Autonomous Robotics

Ex0 - GNSS Raw Measurements

Submitter:

Shlomit Ashkenazi 323024034

Summery:

In this project I have built an Python script that processes Raw Measurements GNSS log files to compute the receiver's position and outputs the results in both CSV and KML formats.

Sources:

https://www.johnsonmitchelld.com/2021/03/14/least-squares-gps.html

https://github.com/johnsonmitchelld/gnss-analysis/tree/main/notebooks

Environment:

Anaconda, Jupyter Notebook.

Requirements:

Jupyter notebook

Python Navpy

Python Pandas

Python Numpy

Python unlzw3

Python georinex

Python simplekml

Note: you can ensure all requirements on Jupyter Notebook terminal using this command:

```
pip install pandas numpy navp unlzw3 georinex
```

Example:

```
PS C:\Users\israe> pip install pandas numpy navpy
Requirement already satisfied: pandas in c:\users\israe\anaconda3\lib\site-packages (2.0.3)
Requirement already satisfied: numpy in c:\users\israe\anaconda3\lib\site-packages (1.26.4)
Requirement already satisfied: navpy in c:\users\israe\anaconda3\lib\site-packages (1.0)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\israe\anaconda3\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\israe\anaconda3\lib\site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in c:\users\israe\anaconda3\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: six>=1.5 in c:\users\israe\anaconda3\lib\site-packages (from python-dateutil>=2.8.2->panda
```

GitHub repo:

https://github.com/slomit1234/Ex0_GNSS_Raw_Measurements/tree/main

AutoRun:

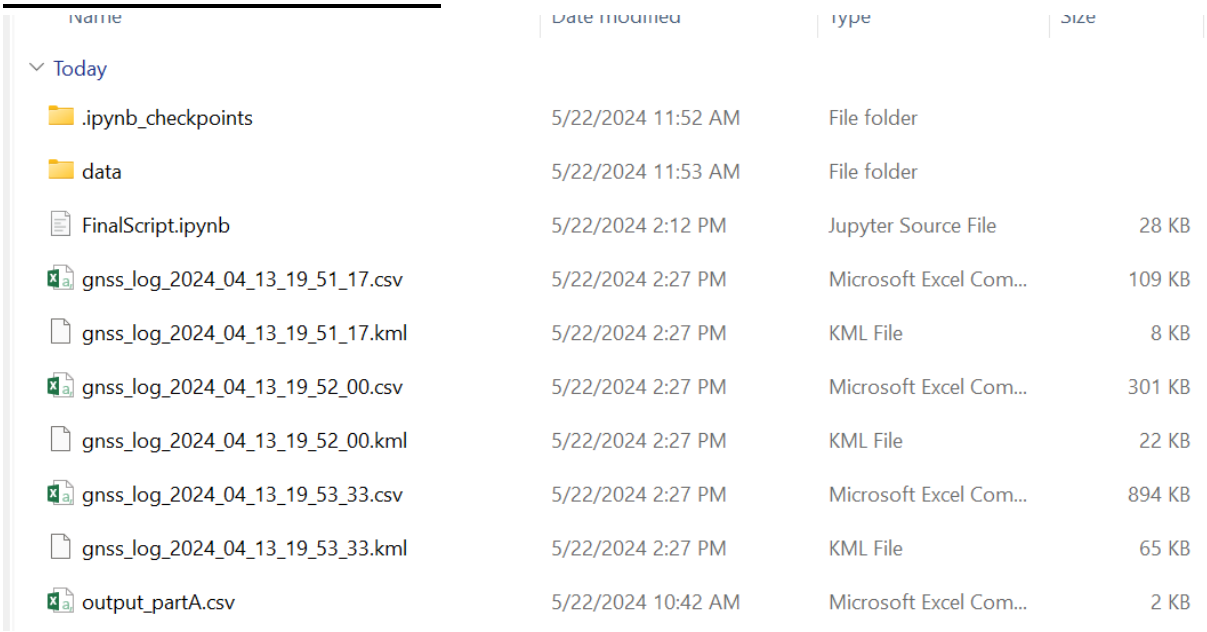
Extract my GitHub repo to your computer and do a quick test run just to see that everything is working properly.

Make sure:

1. You have all the packages that mentioned under “requirements”.
2. All of the GNSS log data are in the same repository as the script (if you extracted all of the repo properly that this is a no-brainer).
3. Run the script “FinalScript.ipynb”

4. It will generate an output file for each log:
   Driving (gnss_log_2024_04_13_19_53_33.txt):
      gnss_log_2024_04_13_19_53_33.kml
      gnss_log_2024_04_13_19_53_33.csv
   Fixed (gnss_log_2024_04_13_19_51_17.txt):
      gnss_log_2024_04_13_19_51_17.kml
      gnss_log_2024_04_13_19_51_17.csv
   Walking(gnss_log_2024_04_13_19_52_00.txt):
      gnss_log_2024_04_13_19_52_00.kml
      gnss_log_2024_04_13_19_52_00.csv

It should look like this:



Name	Date modified	Type	Size
Today			
.ipynb_checkpoints	5/22/2024 11:52 AM	File folder	
data	5/22/2024 11:53 AM	File folder	
FinalScript.ipynb	5/22/2024 2:12 PM	Jupyter Source File	28 KB
gnss_log_2024_04_13_19_51_17.csv	5/22/2024 2:27 PM	Microsoft Excel Com...	109 KB
gnss_log_2024_04_13_19_51_17.kml	5/22/2024 2:27 PM	KML File	8 KB
gnss_log_2024_04_13_19_52_00.csv	5/22/2024 2:27 PM	Microsoft Excel Com...	301 KB
gnss_log_2024_04_13_19_52_00.kml	5/22/2024 2:27 PM	KML File	22 KB
gnss_log_2024_04_13_19_53_33.csv	5/22/2024 2:27 PM	Microsoft Excel Com...	894 KB
gnss_log_2024_04_13_19_53_33.kml	5/22/2024 2:27 PM	KML File	65 KB
output_partA.csv	5/22/2024 10:42 AM	Microsoft Excel Com...	2 KB

Final Solution:

After doing each task (1) – (4) separately,
I incorporated everything into a single final script.

The Script implemented the following steps (as required):

1. Parse the GNSS Log File
2. Compute Satellite Positions and Pseudoranges
3. Positioning Algorithm
4. Output Files

Part A – the parsing tool

This part included the following steps:

1. Read the txt log file an reads each line.
2. Then we need to separate "Fix" and "Raw" data into respective lists.
3. Create the DataFrame so we will convert lists into pandas DataFrames for easier manipulation.
4. Ensures satellite IDs are consistent in length and format.
5. Removes non-GPS measurements.
6. Converts columns to appropriate numeric types for further calculations.
7. Time Handling - we compute the GPS time and split the data into measurement epochs.

The output of part (1), should look like that:

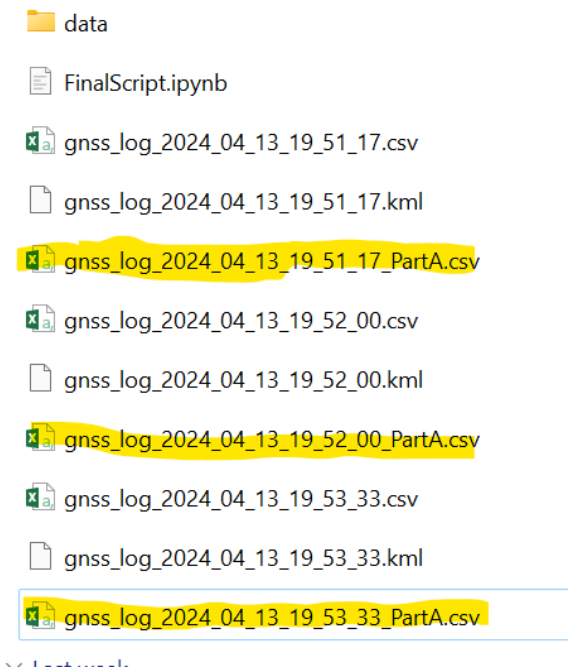
GPS time	SatPRN	Sat.X	Sat.Y	Sat.Z	Pseudo-Ra	CN0	Doppler
2024-04-13 16:51:36.417342720+00:00	G02	14573207	-5239568	22040176	22948159	43	-505.605
2024-04-13 16:51:36.417342720+00:00	G03	23187581	-1.2E+07	4011163	24058778	42	-504.386
2024-04-13 16:51:36.417342720+00:00	G08	24906736	4694271	8528082	21133596	47.8	165.7996
2024-04-13 16:51:36.417342720+00:00	G10	-1329165	17148086	20352420	22788508	39	355.6266
2024-04-13 16:51:36.417342720+00:00	G21	15699709	1536361	21913664	21707764	44.3	-348.52
2024-04-13 16:51:36.417342720+00:00	G27	23150795	13293206	-2593026	22142539	45.2	533.8138
2024-04-13 16:51:36.417342720+00:00	G28	5200268	25758890	-3885489	23880923	35.5	-510.615
2024-04-13 16:51:36.417342720+00:00	G32	8094502	17964730	18113975	21445039	45.6	236.6739

you can see the partA code at the last box of the file “FinalScript”:

 # only part A code (as you request to a attach it):

In [45]:
```
def parse_gnss_log(filepath):
    with open(filepath) as csvfile:
        reader = csv.reader(csvfile)
        for row in reader:
            if row[0][0] == '#':
                if 'Fix' in row[0]:
                    android_fixes = [row[1:]]
                elif 'Raw' in row[0]:
                    measurements = [row[1:]]
            else:
                if row[0] == 'Fix':
                    android_fixes.append(row[1:])
                elif row[0] == 'Raw':
                    measurements.append(row[1:])

    android_fixes = pd.DataFrame(android_fixes[1:], columns=android_fixes[0])
    measurements = pd.DataFrame(measurements[1:], columns=measurements[0])

    # Format satellite IDs
    measurements.loc[measurements['Svid'].str.len() == 1, 'Svid'] = '0' + measurements['Svid']
    measurements.loc[measurements['ConstellationType'] == '1', 'Constellation'] = 'G'
    measurements.loc[measurements['ConstellationType'] == '3', 'Constellation'] = 'R'
    measurements['SvName'] = measurements['Constellation'] + measurements['Svid']

    # Remove all non-GPS measurements
    measurements = measurements.loc[measurements['Constellation'] == 'G']

    # Convert columns to numeric representation
    measurements['Cn0DbHz'] = pd.to_numeric(measurements['Cn0DbHz'])
    measurements['TimeNanos'] = pd.to_numeric(measurements['TimeNanos'])
    measurements['FullBiasNanos'] = pd.to_numeric(measurements['FullBiasNanos'])
    measurements['ReceivedSvTimeNanos'] = pd.to_numeric(measurements['ReceivedSvTimeNanos'])
    measurements['PseudorangeRateMetersPerSecond'] = pd.to_numeric(measurements['PseudorangeRateMetersPerSecond'])
    measurements['ReceivedSvTimeUncertaintyNanos'] = pd.to_numeric(measurements['ReceivedSvTimeUncertaintyNanos'])

    if 'BiasNanos' in measurements.columns:
        measurements['BiasNanos'] = pd.to_numeric(measurements['BiasNanos'])
    else:
        measurements['BiasNanos'] = 0
    if 'TimeOffsetNanos' in measurements.columns:
        measurements['TimeOffsetNanos'] = pd.to_numeric(measurements['TimeOffsetNanos'])
    else:
        measurements['TimeOffsetNanos'] = 0

    measurements['GpsTimeNanos'] = measurements['TimeNanos'] - (measurements['FullBiasNanos'] - measurements['BiasNanos'])
```

Running this in the same matter of will produce the part A files for each log file:



These file also attached in the solution.

Part B – the Positioning algorithm

This part's code is incorporated in "FinalScript.ipynb" under "calculate_satellite_position", " least_squares" functions.

Function: calculate_satellite_position(ephemeris, transmit_time):

The purpose of this function is to compute the satellite positions based on ephemeris data and transmit times.

Details:

1. Orbital Parameters: Uses Keplerian elements to calculate the position of satellites.
2. Corrections: Applies necessary corrections for precise satellite positioning.

Function: least_squares(xs, measured_pseudorange, x0, b0)

The purpose of this function is to compute the receiver's position using an iterative least squares algorithm.

Details:

1. Initial Guesses: Uses initial estimates for receiver's position and clock bias.
2. Iterations: Iteratively adjusts the position and clock bias to minimize the pseudorange errors.

3. Matrix Setup: Constructs the design matrix G and solves for the adjustments.
4. Termination: Continues until the adjustments are sufficiently small or a maximum number of iterations is reached.

Part C – the final solution

As I mentioned before, this part was all about incorporating everything together, and fine-tuning
(like converting from ECEF to LLA).

You can find this part's code under "FinalScript.ipynb" on the GitHub repo.

```
# the final solution (Part c on readme)

In [1]: from ftplib import FTP_TLS, FTP
        import ftplib
        import gzip
        import shutil
        import os
        from datetime import datetime, timedelta, timezone
        import georinex
        import xarray
        import unlzw3
        import pandas as pd
        import numpy as np


        # Took it from  https://github.com/johnsonmitchelld/gnss-analysis/tree/main/notebooks
        class EphemerisManager():
            def __init__(self, data_directory=os.path.join(os.getcwd(), 'data', 'ephemeris')):
                self.data_directory = data_directory
                nasa_dir = os.path.join(data_directory, 'nasa')
                igs_dir = os.path.join(data_directory, 'igs')
                os.makedirs(nasa_dir, exist_ok=True)
                os.makedirs(igs_dir, exist_ok=True)
```

This solution contains to boxes at Jupyter Notebook.

1. The "EphemerisManager" class (https://github.com/johnsonmitchelld/gnss-analysis/tree/main/notebooks)
2. My solution

Testing:

In this part, I will show you all the steps I did to ensure my solution.

First, if you didn't perform a quick test run, look at "AutoRun" above and do it.

After running the scripts, let's test the results.

The csv files:

The csv files contains the computed positions with additional columns Pos.X, Pos.Y, Pos.Z, Lat, Lon, Alt.

Those files contain the full info on each record.

Example:

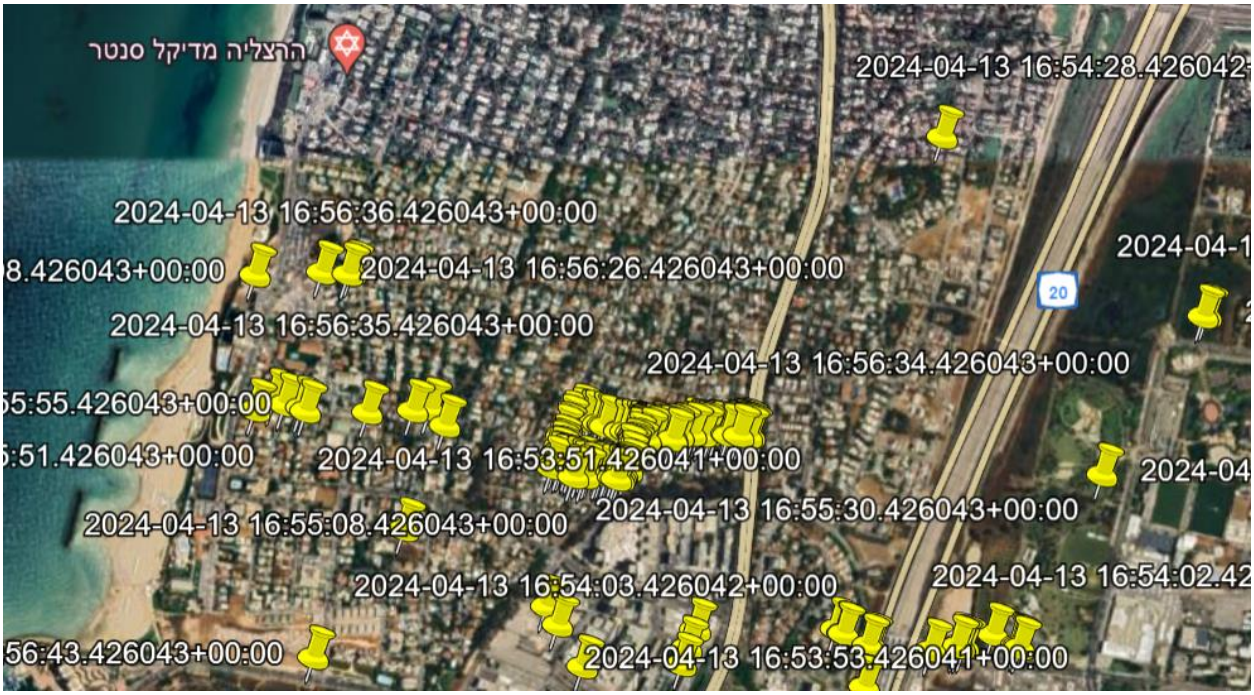
AX	AY	AZ	BA	BB	BC	BD	BE	BF	BG	BH
k	delT_sv	x_k	y_k	z_k	Pos.X	Pos.Y	Pos.Z	Lat	Lon	Alt
096.341	-0.00045	14573207	-5239568	22040176	4436877	3085279	3376311	32.1688	34.81366	38.1373
096.337	0.000309	23187581	-1.2E+07	4011163	4436877	3085279	3376311	32.1688	34.81366	38.1373
096.347	0.00012	24906736	4694271	8528082	4436877	3085279	3376311	32.1688	34.81366	38.1373
112.341	-8.49E-06	-1329165	17148086	20352420	4436877	3085279	3376311	32.1688	34.81366	38.1373
096.345	0.000127	15699709	1536361	21913664	4436877	3085279	3376311	32.1688	34.81366	38.1373
096.343	-1.84E-05	23150795	13293206	-2593026	4436877	3085279	3376311	32.1688	34.81366	38.1373
096.338	-0.0002	5200268	25758890	-3885489	4436877	3085279	3376311	32.1688	34.81366	38.1373
096.346	-0.00062	8094502	17964730	18113975	4436877	3085279	3376311	32.1688	34.81366	38.1373
097.341	-0.00045	14573637	-5236918	22040556	4436883	3085280	3376316	32.16881	34.81364	46.073
097.337	0.000309	23187259	-1.2E+07	4014343	4436883	3085280	3376316	32.16881	34.81364	46.073
097.347	0.00012	24907622	4694947	8525202	4436883	3085280	3376316	32.16881	34.81364	46.073
113.341	-8.49E-06	-1331334	17149414	20351116	4436883	3085280	3376316	32.16881	34.81364	46.073
097.345	0.000127	15700126	1538994	21913270	4436883	3085280	3376316	32.16881	34.81364	46.073
097.343	-1.84E-05	23150430	13293291	-2596155	4436883	3085280	3376316	32.16881	34.81364	46.073
097.338	-0.0002	5200042	25759406	-3882363	4436883	3085280	3376316	32.16881	34.81364	46.073
097.346	-0.00062	8092415	17963877	18115754	4436883	3085280	3376316	32.16881	34.81364	46.073
098.341	-0.00045	14574068	-5234268	22040936	4436875	3085279	3376308	32.16878	34.81367	36.1290
098.337	0.000309	23186937	-1.2E+07	4017523	4436875	3085279	3376308	32.16878	34.81367	36.1290
098.347	0.00012	24908507	4695624	8522322	4436875	3085279	3376308	32.16878	34.81367	36.1290
114.341	-8.49E-06	-1333503	17150741	20349812	4436875	3085279	3376308	32.16878	34.81367	36.1290
098.345	0.000127	15700544	1541626	21912875	4436875	3085279	3376308	32.16878	34.81367	36.1290
098.343	-1.84E-05	23150065	13293376	-2599284	4436875	3085279	3376308	32.16878	34.81367	36.1290

The KML files:

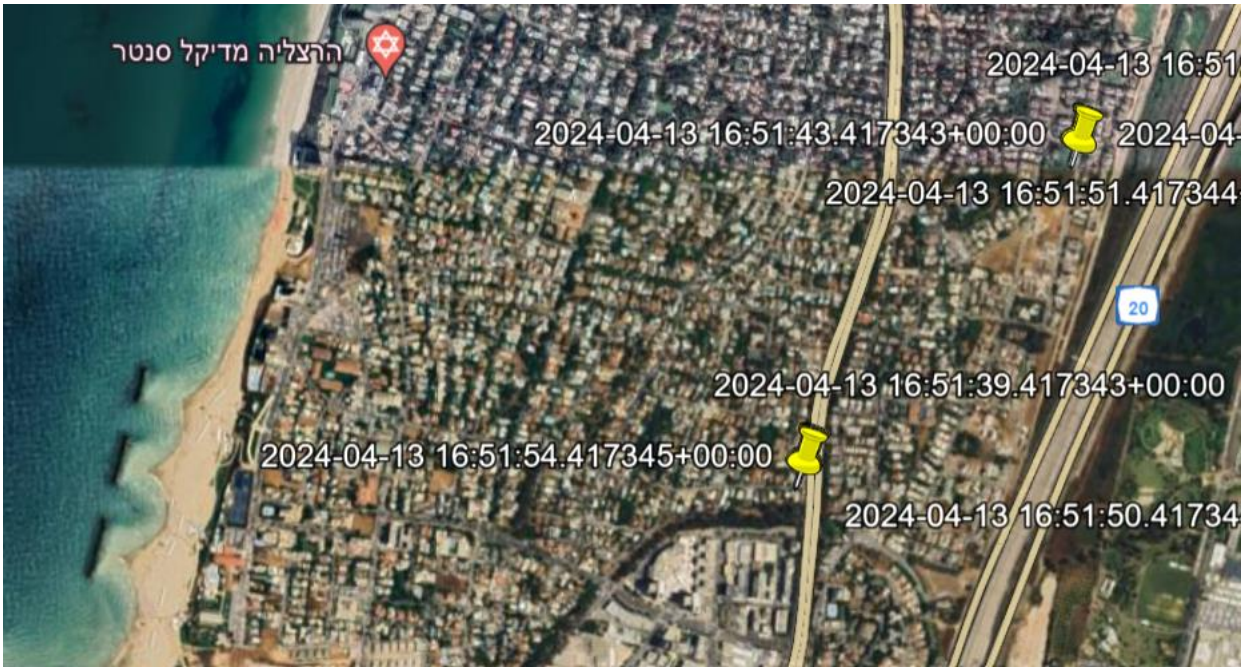
Let’s overview this results using Google Earth.

Upload the KML file to Google Earth.

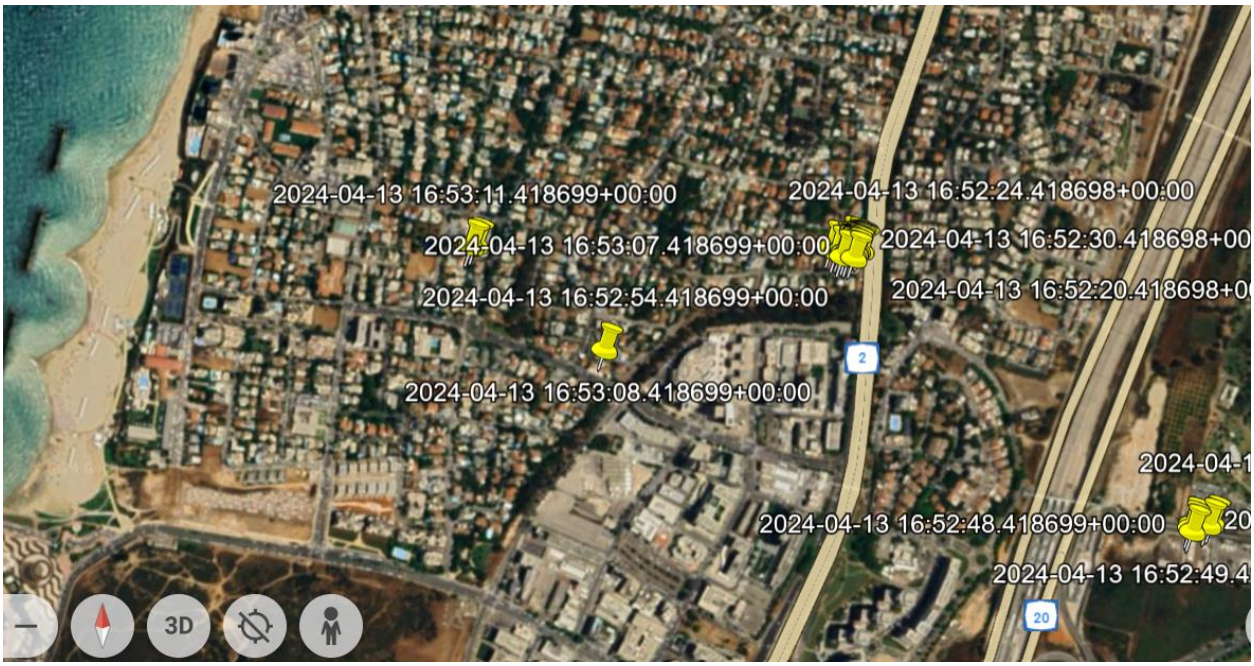
The driving Log data results:



The fixed Log data results:



The walking log data results:



The results match with what I expected them to be

In terms of change between Drive, Walk and Fixed:

1. **In the Drive log** we see a continuous path with long straight segments, we can also see the points on a road. Low variance (a more stable movement).

2. **In the Fixed log** we see minimal changes in coordinates indicating a stationary state. Very low variance in latitude and longitude values.

3. **In the Walking log** we see shorter movements indicative of walking. Higher variance in coordinates due to frequent changes in position.