.

Our idea is to stored specific event in a program or a request from a monitoring station, the vehicular system transmits information about a vehicle, such as position through a GPS module and identity of a vehicle, to a monitoring station and to a mobile phone. The monitoring station, which may be installed in freight trucks, vehicles, motorcycles, and boats, displays this information on a web page for tracking. The system has several possible uses.

With the help of the result obtained from the GPS module which is getting updated in every 15 seconds we can dertermine the excat and also the route in which the vehicle travelled. These information togeather will help us to solve the problem of theft detection and vehicle tracking system.

Based on this basic framework, the product that runs multiple iterations through numerous cycles using design thinking techniques must be designed. An IoT-based GNSS/GPS tracker and a web application that shows specific details on delivery van routes and delivery Hotspots were our first products. The product's objective is to give city planners useful information so they may better design and manage curb space. The suggested design offers real-time information on the identity, speed, and location of the vehicle. The RASPBERRY PI compiles this data utilizing various sections before sending it to the observation station, where it is stored in a database and presented in a user-friendly graphical user interface (GUI). [https://ijisrt.com/wp-content/uploads/2018/01/Real-Time-Vehicular-Tracing-System-using-Raspberry-PI-1.pdf Real Time Vehicular Tracing System using RASPBERRY PI, Volume 3, Issue 1, January – 2018, International Journal of Innovative Science and Research Technology ISSN No:-2456 –2165]

Background and approach

Tracking and positioning play a major role in efficient management and monitoring of the vehicles in modern day. The technology put in use for this purpose uses Global Positioning System to streamline the operations by providing real-time updates, vehicle detection and maintenance, and locating in case of theft.

The Raspberry Pi is a small, low-cost computer that can be used for a variety of projects, including tracking with GPS. GPS stands for Global Positioning System, and it is a satellite-based navigation system that allows users to determine their precise location, speed, and direction.

To use GPS with a Raspberry Pi, we will need to connect a GPS receiver to the Raspberry Pi. There are several options for GPS receivers that are compatible with the Raspberry Pi, including USB-based receivers and receivers that communicate via the serial port. Once our idea of using GPS is fixed, we can connect the GPS receiver to the Raspberry Pi and we will need to install software to allow the Raspberry Pi to communicate with the GPS receiver and process the GPS data.

Once the hardware and software are set up, we can use the GPS data from the Raspberry Pi to track the location of the connected Raspberry Pi, or to use the Raspberry Pi as a GPS tracking device for other objects or people. For example, one could use a Raspberry Pi with GPS to track the location of a vehicle, or to track the location of a person or animal using a GPS-enabled device.

Overall, the Raspberry Pi can be a useful tool for tracking with GPS, and it offers a flexible and low-cost platform for building GPS-based tracking systems. Let us examine how raspberry pi and GNSS 4 click are coupled and made to work.

GNSS 4 Click module

GNSS 4 click module we make use of, carries SAM-M8Q patch antenna module from u-blox. The click is designed to run on a 3.3V power supply and communicates with the target microcontroller over I2C or UART interface. Let us understand and learn more about the GNSS module, its antennas, power management and its working in various environments.

What is GNSS?

GNSS stands for Global Navigation Satellite System, an umbrella term that describes both the United States GPS, the Russian GLONASS global positioning systems and European Galileo and is based on a group of 24 satellites and their ground stations developed by U.S. Department of Defence, for which satellite sends data to its receiver in a form of signal which then computes its position and time.

SAM-M8Q module

The SAM-M8Q module utilizes concurrent reception of up to three GNSS systems (GPS/Galileo and GLONASS), recognizes multiple constellations simultaneously and provides outstanding positioning accuracy in scenarios where urban canyon or weak signals are involved.

Patch antenna

The GNSS patch antenna is RHCP (right hand circular polarization) and has a peak gain of 3 dBic. The patch antenna is insensitive to surroundings and has high tolerance against frequency shifts.

Power management

u-blox M8 technology offers a power-optimized architecture with built-in autonomous power saving functions to minimize power consumption at any given time. Furthermore, the receiver can be used in two operating modes: Continuous mode for best performance or Power Save Mode for optimized power consumption.

Good in hostile environments

Thanks to all these features the SAM-M8Q module is good in GNSS-hostile environments - indoor spaces, urban canyons (when a street is flanked by buildings on both sides), etc.

How it works

A constellation of satellites sends a continuous signal towards Earth. Onboard every satellite is an atomic clock, and all of them are synchronized, thanks to a reference time scale defined by the whole system. So, that the signals coming from the different satellites of the same constellation share the same reference time scale.

If the user wants to utilize GNSS to determine a position, they must have an antenna that receives the signals coming from the satellites, and a receiver that translates these signals. The antenna position will be deduced from the measurements of the time delay between the emission time (satellite) and the reception time (receiver) for at least 4 signals coming from different satellites. Specifications and Pin-Out diagrams for GNSS 4 click are shown below:

Type	GPS/GNSS
Applications	Asset tracking, for navigation devices based on GPS and GLONASS, road navigation devices, public transport, wearable devices, etc.
On-board modules	SAM-M8Q from u-blox
Key Features	High accuracy, I2C and UART interface, 3.3V power supply
Interface	GPIO,I2C,UART
Compatibility	mikroBUS
Click board size	M (42.9 x 25.4 mm)
Input Voltage	3.3V

Fig 1: Specifications for GNSS 4 click

Notes	Pin	mikro BUS				Pin	Notes
	NC	1	AN	PWM	16	NC	
Active Low	**RST_N**	2	RST	INT	15	NC	
	NC	3	CS	TX	14	**TXD**	UART transmit
	NC	4	SCK	RX	13	**RXD**	UART receive
	NC	5	MISO	SCL	12	**SCL**	I2C Clock
	NC	6	MOSI	SDA	11	**SDA**	I2C Data
Power supply	**+3.3V**	7	3.3V	5V	10	NC	
Ground	**GND**	8	GND	GND	9	**GND**	Ground

Fig 2: Pin-out diagram for GNSS 4 click. We only make use of pin numbered 7 (+3.3V), 8 (GND), 13 (TXD UART transmit) and 14 (RXD UART receive)

NMEA (National Marine Electronics Association) signals:

GNSS module inputs the NMEA signals to raspberry pi via microbus pins. In order to understand NMEA signals and decoding them, let us look at the formats and try to analyse one of the NMEA sentence that we put to use in this project:

• $GNGSA - GNSS DOP and active satellite

• $GNGLL - Geographic Position, Latitude/Longitude

• $GNGGA - Global Positioning System Fix Data (we currently use this for our purpose)

• $GNRMC - Recommended minimum speci c GPS/Transit data

These data are encoded and has very useful information, which when on parsing these stream of sentences will shell out information like latitude, longitude, speed, altitude, timestamp, etc.

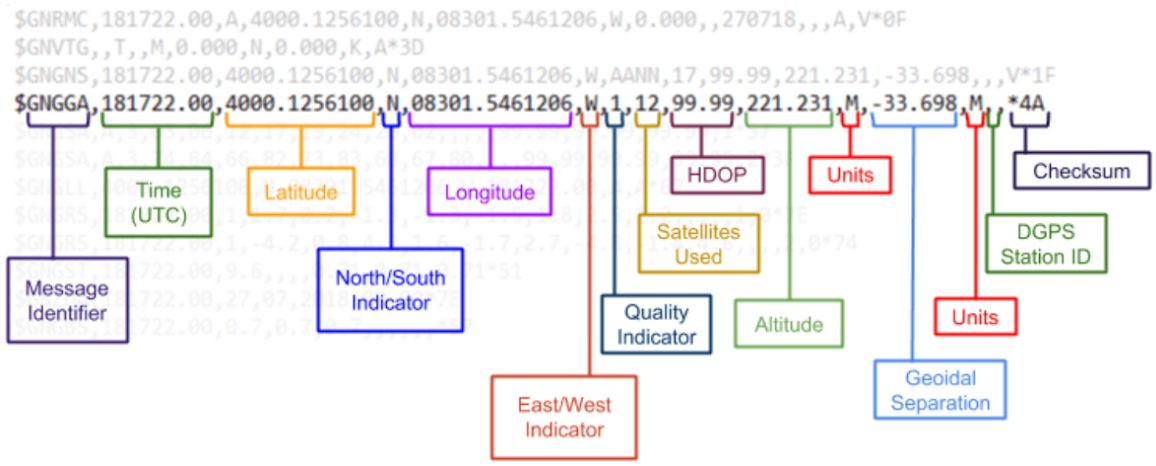


Fig 3: Decoding GNGGA data

The following table describes the functionality it represents:

Time	Universal Time Coordinated (UTC) in hhmmss.ss.
Latitude	Latitude in decimal degrees (ddmm.mmmm).
North/South Indicator	North or south of the Equator.
Longitude	Longitude in decimal degrees (ddmm.mmmm).
East/West Indicator	East or west of the Prime Meridian.
Quality Indicator	The quality of the GNSS data — what kind of fix the receiver has obtained. 0 — No fix 1 — Standard GPS (2D/3D) fix 2 — Differential GPS (DGPS) fix 3 — Precise Positioning System (PPS) fix 4 — RTK fixed solution 5 — RTK float solution 6 — Estimated (Dead Reckoning, or DR) fix
Satellites Used	Number of satellites used in solution.
Altitude	Altitude above mean sea level.
Units	Units used for altitude or geoidal separation (meters)
Geoidal Separation	Difference between the reference ellipsoid and mean sea level
DGPS Station ID	ID of the differential reference station used for DGPS. Blank when DGPS is not used.
Checksum	Used for data validation.

Source: [https://greenvillagedotblog.files.wordpress.com/2018/08/u-blox8-m8_receiverdescrprotspec_ubx-13003221_public.pdf, https://brandidowns.com/?p=77]

Usage of Raspberrypi Zero W with GNSS module

We must configure the Raspberry Pi's OS to be able to communicate with the GPS receiver, so we use a fresh install of the operating system to rule out any configuration problems when you first test your GPS module. To communicate with the GNSS signals via raspberry pi, we need communication to send or receive commands, we use SSH for our purpose

SSH on Raspberry Pi: The "**S**ecure **Sh**ell" (SSH) and is one of the most useful ways for remotely managing a device such as the Raspberry Pi. This is useful to interact with the Raspberry Pi's command line without having to have a keyboard, mouse or screen connected to it. This is what makes the project be "interactive" from remote locations. To enable SSH from Raspberry Pi, the easiest way to enable SSH without a GUI is to make use of the raspi-config tool. To open the raspi-config tool, we run the following command. sudo raspi-config. Within the tool we use the arrow keys on our keyboard to select the Interfacing Options and is shown in next part.

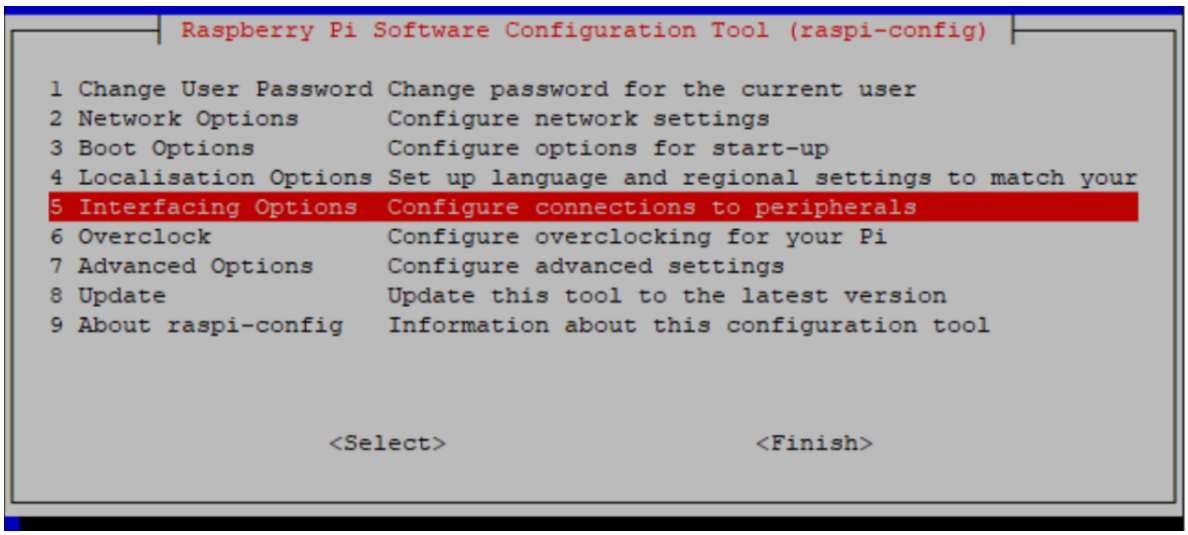
Connecting the GPS Receiver and Raspberry Pi

We use a simple serial connection, so the GNSS modules RX and TX pins must be connected to the Raspberry Pi's TX and RX pins respectively. The voltage configure is 3.3V and the ground is ensured before switching the circuit on.

Configuring the Raspberry Pi for GPS

We start by running raspi-config:

sudo raspi-config, followed by selecting the interfacing options and selecting the serial option to enable it, as shown below:



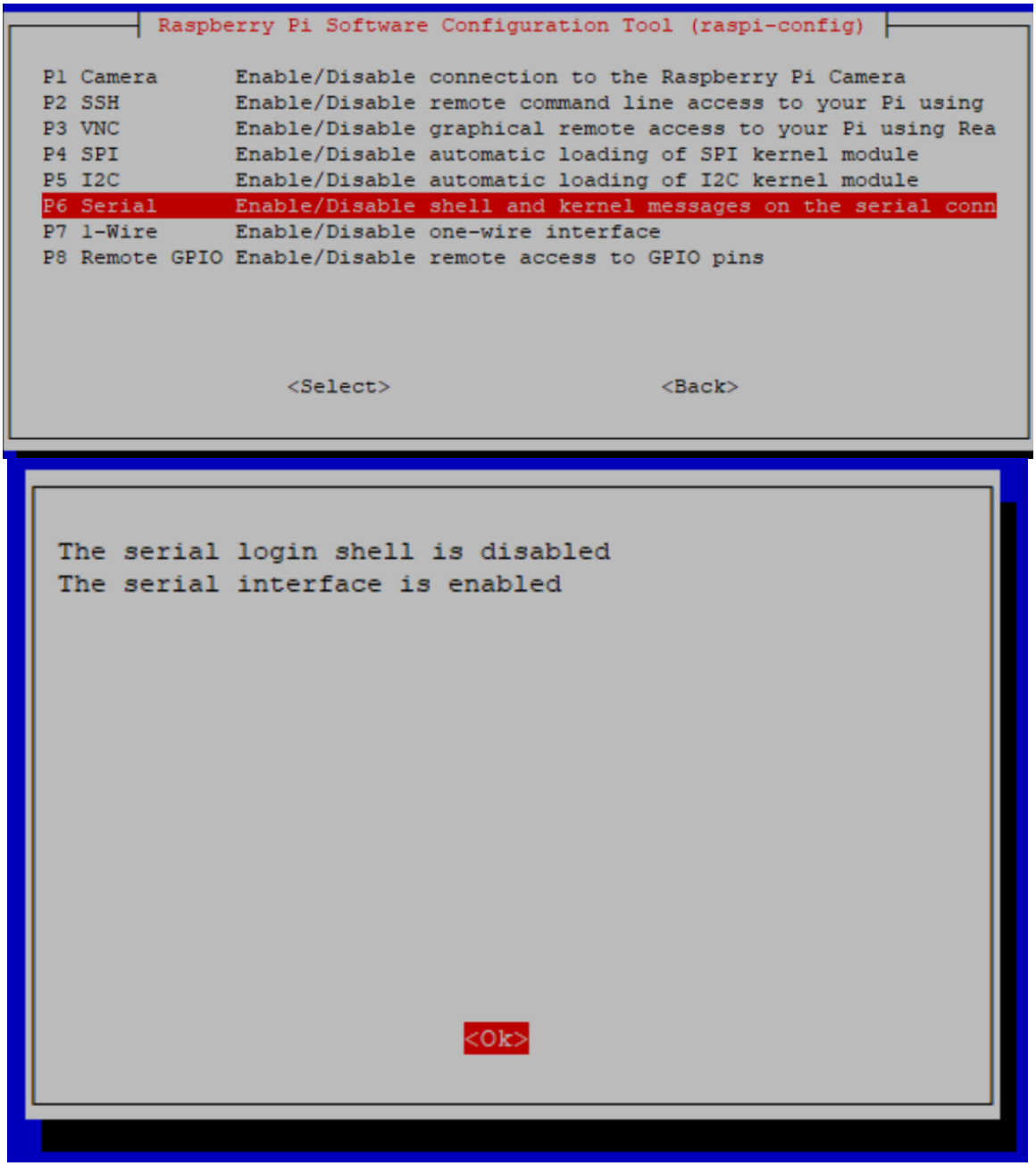


Fig 4 (a,b,c): Enabling serial communication via interfacing options

The communication is done via the software called MOBAXTERM to communicate to raspberry pi (PUTTY can be used as well). We then download the necessary software to decode the signals received by GNSS **receiver by i**nstalling "gpsd" and the "gpsd-client" python library:

sudo apt-get install gpsd gpsd-clients

gpsd is an interfacing daemon for serial GPS receivers which supports different communication standards. We try to use it to get a test reading and verify that the hardware works correctly.

Once the installation is done, we verify that we can receive data from the GPS module. To do that, we execute the command below to get the output data that it sends over the serial port:

cat /dev/serial0

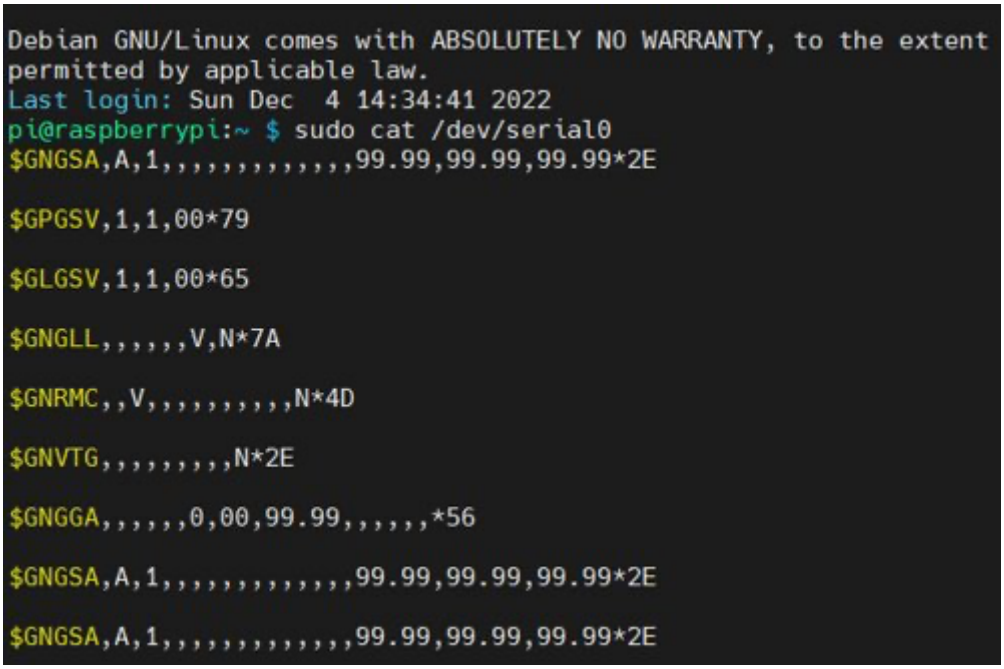


Fig 5: Output of serial communication via port 0

It doesn't matter what data we receive at this point if we receive something. If the port closes immediately or the Pi doesn't receive any data at all, there will be no output received correctly.

Now it's finally time to determine the position of the Raspberry Pi, right? We type the following command to stop the gpsd service that got started automatically when it was installed earlier. we must do this because the default options aren't properly configured for the raspberry pi:

sudo systemctl stop gpsd.socket

We then start a new gpsd instance that redirects the data of the correct serial port to a socket:

sudo gpsd /dev/serial0 -F /var/run/gpsd.sock

and run sudo gpsmon, to display the GPS data.

Note: It might take up to 30 minutes until the module can determine your position during the first boot if the device is indoor.

Once the setup is done and completed via software and necessary libraries installed, we then store the decoded information to the database and the database used here is SQLite3. This is the inbuilt database for python, and it is easy to operate the data in real-time.

Data Processing

To setup the database, we provide the database name and the table for it. The columns the table has contains: 1) id, 2) date, 3) latitude and 4) longitude (see Fig 14).

Initially all the data collected by us in the first attempt was stored in csv file which was then transferred to SQLite database. The following snapshot shows the usage, and the points are used to pinpoint the location via google maps API:

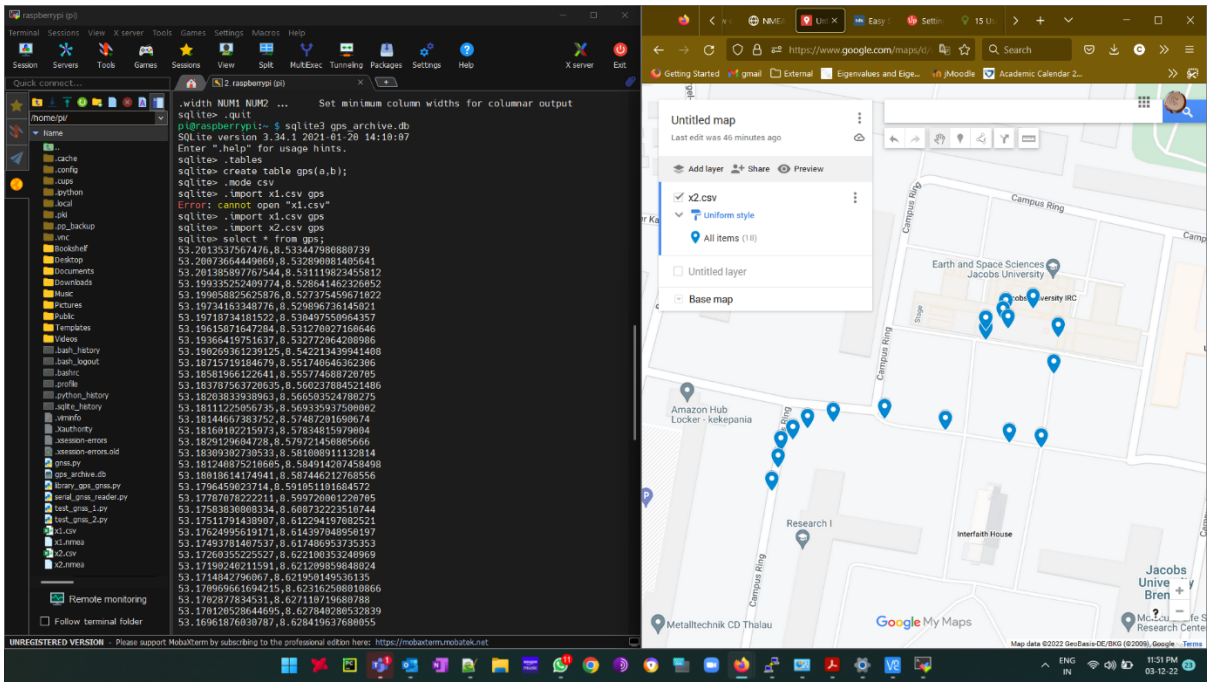


Fig 6: Storing the decoded data into sqlite database and using that data to plot on google map

We can also store the NMEA data in .nmea files and then use the website to load the NMEA file to pin point.

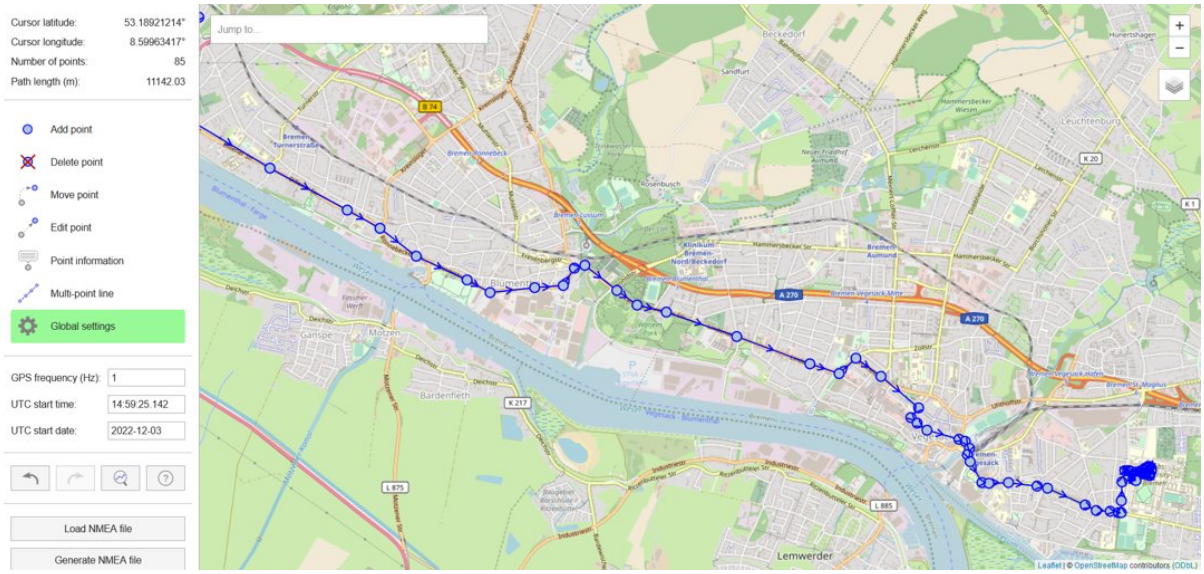


Fig 7: Using NMEA file generated to plot the points on the map (Blumenthal-Vegesack)

The data is collected and stored in the memory for every 15 seconds. This means every point the map shows is captured at the interval of 15 seconds as seen below. The following NMEA were generated on the day of demonstration, inside the University.

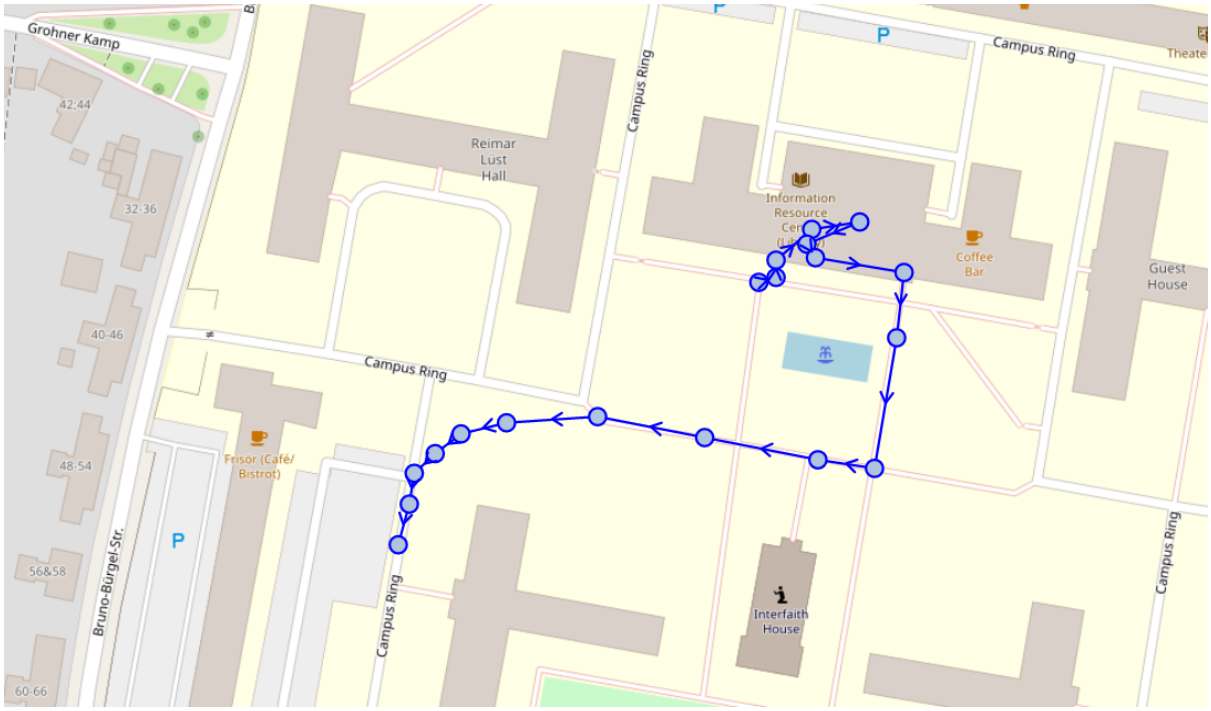


Fig 8: Using NMEA file generated to plot the points on the map

To demonstrate the presentable version of our project, we used the robo vehicle fitted with LEDs when turned on, and are connected to the raspberry pi for powering it up. The LEDs are white when they are idle (without running the program). Once the program starts running, the LEDs turn blue and the python program collected the location every 15 seconds.

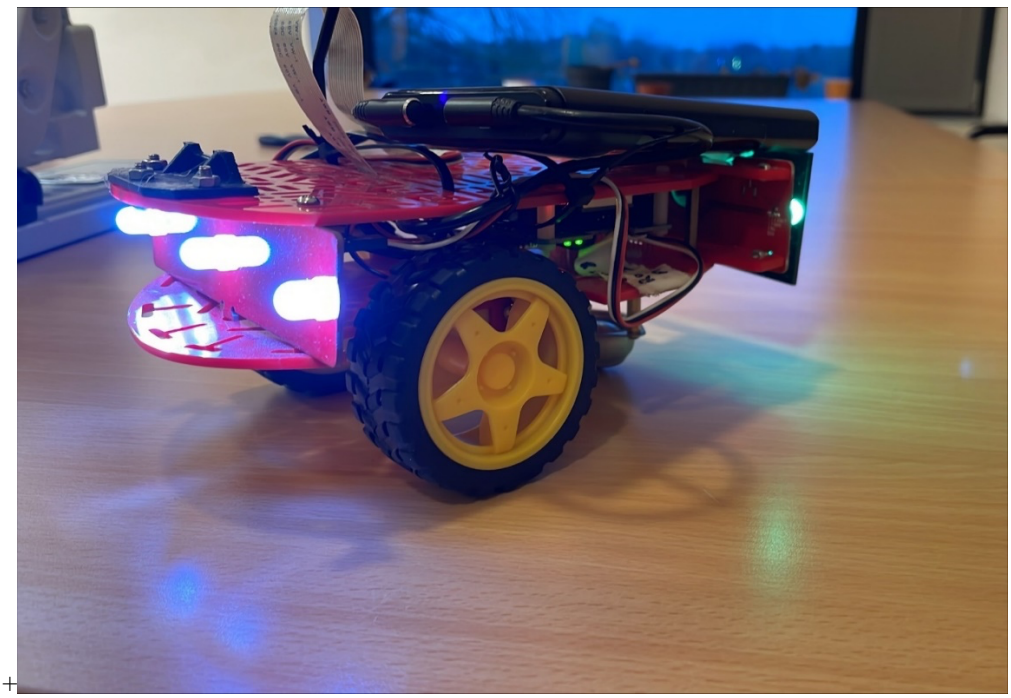


Fig 9: The model where the GNSS module connected to raspberrypi is placed, powered by 20000mAH powerbank with LEDs indicating the device is on and trackable.

We not only just use the SSH to communicate, but also made use of 16*2 LCD display module to show the real-time location as shown below:

(Current location shown in the figure is Constructor University Bremen)

Fig 10: The LED component (16*2) display used as additional display device on the model to determine the location coordinates (latitude, followed by longitude).

To ensure the device works properly at scheduled time, say every morning at 9 or when the vehicle starts moving for example, we make use of CRONjob in python. We tried using the cron as shown below:

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
* * * * * ./erial_gnss_reader.py
~
~
```

Fig 11: cron job for python, used for scheduled tracking

```
pi@raspberrypi:~ $ sqlite3 gps_receiver.db
SQLite version 3.34.1 2021-01-20 14:10:07
Enter ".help" for usage hints.
sqlite> select * from positiondata_06122022;
id|date|latitude|longitude
0|06-12-22|53.20135376|8.533447981
1|06-12-22|53.20073664|8.532890081
2|06-12-22|53.2013859|8.531119823
3|06-12-22|53.19933525|8.528641462
4|06-12-22|53.19905883|8.52737546
5|06-12-22|53.19734163|8.529896736
6|06-12-22|53.19718734|8.530497551
7|06-12-22|53.19615872|8.531270027
8|06-12-22|53.1936642|8.532772064
9|06-12-22|53.19026936|8.54221344
10|06-12-22|53.18715719|8.551740646
11|06-12-22|53.18581966|8.555774689
12|06-12-22|53.18378756|8.560237885
13|06-12-22|53.18203834|8.566503525
14|06-12-22|53.18111225|8.569335938
15|06-12-22|53.18144667|8.574872017
```

Fig 12: Formatted data being stored in SQLite database

Implementation to store the data initially we planned, is decribed in the table above.

For future implementation, we have decided to include 3 base tables that contains 3 tables namely **accounts**, **track** and **agentinfo**. This is supposed to contain the driver information along with the agents information. These tables are then corelated and cross-referenced to obtain the location of particular vehicle with the drivers information in **track** table.

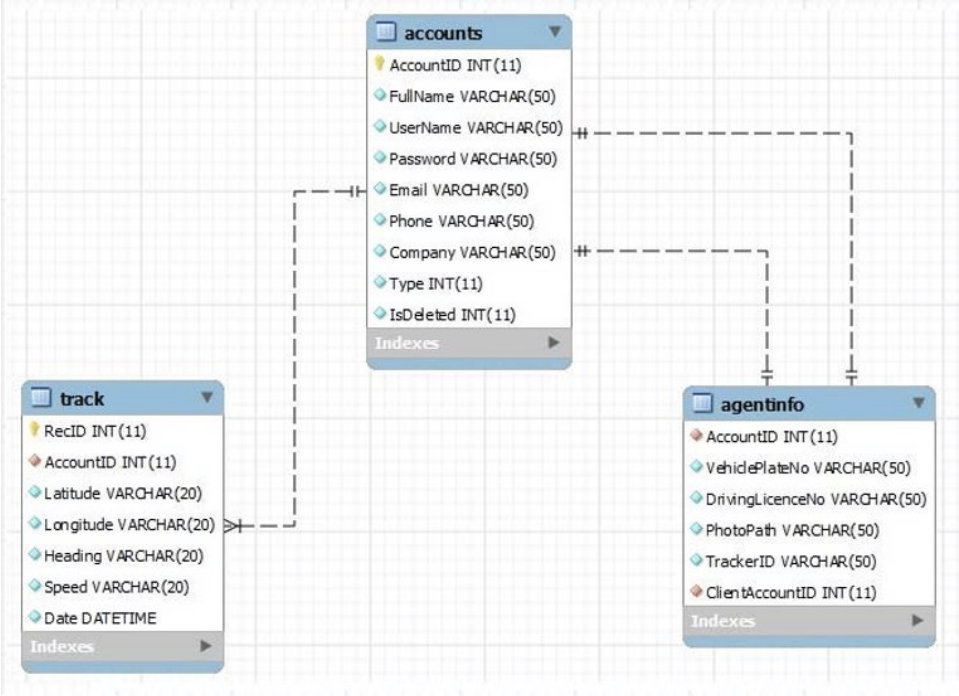


Fig 13: Future implementation of database

Results and Conclusion:

The main objective of our project was to design and construct a cost-effective system to track position or movements of vehicles using a Network based positioning system. We can say the primary objective was achieved successfully. In conclusion of this project, it can be assured that the main objective of this project for Masters course, being the gain of knowledge in relevant area of vehicle movement tracking, gain theoretical and practical knowledge, concluding in the design and development of a working system for the intended end-user was successful. The whole idea was to prototype a working device nothing but GNSS tracker to collect the precise geolocation data to form a tracker device that can be used to generate highly accurate Geo location data, which can then be analyzed for further use.

We are extremely thankful to our supervisor Dr. Fangning Hu, whose constant motivation, inspiration, strong encouragement, and excellent guidance helped us complete the project effortlessly and we are happy that our team members were able to coordinate. Tasks were divided amongst each other in following design:

Abhishek Mathur: Creating tables, handling the database transactions in SQLite and testing various scenarios including future implementation proposal.
Mohammed Alam Tanzil: Idea and Problem identification, coordinating the idea and managing the designed model working.
Chandra Shekar Srinivasaiah: Circuit design, connections, trial runs and end-to-end project testing

Appendix:

Realtime tracking v1.py:

```
import serial
from time import sleep
import webbrowser
import sys

gpgga_info = "$GPGGA,"
ser = serial.Serial ("/dev/Serial0")
GPGGA_buffer = 0
NMEA_buff = 0
lat_in_degrees = 0
long_in_degrees = 0

def GPS_Info():
    global NMEA_buff
    global lat_in_degrees
    global long_in_degrees
    nmea_latitude, nmea_longitude = []
#extract latitude
    nmea_latitude = convert_to_degrees(float(NMEA_buff[1]))
#extract longitude
    nmea_longitude = convert_to_degrees(float(NMEA_buff[3]))
    print ("NMEA Latitude:", nmea_latitude,"NMEA Longitude:", nmea_longitude,'\n')

def convert_to_degrees(raw_value):
    decimal_value = raw_value/100.00
    degrees = int(decimal_value)
    mm_mmmm = (decimal_value - int(decimal_value))/0.6
    position = degrees + mm_mmmm
    position = "%.4f" %(position)
    return position

try:
    while True:
        received_data = (str)(ser.readline()                #read NMEA string
        GPGGA_data_available = received_data.find(gpgga_info)
        if (GPGGA_data_available>0):
            #store data from the string
            GPGGA_buffer = received_data.split("$GPGGA,",1)[1]
            #store CSV in buffer
            NMEA_buff = (GPGGA_buffer.split(','))
            GPS_Info()
```

```
        print("lat in degrees:", nmea_latitude," long in degree: ", nmea_longitude, '\n')
            map_link = 'http://maps.google.com/?q=' + nmea_latitude + ',' + nmea_longitude
```

Code snippet used to take the location via “gps” library and put it to sqlite database:

```
import gps
import time
# Streaming data and functions to initial states
from ISStreamer.Streamer import Streamer
# parsing NMEA data to lat,lon
from geopy.geocoders import Nominatim
from geopy.exc import GeocoderTimedOut
# print exceptions and traceback
import sys, traceback
from sys import exit

# for Open Street Maps (OSM) Geocoding and reverse Geocoding
geolocator = Nominatim()
# extablish session of the GNSS/GPSS program
session = gps.gps("127.0.0.1", "2947")
session.stream(gps.WATCH_ENABLE | gps.WATCH_NEWSTYLE)
publish_data = Streamer(bucket_name="GNSS",bucket_key="7FXC47YPNBUJ",access_key="ist_o6RzYcFogfm4RPBSFFOPOuM2GSunZsHH")
# stream data from raspbery pi to initial satate replace bucket_id and bucket_key
while True:
  try:
time.sleep(0.1)
raw_data = session.next()
if raw_data[class] == GNRMC:
if hasattr(raw_data,lat) and hasattr(raw_data,lon):
publish_data.log("Location","{lat},{lon}".format(lat=raw_data.lat,lon=raw_data.lon))
print ("Latitude is = "+str(raw_data.lat))
print ("Longitude is = "+str(raw_data.lon))
coordinates = str(raw_data.lat) + "," + str(raw_data.lon)
where_it_is = geolocator.reverse(coordinates,timeout=10)
publish_data.log("Vehicle is located at",where_it_is.address)
print(where_it_is.address)
if raw_data[class] ==GNRMC:
if hasattr(raw_data,speed):
publish_data.log("Speed of the vehicle", raw_data.speed)
print ("Vehicle is moving at = "+str(raw_data.speed)+" KPH")
if raw_data[class] ==GNRMC:
if hasattr(raw_data,alt):
publish_data.log("Altitude",raw_data.alt)
print ("The altitude is = "+str(raw_data.alt)+" m")
```

```
if raw_data[class] == GNRMC:
if hasattr(raw_data,time):
publish_data.log("Time",raw_data.time)
print ("The curent date and time is = "+str(raw_data.time)+"\n")

except GeocoderTimedOut as e:
publish_data.log("msg","Geocoder Timeout")
pass
except KeyError:
pass
except (KeyboardInterrupt, SystemExit):
publish_data.close()
print ("\nEnding the current process")
gps.running = False
exit()
quit()
except StopIteration:
session = None
print("No incoming data from the GNSS module")
```