Title: GNSS Lab Report

Lab 1: GNSS Positioning

Course:	AAE4203 — Guidance and Navigation
Group:	Group 2
Student:	Lee Tsz Pui
Studeent ID:	24015054D
Instructor:	Prof. Weisong WEN
Department:	Department of Aeronautical and Aviation Engineering
School:	The Hong Kong Polytechnic University
Date:	October 24, 2025

Introduction:

The purpose of this lab is to gain hand on experience with GNSS technology and data processing workflows. The lab involves collecting GNSS data using a receiver with the u-centre software, processing raw data with RTKLIB and developing Python algorithm for estimation.

Data Collection:

To start the data collection process, we need to fulfil the following hardware and software requirements.

- Hardware: Laptop, U-blox ZED-F9P + Antenna / U-blox boards
- Software: U-Center

Data collection procedure:

1. Hardware setup: Connect the antenna, receiver and laptop
2. Open u-center. Select the correct COM port for the receiver and set the baud rate to establish communication.
3. Once connected verify the data stream.
4. Start recording. Click the record button in the menu bar and chose the ubx file type.
5. Once collected sufficient data, click the stop button, the ubx file is saved and ready for processing.

Environmental conditions:

Experimental site: Block X, The Hong Kong Polytechnic University

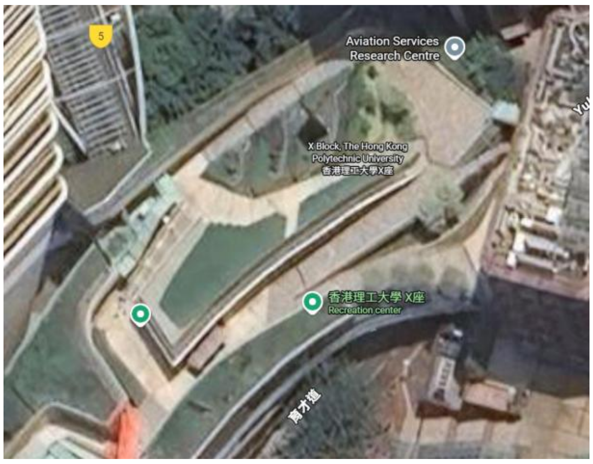

5
Aviation Services Research Centre
X Block, The Hong Kong Polytechnic University 香港理工大學X座
香港理工大學 X座
Recreation center

Figure 1: Collection location

Experimental environment: The location is a semi-open area surrounded by buildings, where the GNSS signals can be blocked or reflected.

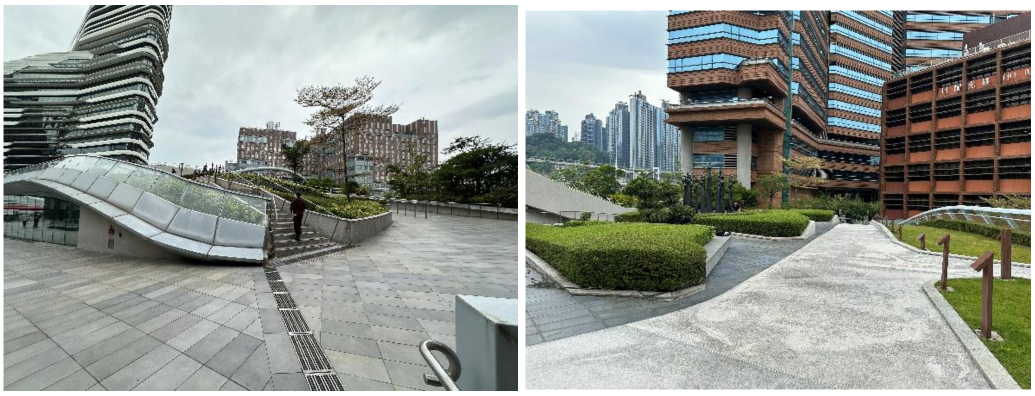


Figure 2: Collection environment

Testing path:

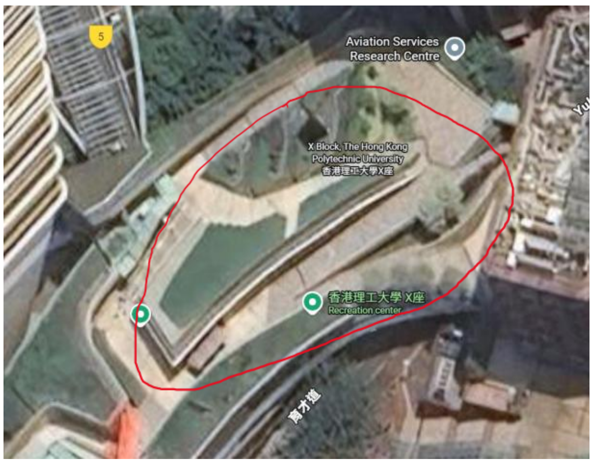

5
Aviation Services Research Centre
X Block, The Hong Kong Polytechnic University 香港理工大學X座
香港理工大學 X座
Recreation center

Figure 3: Testing path

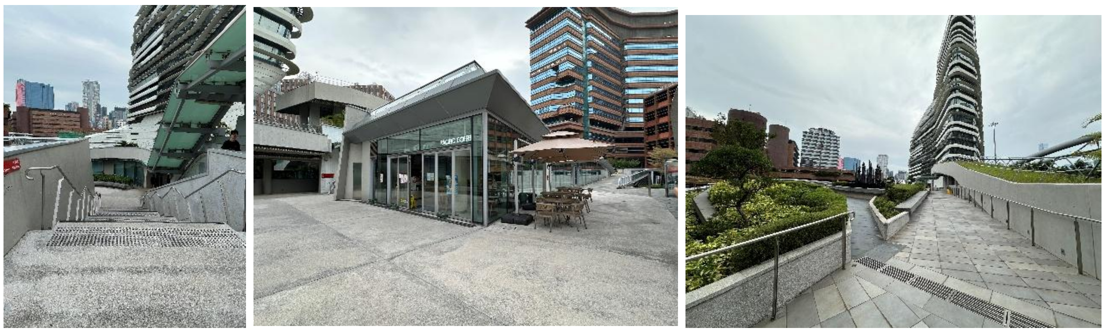


Figure 4: Left (starting point), Middle (Midway), Right (Ending point)

Raw Data Analysis:

After collecting all the data. We need to convert raw data to RINEX. RINEX is the standard format for raw GNSS data. By using the RTKCONV and RTKPOST from the RTKLIB.

Procedures:

1. Open RTKCONV, ensure the format is set to u-blox, and make sure the obs and nav files are checked.
2. Download the precise ephemeris data from the HK SatRef RINEX
3. Use RTKPOST to compute the position.
4. Use the RTKPOST to visualize the results.

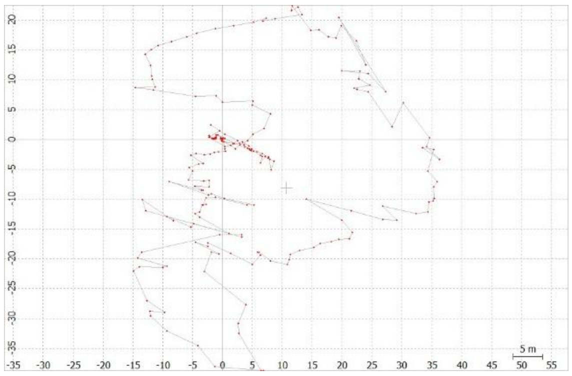


Figure 5: Ground track

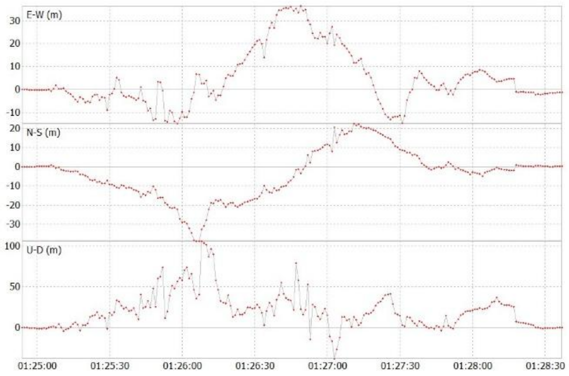


Figure 6: Position over time

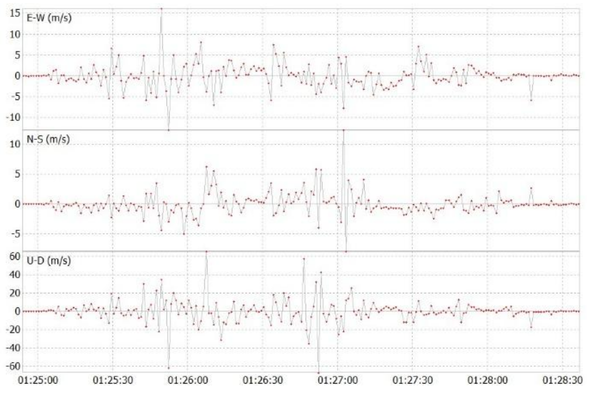


Figure 7: Velocity over time

5. View in Goole Earth: In RTKPOST, convert to KML and use Goole Earth to visualize the track.

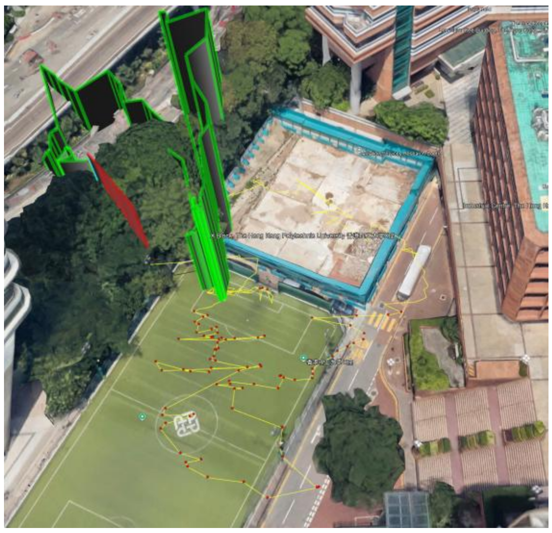


Figure 8: KML generated on Google Earth

RINEX date processing: Using the observation and navigation files to process the detailed signal quality and satellite geometry analysis.

1. SNR analysis: The L1 SNR analysis reveals the signal quality characteristics over the observation period. The SNR plot shows variations in signal strength ranging from 10 to 50 dB-Hz, with color-coded representation indicating signal quality levels.

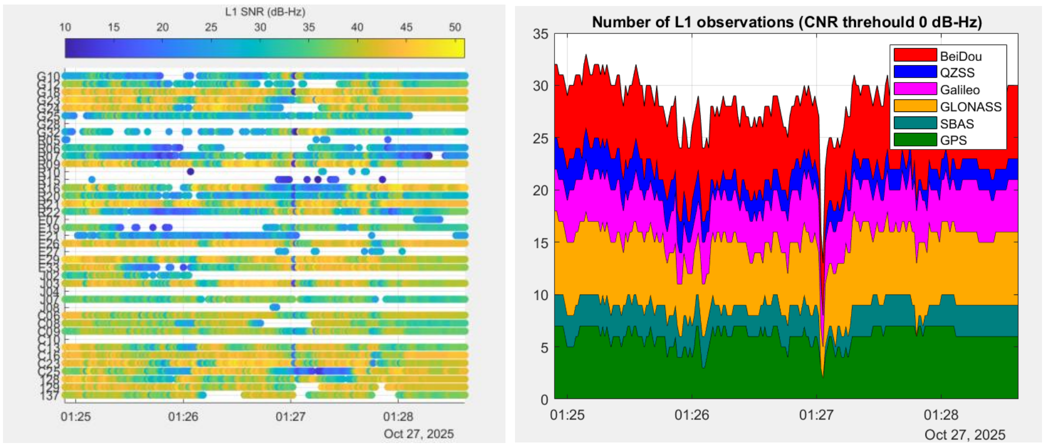


Figure 9: Multi-Constellation Satellite Visibility and SNR Analysis

2. Sky plot and Satellite Geometry: The sky plot visualization provides critical insights into satellite geometry and potential dilution of precision (DOP) effects. The plot shows satellite positions in polar coordinates with azimuth (0°-360°) and elevation angles.

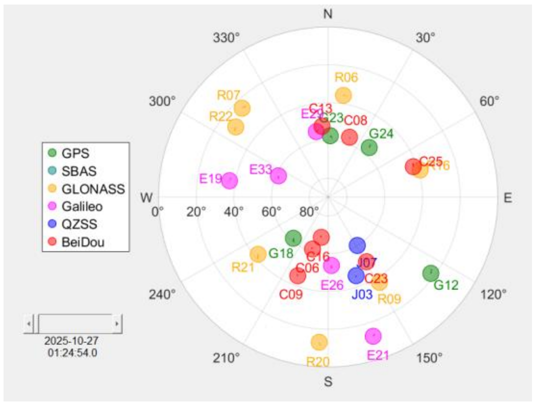


Figure 10: Sky Plot

UBX to CSV conversion for ground truth: To utilize the raw ubx data, we must convert it to readable format such as csv. The pyubx2 provides an efficient tool for this conversion. After the conversion, we will get the ground truth data, the file name is NAV-HPPOSECEF.

SPP Algorithm:

Single Point Positioning (SPP) is a fundamental technique in GNSS, such as GPS, used to determine the position of a receiver using signals from satellites. SPP relies on pseudorange measurements, which are the apparent distances from the receiver to visible satellites, corrected for various errors like satellite clock biases, atmospheric delays, and other factors. Unlike differential methods SPP uses a single receiver without real-time corrections from a base station, making it simpler but less.

Key Principles of SPP:

1. Pseudorange Measurement: The receiver measures the time delay of signals from satellites to compute pseudoranges. The basic equation for each satellite is:

$$P_i = \rho + c(\delta t_r - \delta ts) + I_i + T_i + \epsilon$$

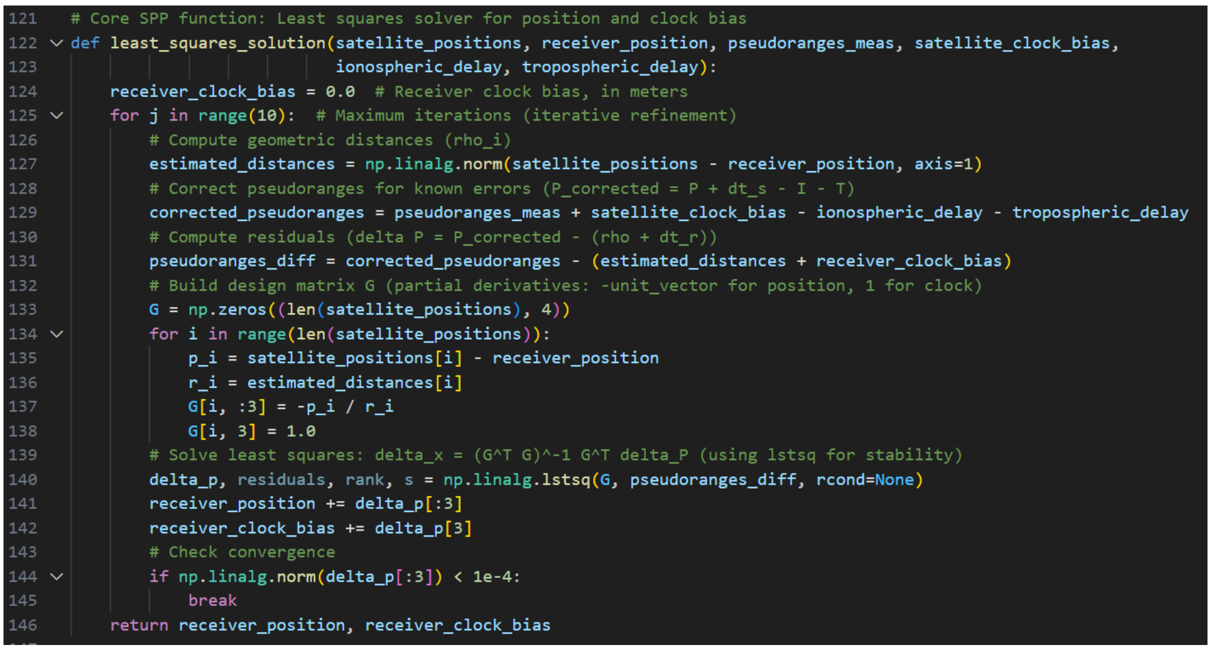
Where:

- ρ_i: True geometric distance between receiver and satellite.
- c: Speed of light.
- δt_r: Receiver clock bias.
- δt_s: Satellite clock bias.
- I_i: Ionospheric delay.
- T_i: Tropospheric delay.

2. Satellite Positions: Known from ephemeris data broadcast by satellites (ECEF coordinates: X, Y, Z).

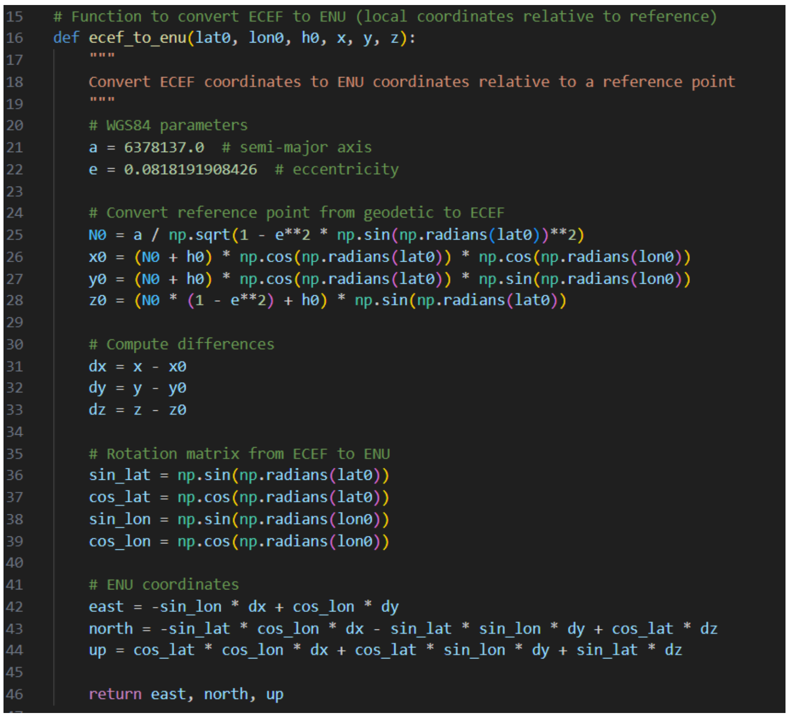
3. Unknowns: The receiver's position (X, Y, Z in ECEF) and clock bias (δt_r). At least 4 satellites are needed to solve for these 4 unknowns.
4. Least Squares Estimation: SPP uses an iterative least squares method to minimize residuals between measured pseudoranges (corrected for known errors) and estimated distances. The system is nonlinear, so it starts with an initial guess (e.g., [0,0,0]) and refines via linearization:
   - Design matrix G includes unit vectors from receiver to satellites and a column for clock bias.
   - Solve, $\Delta x = (G^T G)^{-1} G^T \Delta P$where ΔP are pseudorange residuals.
   - Iterate until convergence (e.g., position change < 1e-4 m).
5. Coordinate Transformations:
   - ECEF to LLA (Latitude, Longitude, Altitude) for human-readable output.
   - ECEF to ENU (East, North, Up) for local tangent plane analysis relative to a reference point.
6. Error Analysis: Compare estimated positions to ground truth (e.g., from high-precision measurements) to compute metrics like RMS (Root Mean Square) error in 2D (horizontal) and 3D.

Python code implementation:



```python
# Core SPP function: Least squares solver for position and clock bias
def least_squares_solution(satellite_positions, receiver_position, pseudoranges_meas, satellite_clock_bias,
                           ionospheric_delay, tropospheric_delay):
    receiver_clock_bias = 0.0  # Receiver clock bias, in meters
    for j in range(10):  # Maximum iterations (iterative refinement)
        # Compute geometric distances (rho_i)
        estimated_distances = np.linalg.norm(satellite_positions - receiver_position, axis=1)
        # Correct pseudoranges for known errors (P_corrected = P + dt_s - I - T)
        corrected_pseudoranges = pseudoranges_meas + satellite_clock_bias - ionospheric_delay - tropospheric_delay
        # Compute residuals (delta P = P_corrected - (rho + dt_r))
        pseudoranges_diff = corrected_pseudoranges - (estimated_distances + receiver_clock_bias)
        # Build design matrix G (partial derivatives: -unit_vector for position, 1 for clock)
        G = np.zeros((len(satellite_positions), 4))
        for i in range(len(satellite_positions)):
            p_i = satellite_positions[i] - receiver_position
            r_i = estimated_distances[i]
            G[i, :3] = -p_i / r_i
            G[i, 3] = 1.0
        # Solve least squares: delta_x = (G^T G)^-1 G^T delta_P (using lstsq for stability)
        delta_p, residuals, rank, s = np.linalg.lstsq(G, pseudoranges_diff, rcond=None)
        receiver_position += delta_p[:3]
        receiver_clock_bias += delta_p[3]
        # Check convergence
        if np.linalg.norm(delta_p[:3]) < 1e-4:
            break
    return receiver_position, receiver_clock_bias
```

Figure 11: Core SPP function



```python
# Function to convert ECEF to ENU (local coordinates relative to reference)
def ecef_to_enu(lat0, lon0, h0, x, y, z):
    """
    Convert ECEF coordinates to ENU coordinates relative to a reference point
    """
    # WGS84 parameters
    a = 6378137.0  # semi-major axis
    e = 0.0818191908426  # eccentricity

    # Convert reference point from geodetic to ECEF
    N0 = a / np.sqrt(1 - e**2 * np.sin(np.radians(lat0))**2)
    x0 = (N0 + h0) * np.cos(np.radians(lat0)) * np.cos(np.radians(lon0))
    y0 = (N0 + h0) * np.cos(np.radians(lat0)) * np.sin(np.radians(lon0))
    z0 = (N0 * (1 - e**2) + h0) * np.sin(np.radians(lat0))

    # Compute differences
    dx = x - x0
    dy = y - y0
    dz = z - z0

    # Rotation matrix from ECEF to ENU
    sin_lat = np.sin(np.radians(lat0))
    cos_lat = np.cos(np.radians(lat0))
    sin_lon = np.sin(np.radians(lon0))
    cos_lon = np.cos(np.radians(lon0))

    # ENU coordinates
    east = -sin_lon * dx + cos_lon * dy
    north = -sin_lat * cos_lon * dx - sin_lat * sin_lon * dy + cos_lat * dz
    up = cos_lat * cos_lon * dx + cos_lat * sin_lon * dy + sin_lat * dz

    return east, north, up
```

Figure 12: ECEF to ENU function

Results and Discussion:

The Python algorithm implements SPP using least squares on pseudorange data from CSV files. It processes epochs (time steps), estimates positions, converts coordinates, computes errors against ground truth, and visualizes results with Matplotlib.

Trajectory Analysis Results:

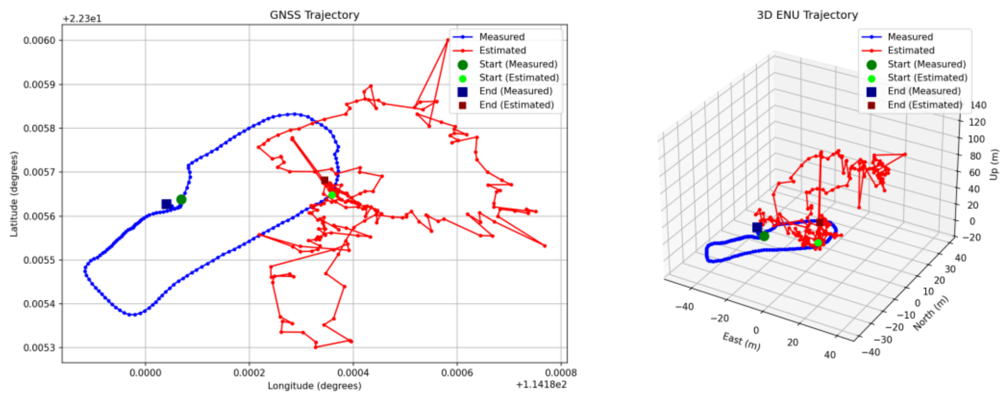


Figure 13: Geographic View and 3D ENU Frame

Trajectory Statistics:

- Total points: 224
- Position accuracy range: 2830.6 - 5341.2 cm

Estimated Position in Latitude, Longitude, Altitude for last epoch:

- Latitude: 22.30568216018537°

- Longitude: 114.18034546627423°
- Altitude: 45.842572652734816m

Directional Positioning Errors:

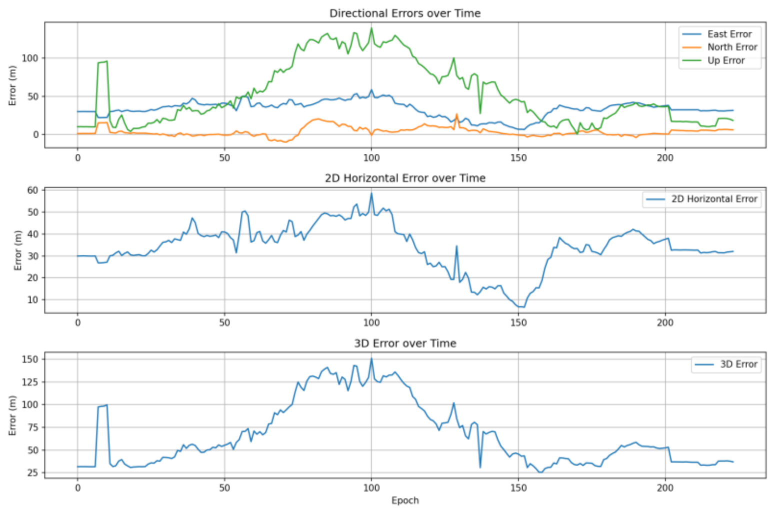


Figure 14: Directional Positioning Errors Analysis (East, North, Up, and 2D/3D Errors)

Error Component	Characteristics	Typical Range
East Error	High variability with decreasing trend over epochs	0-100 meters
North Error	Relatively stable with occasional excursions	-20-40 meters
Up Error	Low variability, consistently stable	-10-10 meters

Table 1: Directional error characteristics

Statistical Error Distribution Analysis:

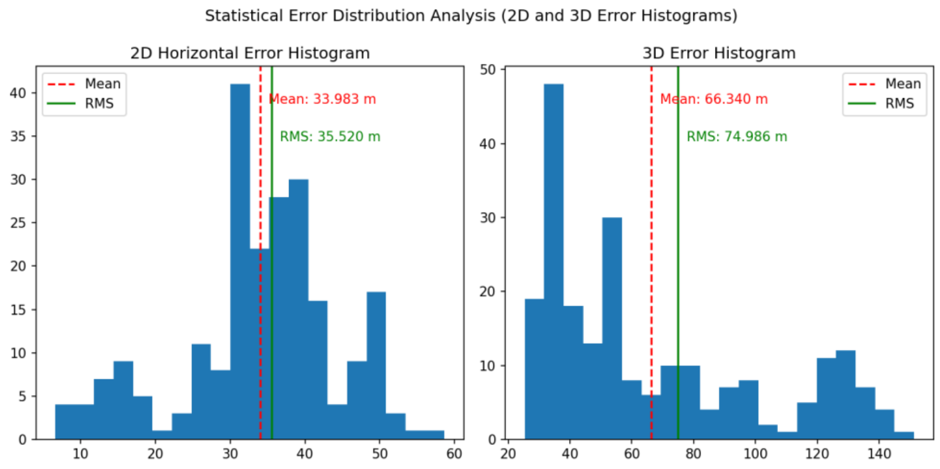


Figure 15: Statistical Error Distribution Analysis (2D and 3D Error Histograms)

Positioning Error Statistics:

- East Errors (m):
  Mean: 33.295, RMS: 34.882, Std: 10.400, Min: 6.486, Max: 58.655
- North Errors (m):
  Mean: 3.472, RMS: 6.706, Std: 5.737, Min: -10.064, Max: 26.863

- Up Errors (m):
  Mean: 52.645, RMS: 66.039, Std: 39.871, Min: 0.300, Max: 139.383
- 2D Horizontal Errors (m):
  Mean: 33.983, RMS: 35.520, Std: 10.336, Min: 6.588, Max: 58.665
- 3D Errors (m):
  Mean: 66.340, RMS: 74.986, Std: 34.955, Min: 25.356, Max: 151.225

Error Evolution and Convergence Analysis:

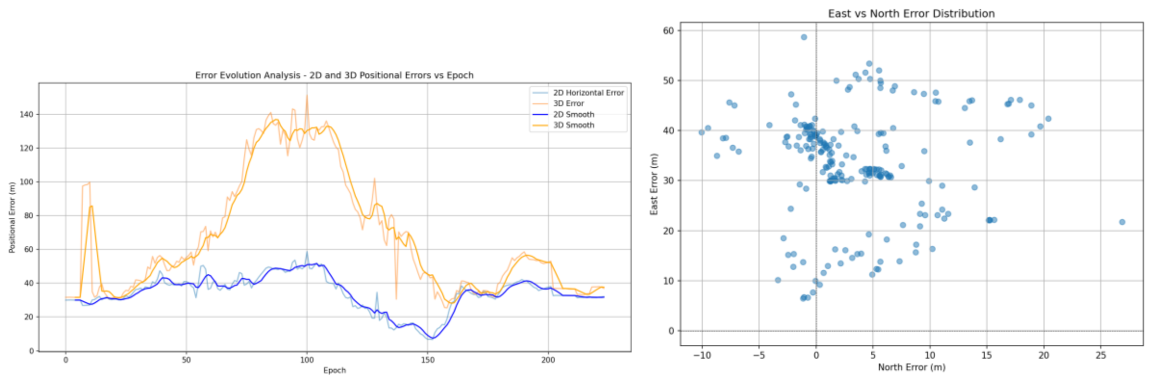


Figure 16: Error Evolution Analysis and East vs North Error Distribution

The error evolution and convergence analysis demonstrates the convergence characteristics of the algorithm:

- Middle phase (50-150) period showing highest errors
- Stabilization phase from epochs 150-200 with consistent performance

---

Conclusion:

With this experiment, we gained hands-on experience of GNSS technology, including on how to conduct data acquisition, process raw signal and estimate the position independently by both existing tool (RTKLIB) as well custom Python code. The laboratory trial showed the success of SPP in estimating receiver positions using pseudorange measurements, with a reasonable accuracy attained on semi-urban environment despite signal multipath and obstruction by surrounding buildings. The most interested insights are the fluctuating SNR and satellite geometry, which is clearly presented by sky plot and DOP analysis, respectively and how (in)significant environmental influences on positional precision were.

In general, this experimentation strengthened the awareness of accurate ephemeris data, environmental factors and the iterative calculation process within GNSS processing. Future work could also be done using RTK techniques or enhanced error mitigation models for accuracy improvements, making it feasible for applications of navigation, surveying and self-driving systems. Theoretical knowledge was supplemented with practical session which allowed us to develop real handling and processing skills.
