Northeastern University
College *of* Engineering

Department of Computer and Electrical Engineering

ME/EECE5554: Robotic Sensing and Navigation

GNSS Driver and GPS Navigation (Lab 1 & 2 Report)

Submitted by

Sakib Azgar

Class Section: Section 2

Course Instructors: Professor Kris Dorsey and Professor Mohammad Javadi

Course TAs: Mani Chandhan Chakinala, Francis Jacob Kalliath, Aditya Bondada, Kaustubh Chaudhari, Arunbhaarathi Anbu, Thomas Consi

Background Information

The Global Positioning System (GPS) has revolutionized navigation and spatial analysis by providing precise, real-time location data. This report focuses on exploring and applying GPS technologies, with a particular emphasis on pseudo ranging techniques and the shift towards Real-Time Kinematic (RTK) positioning for accuracy comparisons.

The first part of the study teaches the basics of GPS, using a GNSS receiver to collect and analyze data. It's designed to help students understand tasks like getting data from a GPS device, changing it into Universal Transverse Mercator (UTM) coordinates, and creating custom ROS messages. The other part of this lab examines RTK GNSS, comparing its accuracy with standalone GNSS data. This comparison highlights RTK's potential to significantly improve location data precision, particularly in challenging environments. The impact of RTK technology on GNSS measurement reliability and accuracy is analyzed through data sets collected under varied conditions [1].

At the core of this exploration lie fundamental equations crucial to GPS technology. For instance, determining a position from satellite signals involves calculating the distance to the satellite, considering the speed of light and the signal's time of flight, as described below:

$$d = c * t \tag{1}$$

Equation 1 here describes this relationship, where d represents the distance to the satellite, c is the speed of light, and t is the signal's time of flight.

In ensuring the precision and reliability of GPS technologies, addressing GNSS vulnerabilities is crucial. A comprehensive review highlights the susceptibility of GNSS systems to various interferences like jamming and spoofing, emphasizing the necessity of integrating GNSS with other localization techniques to enhance robustness, especially in urban environments. This perspective aligns with our lab's exploration of GPS and RTK GNSS technologies, emphasizing the importance of adopting multifaceted approaches to counteract GNSS vulnerabilities for accurate and reliable positioning.

Recognizing the changing landscape of GNSS adds depth to the understanding. From the original U.S. GPS system to today's global systems like Russia's GLONASS, China's Beidou, and Europe's Galileo, significant evolution has occurred. This evolution underscores the lab's focus on GPS and RTK GNSS technologies, showcasing a shift from early excitement to an emphasis on methods ensuring precise satellite surveying [2]. Thus, the lab serves not only as an exploration of GPS and RTK GNSS technologies but also as a reflection on the historical development and methodological rigor that have advanced the field to its current state of innovation and precision.

Procedure

This laboratory exercise required the deployment of Ubuntu (Linux), Visual Studio Code (Python) and a GPS puck provided by the course teaching assistants. The initial phase of the lab involved the development of a functional driver, guided by comprehensive documentation available on EECE5554 lab1 GitHub repository. The creation of the driver (standalone_driver.py), entailed the

processing of data from a GPS sensor, specifically the class-provided GPS puck, connected via a serial port. Data collection was facilitated through the GPS puck, necessitating the acquisition of three distinct sets of data using ROSbags. This included a five-minute interval of stationary data in an area obstructed by buildings near the Interdisciplinary Science and Engineering Complex (ISEC) and EXP buildings, another five-minute interval in an unobstructed space at Carter Field near Northeastern University, and a 200–meter walking dataset along Columbus Avenue, transitioning from an occluded to an open area. After data collection, the GPS data extracted from ROS bags were converted into CSV and ODS files for advanced data analysis.

The second part of the laboratory, focusing on the RTK driver (rtk_driver.py), necessitated minor modifications to the standalone driver, employing resources from EECE5554 lab2 GitHub repository. Unlike the first lab, data for this segment was provided, eliminating the need for personal data collection. This pre-supplied data was then transformed into ROSbags, which were later converted into CSV and ODS files for comprehensive data analysis. The analytical phase for both laboratory exercises followed identical procedures to generate plots, outlining the methodological approach undertaken in these laboratory tasks.

Methods, Discussions, and Analysis

Section A: Standalone_driver.py and rtk_driver.py Methods

This process included parsing GPGGA/GNGGA sentences to extract GPS information, converting latitude and longitude coordinates into Universal Transverse Mercator (UTM) format, and subsequently publishing this data as a custom ROS message. The script was also designed to manage various error conditions and exceptions to ensure uninterrupted operation within a ROS framework.

Seven main functions were created. The first function examines whether the input string contains the GPGGA prefix, which is specific to GPS data, returning True if the prefix is found and False otherwise. The second function involved the conversion of degree and decimal minutes to decimal degrees format, with the basic equation utilized for this purpose.

$$DD = D + \frac{M}{60} \tag{2}$$

DD means Decimal Degrees, D is degrees portion of coordinate, M is decimal minutes portion of coordinate.

The third function involved converting western latitude and southern latitude coordinates, incorporating negative values as necessary. Check piecewise equation below.

$$DD = \begin{cases} -DD\ ; if\ direction\ is\ S\ or\ W \\ DD\ ; if\ direction\ is\ N\ or\ E \end{cases} \tag{3}$$

The fourth function converts geographic coordinates (latitude and longitude) into UTM coordinates, while the fifth function handles the conversion from UTC to Epoch Time using a variety of equations.

$$UTC_{secs} = (H * 3600) + (M * 60) + S + \frac{D_{seconds}}{100} \quad (4)$$

$$Epoch_{Start} = Epoch_{CurrentTime} - \left(Epoch_{CurrentTime} mod(86400)\right) + (5 * 3600) \quad (5)$$

$$Epoch_{time} = Epoch_{Start} + UTC_{secs} \quad (6)$$

$$Epoch_{time\_nanoseconds} = int[(Epoch_{time})] * 10^9 \quad (7)$$

Equation 4 (H represents hours, M represents minutes, S represents seconds, and $D_{seconds}$ represents decimal seconds) calculates the total time since midnight. It is essential for determining the exact number of seconds that have elapsed since midnight, considering hours, minutes, seconds, and any fractional seconds. This is critical for accurately calculating the specific point in time within a day. Equation 5 adjusts the current epoch time to the start of the day according to a specific timezone. This is necessary to calculate times accurately for events or logs in a localized context, ensuring the base time is correctly aligned with the local day's beginning. Equation 6 uses results from (4) and (5) since it combines the adjusted start of the day with the total seconds since midnight to find the epoch time for a specific moment within the day, allowing for precise timestamping of events or data. Equation 7 uses (6) results in turning Epoch time into nanoseconds and seconds for later conversions. These equations are necessary when dealing with time from various places across the world with different time zones.

The sixth and seventh functions from the drivers serve to translate data into the serial port and establish a publisher responsible for generating a node and a topic. These functions operate by continually monitoring the serial port for incoming data. When data becomes available, they retrieve lines from the serial port. If a line contains GPGGA data, the functions utilize the publish_ros_message() method to broadcast a ROS message. This method interprets the GPGGA data, isolates pertinent GPS details like latitude, longitude, and HDOP, transforms coordinates into UTM format, and constructs a bespoke ROS message housing this information. Subsequently, this message is disseminated to a ROS topic designated as 'GPS_Reader'.

Section B: CSV and ods file method

After the data from both drivers was collected in ROSbags, each bag was converted into CSV files. The motivation behind this conversion was to enhance effectiveness. Familiarity with interpreting data from spreadsheets, owing to experience with MATLAB, led to this decision. Conducting extensive graph analysis using data extracted from CSV files and integrating them into the code path file was a common practice. CSV and ODS files provide a more accessible format for preliminary data analysis compared to ROSbags. Thus, the creation of the bag_to_csv.py script facilitated this conversion.

Data cleaning proved essential due to variations in the organization of spreadsheet files. For instance, in the ROSbags obtained from standalone_driver.py, there were noticeable gaps between each row. These gaps complicated the analysis of data from different columns, particularly when examining northing or easting values [3].

Section C: Stationary Occluded and Open Datasets

The standalone_driver.py script extracts data from GPS pucks and converts it into .ods files for further analysis, while the rtk_driver.py uses a similar method, except it takes data from a text file. Both scripts implement a method to filter out unnecessary data and subsequently plot both occluded and open data on a single graph.

Regarding the calculation of centroids for ROSbags obtained from standalone_driver.py and rtk_driver.py, a straightforward approach was adopted by computing the mean of the x and y coordinates. While some sources, like Stack Exchange, advocate for this method, others suggest an alternative approach where the centroid is calculated by taking the average of all points and dividing it by the total number of points. However, for simplicity and consistency, the decision was made to stick with computing the mean of the points.

The centroid mean equation used for calculating the points is as follows:

$$(x_{centroid}, y_{centroid}) = (\frac{x_{easting-sum}}{N_{easting}}, \frac{y_{northing-su}}{N_{northing}}) \quad (8)$$

In (8) provided, the variable $x_{centroid}$ denotes the x-coordinate (easting) of the centroid, while $y_{centroid}$ represents the y-coordinate (northing) of the centroid. Moreover, $x_{easting-su}$ signifies the sum of all easting values, $y_{northing-su}$ represents sum of all northing values, and $N_{easting}$ and $N_{northing}$ denote the total number of values for easting and northing, respectively.

An error occurred in this process as the centroid was not correctly recentered to 0, potentially leading to varying results. However, this mistake was rectified when generating a similar plot for rtk_driver.py. Following the centroid calculation, the Euclidean distance formula was employed to compute the distance of each point from the centroid.

$$D_i = \sqrt{\left(x_{i,point} - x_{centroid}\right)^2 + \left(y_{i,point} - y_{centroid}\right)^2} \quad (9)$$

Here, $x_{i,point}$ and $y_{i,point}$ denote the easting and northing coordinates of the i^{th} data point, respectively. D_i represents the Euclidean distance.

The stationary occluded and open data points obtained from the standalone driver and RTK driver were plotted using Matplotlib in Python. In the case of the standalone driver, both occluded and open data points were shared on a single plot. However, for the occluded and open data extracted from the RTK ROSbags, subplots were utilized instead of a shared plot to enhance visualization.

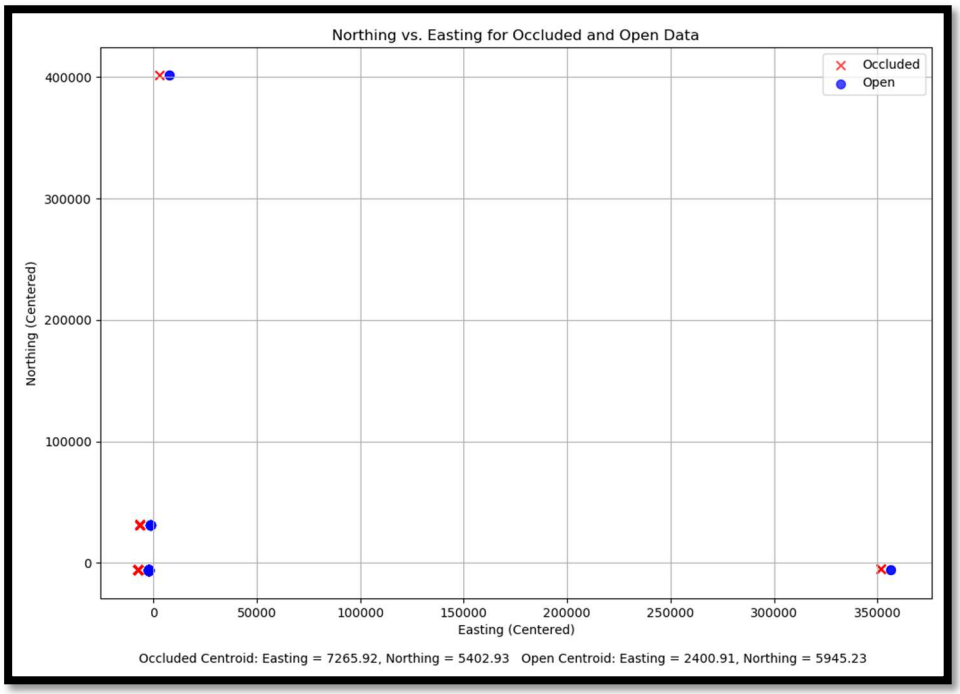
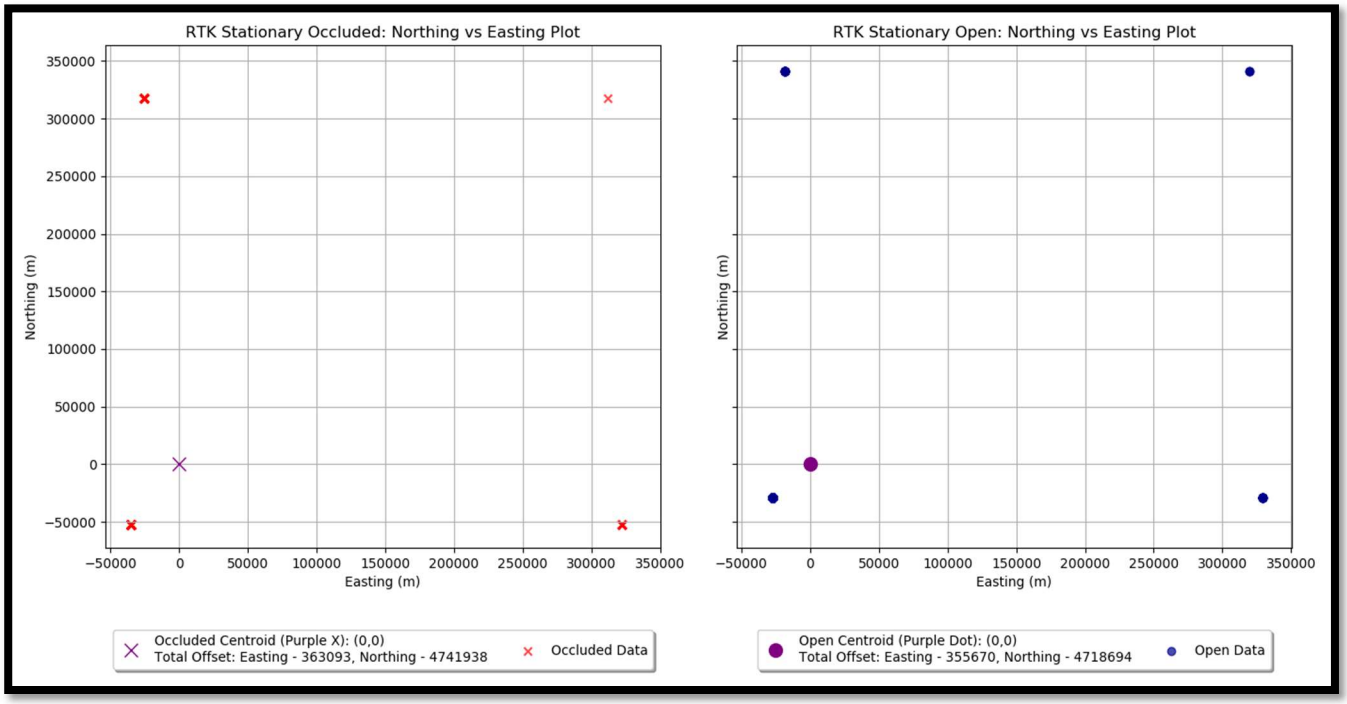


Fig.1. Scatterplot of open and occluded data collected from ROSBags using the Standalone Driver, with the centroid (not recentered) included. Northing and Easting values are measured in meters.



(a) (b)

Fig.2. Scatterplots of (a) open and (b) occluded data collected from ROSBags using the Standalone Driver, with centroid recentered and total offset included. Northing and Easting values are measured in meters.

Based on the stationary plots (Fig.1 and Fig.2), there seems to be a lot of error from the collected data. There are several ways to calculate errors such as the Average Euclidean Error or Root Mean Square Error (RSME).

$$Euclidean_{average\ error} = \frac{\sum_{i=1}^{n}(D_i)^2}{n} \quad (10)$$

Here D_i represents the Euclidean distance for the i^{th} point, and n represents the total number of points. The average error for the dataset is the mean of these distances. The RMSE is a measure of the differences between values predicted by a model, or an estimator and the values observed.

$$\text{RSME} = \sqrt{\frac{\sum_{i=1}^{n}(D_i)^2}{n}} \quad (11)$$

For the data obtained from the standalone driver, the occluded dataset exhibited centroid coordinates of (Easting = 7265.92, Northing = 5402.93). When computing the average Euclidean error, it yielded approximately 20,148.52 units, and the RSME value was approximately 60,626.52 units. Conversely, the open dataset displayed centroid coordinates of (Easting = 2400.91, Northing = 5945.23). Here, the average Euclidean error was found to be 13,345.76 units, while the RSME value was approximately 44,018.42 units.

In the data obtained from the RTK driver, the occluded dataset presented centroid coordinates of (Easting = 0, Northing = 0) with an offset of (Easting = -363093, Northing = -4741938). Upon computing the average Euclidean error, it resulted in approximately 123,566.54 units, while the RSME value was approximately 165,858.23 units. Conversely, the open dataset exhibited centroid coordinates of (Easting = 0, Northing = 0) with an offset of (Easting = -355670, Northing = -47118694). Here, the average Euclidean error was calculated to be 84,328.65 units, and the RSME value was approximately 137,062.98 units. Unlike the Standalone driver, the Euclidean and RSME errors were determined based on the recentered centroid.

A unified metric was derived for both the average Euclidean error and RMSE across the occluded and open datasets. This involved computing separate average errors for each dataset and then averaging them with weights based on the number of data points. Subsequently, the combined RMSE was obtained by merging squared distances from each dataset into a single series, calculating their mean, and taking the square root.

Error estimates in Real-Time Kinematic (RTK) systems compared to standalone receivers vary due to different factors affecting GNSS fix quality. In obstructed environments, signal degradation from obstructions leads to multipath errors and attenuation, particularly impacting RTK systems' precision. In open environments, RTK systems benefit from unobstructed signals, enabling centimeter-level accuracy by correcting for atmospheric delays and other errors. However, standalone receivers, lacking advanced corrections, are inherently less accurate, especially in open conditions where they rely solely on general GNSS signals. RTK's dependence on a reference station and its ability to mitigate multipath effects and atmospheric delays contribute to higher accuracy, despite being vulnerable to signal obstructions compared to standalone systems [4].

Section D: Walking Data Northing vs Easting Plots

Data from walking trials were captured for both the Standalone driver and RTK driver. Unlike the stationary plots, where Euclidean distances and centroids were calculated, the moving data for both drivers were truncated, and line of best fits were generated instead. This approach allowed for a dynamic visualization of the trajectory patterns during walking trials and provided insights into the drivers' performance in mobile scenarios.

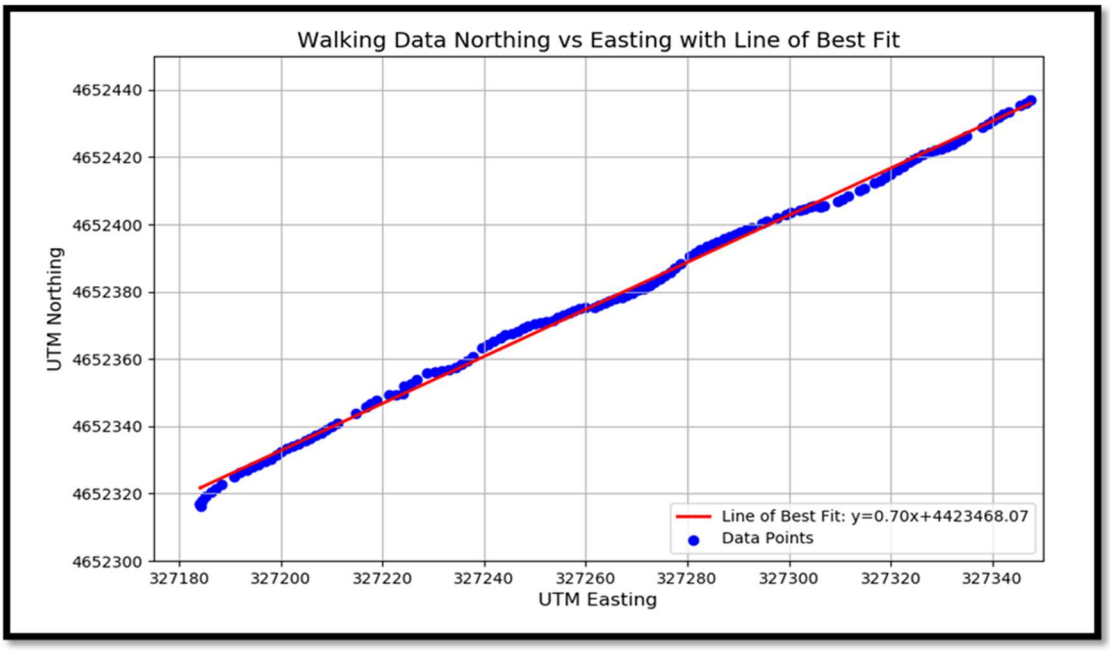


Fig.3. Walking Data Northing vs Easting Scatterplot with line of best fit for Standalone driver data

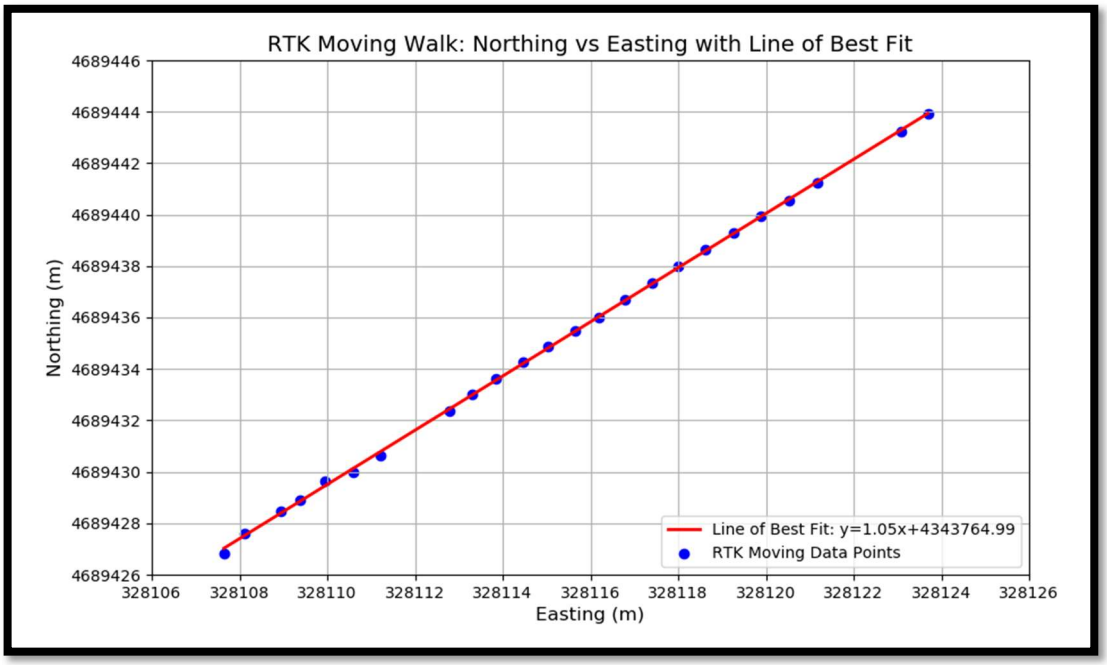


Fig.4. Walking Data Northing vs Easting Scatterplot with line of best fit for RTK driver data

The slope of the data collected from the drivers differs between Fig.3 and Fig.4. In Fig.3, the slope was 0.70, whereas in Fig.4, it was 1.05. A larger absolute value of the slope indicates a steeper path, while a smaller absolute value indicates a flatter path. These values align with the context, as the data for Fig.3 was collected during a walk along Colombus Ave, Boston, Massachusetts, a relatively flat area, whereas the data for Fig.4 was gathered in a steeper terrain.

There is error present in both data due to GPS signaling issues; the standard error of slope formula should be used to calculate the error of the slopes to get a closer look at steep and flat terrain conclusions [5].

$$SE = \sqrt{\frac{[\sum(y_i - \widehat{y_I})^2/(n-2)]}{\sum(x_i - \bar{x})^2}} \tag{12}$$

The following parameters are used in (12): y_i are the observed values for northing, $\widehat{y_I}$ are the predicted values for northing, x_i are the observed values for easting, $\bar{x}$ are the mean values for easting, n represents number of observations.

Using (12) for both the Standalone driver and RTK driver walking datasets, the standard error for the slope was calculated as 0.0482 for the Standalone driver and 0.3396 for the RTK driver. A lower error indicates more precise estimates. Typically, RTK data should be more precise than Standalone GPS data, but the larger error in the RTK dataset could be attributed to the fewer data points in the truncated region. Additionally, the error could stem from the location where the data was collected and potential systematic errors from the computer recording the data. As for the general known values for GPS error, it can vary depending on factors such as signal quality, atmospheric conditions, and receiver type. However, typical GPS errors range from a few meters to tens of meters in non-differential GPS systems, while RTK systems can achieve centimeter-level accuracy under optimal conditions [5].

Section E: GPS and RTK Errors from Section C and D

In evaluating the errors within the data received from the drivers, it is essential to initially examine errors associated with GPS measurements. There are three primary errors that may arise with GPS measurements: atmospheric effects, satellite shading, and multipath errors. Atmospheric effects occur when GPS satellite signals traverse the Earth's atmosphere, including the ionosphere and troposphere. Variations in the density and moisture content of these atmospheric layers can lead to signal transmission delays, thereby introducing errors in position estimates. Satellite shading, also referred to as Dilution of Precision (DOP), pertains to the relative positions of GPS satellites in the sky. If satellites are closely positioned from the receiver's perspective, poor geometry can result, leading to larger position errors. Conversely, widely spaced satellites indicate good geometry, resulting in more precise readings. Multipath errors occur when GPS signals bounce off large objects such as buildings or rock faces before reaching the GPS receiver, causing the signal to travel a longer path, and resulting in errors in the calculated position [6].

Two of the primary errors in GPS measurements, atmospheric effects, and multipath errors, also contribute to the errors in Standalone GPS datasets. Standalone GPS units are susceptible to

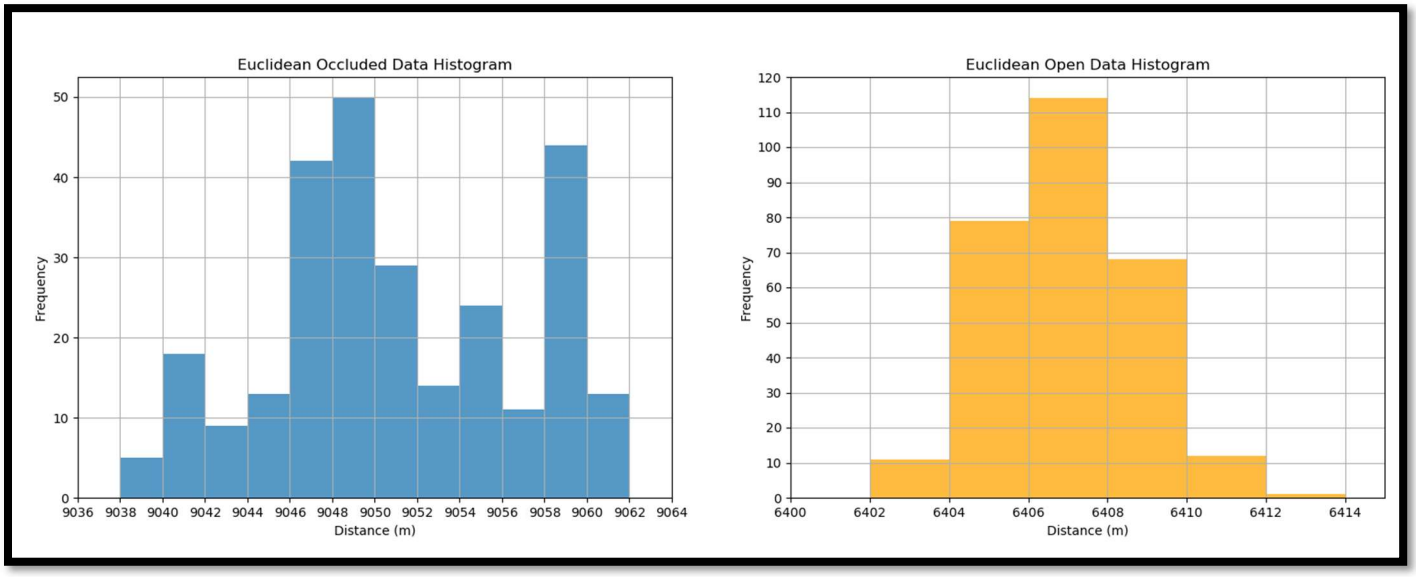
atmospheric conditions, especially without ground-based augmentation or advanced atmospheric modeling. The standard error in the dataset reflects potential noise from atmospheric interference, particularly noticeable when the receiver is in motion amidst changing conditions. Similarly, in environments with tall buildings or reflective surfaces, multipath errors can significantly affect GPS accuracy. The standard error of the slope captures these effects, particularly in urban areas where such errors are more common.

When comparing estimated error values for stationary versus moving datasets in both standalone GPS and RTK GPS systems, distinct patterns emerge. In standalone GPS datasets, the error tends to be larger for moving data due to the heightened probability of encountering various sources of error while in motion. Factors such as changing atmospheric conditions and varied multipath effects contribute to this increased error. Conversely, the error is often lower for stationary datasets because the receiver could average out some of the noise over time. In contrast, RTK GPS systems utilize a fixed base station to provide corrections, significantly reducing errors attributable to atmospheric conditions and enhancing measurement precision. Consequently, the error is typically smaller for RTK systems, regardless of whether the receiver is stationary or in motion. However, if movement introduces less favorable satellite geometry or increased multipath errors, this could result in a higher error in moving datasets compared to stationary ones [7].

Based on the driver's results, the Standalone GPS system rely solely on satellite signals without ground-based augmentation; this leaves the system susceptible to new errors introduced by changing environmental conditions during movement that are absent when stationary. Conversely, in RTK GPS systems, despite benefiting from corrections from the base station which enhance accuracy, the accuracy may diminish slightly during movement due to noise affecting the location-specific corrections. This analysis suggests that for standalone GPS, measurements tend to be more accurate when stationary due to reduced environmental noise affecting the signal. On the other hand, with RTK GPS, overall accuracy is typically higher, yet may decrease slightly when in motion, contingent upon the environmental conditions and the extent of movement relative to the base station's corrections.

Section F: Histogram Plots and Analysis

For the open and occluded data sets, the Euclidean distances and centroid values for both Standalone driver and rtk driver were plotted on histogram plots. These plots are often used for evaluating error characteristics and comparing performance under different environmental conditions. RTK GPS, for example, is expected to be less affected by obstructions due to its correction mechanisms.



(a) (b)

Fig.5. Histogram Plots of Euclidean Distances from Standalone Driver Data - (a) Occluded Region, (b) Open Region.

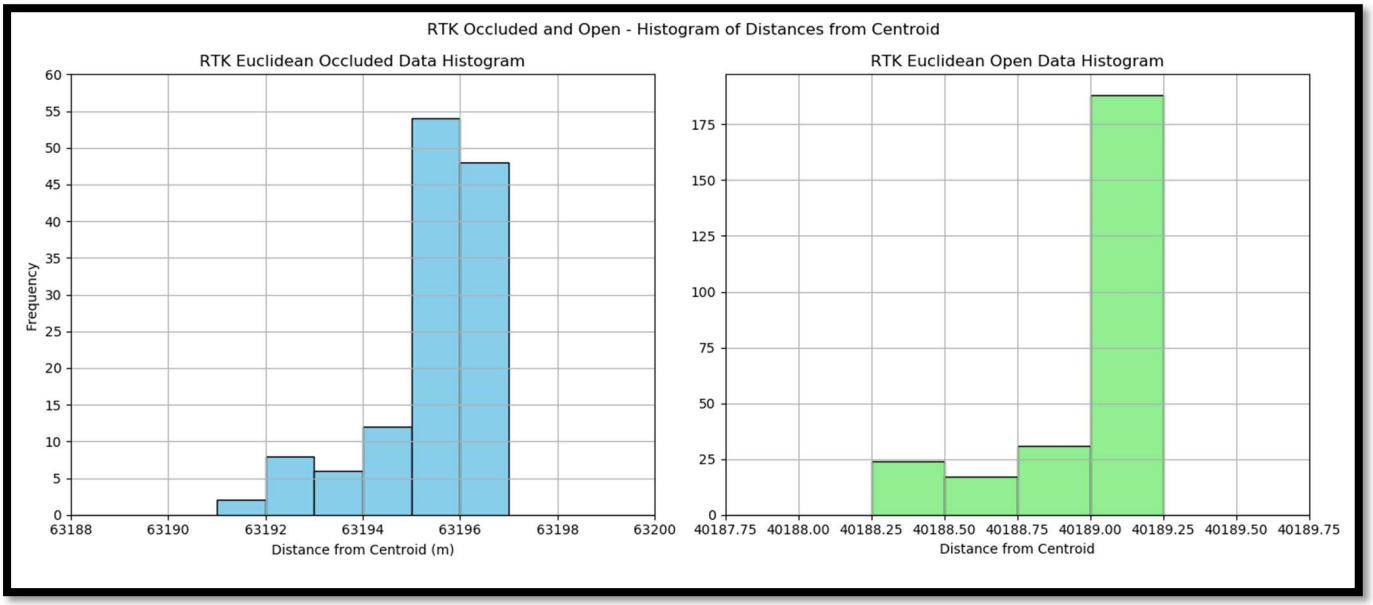


Fig.6. Histogram Plots of Euclidean Distances from RTK Driver Data - (a) Occluded Region, (b) Open Region

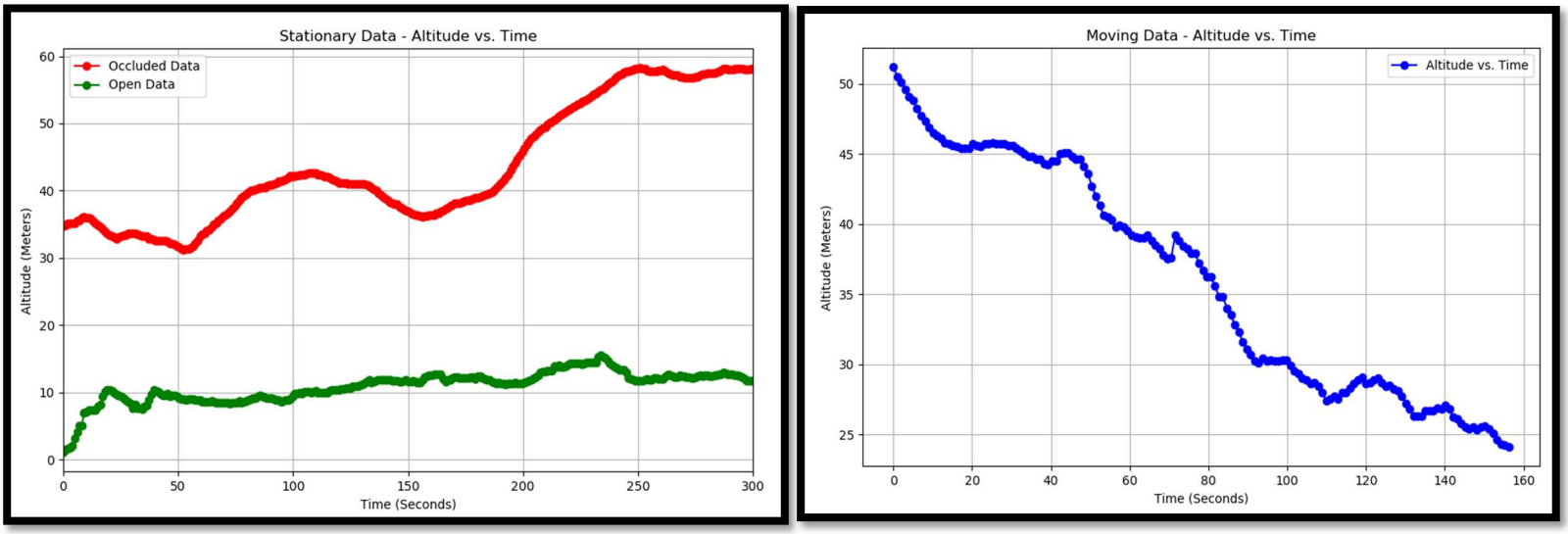
The histograms of Euclidean distances from standalone GPS data reveal distinct patterns between occluded and open conditions. In occluded environments, the data exhibits a wider spread, indicative of larger variability in GPS measurements due to multipath errors and signal blockages. Conversely, in open conditions, the histogram shows a much tighter distribution, suggesting greater consistency and precision in measurements when there is a clear line of sight to satellites. This narrower spread implies fewer errors from atmospheric conditions or obstructions.

In contrast, the histograms derived from RTK GPS data demonstrate a higher level of accuracy and consistency, even in less-than-ideal conditions. The histogram of occluded data still exhibits a relatively tight distribution, albeit not as tight as the open data. This indicates that RTK GPS maintains precision and accuracy even amidst obstructions, likely due to its correction signals. In open conditions, the RTK GPS histogram displays an extremely narrow spread, reflecting a high degree of precision and accuracy facilitated by ground-based reference stations providing corrections.

Comparing standalone and RTK GPS histograms underscores the superior accuracy and consistency of RTK GPS systems, particularly evident in occluded environments. RTK GPS systems offer a preferable choice for applications demanding high precision, such as surveying or precision agriculture. Conversely, standalone GPS may suffice for general navigation purposes, especially in open areas with good satellite visibility.

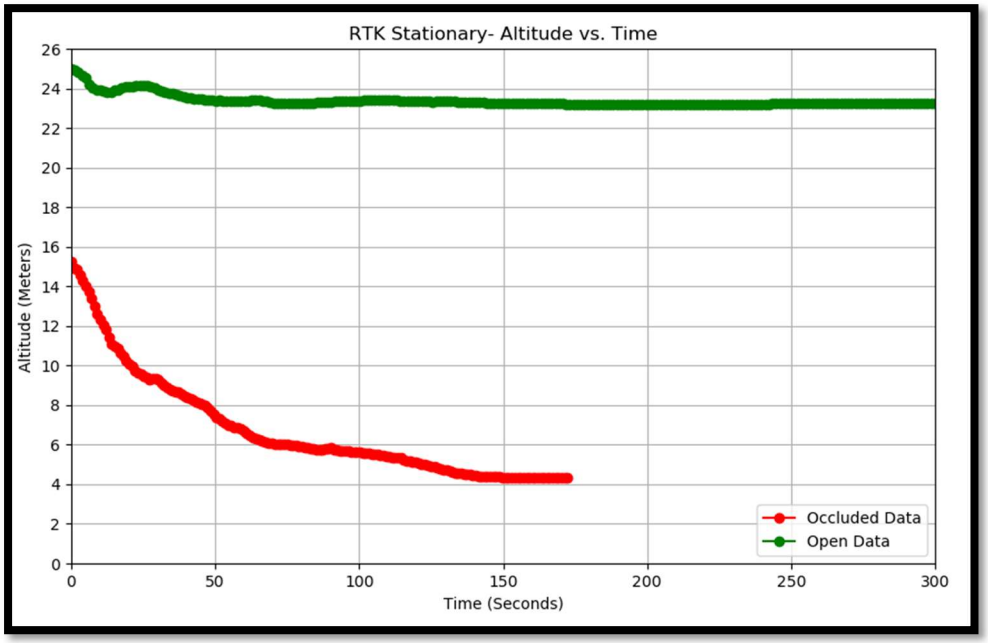
Section G: Altitude vs Time Plots and Analysis

Analyzing altitude over time provides insights into the accuracy of GPS measurements in tracking varying altitudes. However, during data recording, both Standalone and RTK drivers initially set altitude values to 0 when converting to CSV or ods files, resulting in erroneous flat lines in altitude vs. time plots. To address this issue, identification of GPGGA strings from Standalone drivers and GNGGA strings from RTK drivers in the CSV files was crucial. Altitude, typically represented as the 10th term in these strings, prompted the development of an algorithm capable of extracting the specific altitude value from each GPGGA or GNGGA string and integrating it into altitude values for accurate plotting [8-11].

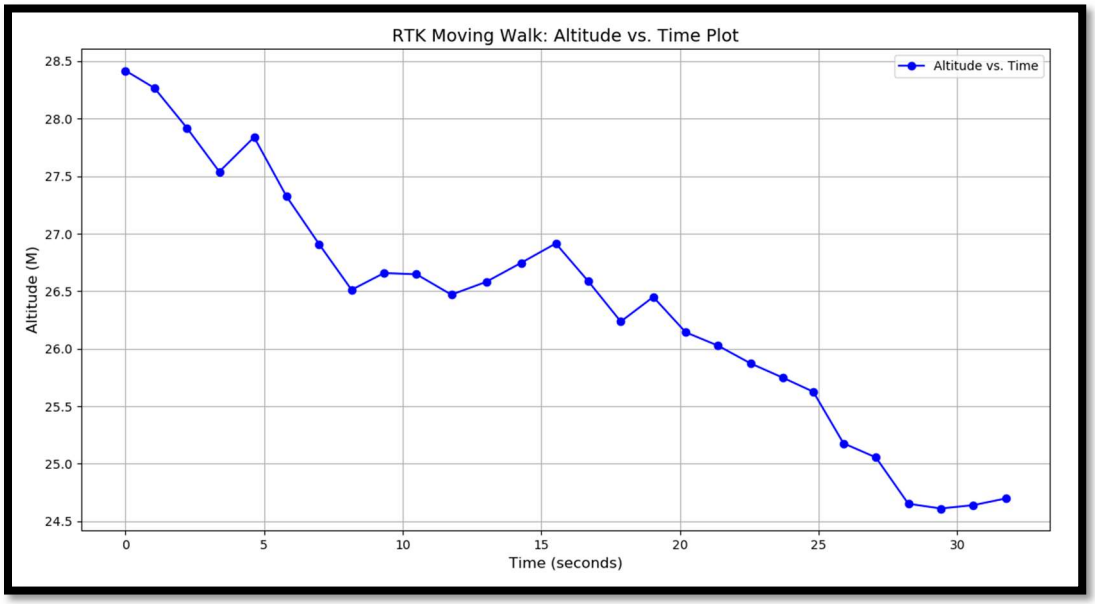


(a) (b)

Fig.7. Altitude vs Time Plots from Standalone Driver Data where (a) represents stationary occluded and open area and (b) represents moving data



(a)



(b)

Fig.8. Altitude vs Time Plots from RTK Driver Data where (a) represents stationary occluded and open area and (b) represents moving data.

The altitude readings from both standalone and RTK GPS systems reveal distinct patterns in stationary and moving conditions. In the stationary setting, standalone GPS data exhibit significant variation over time, with noticeable drifts in altitude readings particularly evident in occluded

conditions. Conversely, RTK GPS measurements show greater stability, with minimal variation observed in both open and occluded conditions. When transitioning to moving conditions, standalone GPS altitude readings display consistent decreases over time, potentially influenced by actual changes in altitude along the path and errors introduced during motion, such as signal loss or multipath effects. Meanwhile, RTK GPS data exhibit less pronounced trends in altitude variation, suggesting more accurate altitude tracking capabilities even while in motion, likely facilitated by the precision of RTK corrections.

Comparing the two systems reveals that RTK GPS consistently provides more stable and accurate altitude readings across both stationary and moving scenarios. In occluded conditions, RTK GPS outperforms standalone GPS by maintaining stability and accuracy, indicative of its capability to correct for common errors associated with obstructed signal reception. Furthermore, in moving conditions, RTK GPS demonstrates superior altitude tracking capabilities, as evidenced by smoother altitude profiles compared to standalone GPS. These observations underscore the advantages of RTK GPS technology in delivering reliable altitude measurements for various applications, particularly in challenging environments and dynamic scenarios.

To calculate the error of altitude vs time plots, the residuals can be computed by subtracting the GPS reading from the reference altitude at that location. The root mean square error for residuals can then be computed for further error analysis. The main focus of the lab is not really to calculate these errors, but the equations provided below are references to how to calculate the errors if necessary.

$$Residual = Observed_{value} - Predicted_{value} \tag{13}$$

$$RMSE_{residual} = \sqrt{\frac{1}{n}\sum(Residual)^2} \tag{14}$$

Conclusion

This report highlights the application and benefits of GNSS technologies, particularly GPS and RTK GNSS systems, in robotic sensing and navigation. Through rigorous experimentation, it demonstrates RTK GNSS's superiority in accuracy and reliability and underscores the importance of precision in navigation. The study also explores pseudo-ranging techniques, emphasizing RTK's potential to enhance data accuracy in challenging environments. Future iterations could integrate GNSS with other localization technologies to improve reliability in obstructed settings.

Moving forward, it's crucial to address GNSS system vulnerabilities, such as interference, through comprehensive error correction strategies and refined RTK integration. This would maximize RTK's potential for precise positioning across various applications, from urban autonomous vehicles to detailed geospatial data collection. Addressing these areas will ensure continued advancement and application of GNSS technologies in complex navigation scenarios.

References:

[1] L. A. Rapoport, “GPS Satellite Surveying ,” Online Library Wiley, https://www.researchgate.net/publication/282752562_http_onlinelibrarywileycom_doi_101002_eji1830270613_epdf (accessed Feb. 12, 2024).

[2] J. Zidan, C. R. Ford, M. D. Higgins, S. A. Birrell, and E. Kampert, “(PDF) GNSS vulnerabilities and existing solutions: A review of ...,” ResearchGate, https://www.researchgate.net/publication/339248814_GNSS_Vulnerabilities_and_Existing_Solutions_A_Review_of_the_Literature (accessed Feb. 12, 2024).

[3] “Pythonic data cleaning with pandas and NumPy,” Real Python, https://realpython.com/python-data-cleaning-numpy-pandas/ (accessed Feb. 11, 2024).

[4] “Chapter 4: GNSS error sources,” NovAtel, https://novatel.com/an-introduction-to-gnss/gnss-error-sources (accessed Feb. 11, 2024).

[5] “How to calculate slope and intercept error of linear regression,” Saturn Cloud Blog, https://saturncloud.io/blog/how-to-calculate-slope-and-intercept-error-of-linear-regression/#:~:text=The%20standard%20error%20of%20the,X%20for%20the%20ith%20observation (accessed Feb. 11, 2024).

[6] A. Hunegnaw et al., “On the impact of GPS multipath correction maps and post-fit residuals on slant wet delays for tracking severe weather events,” MDPI, https://www.mdpi.com/2073-4433/14/2/219 (accessed Feb. 11, 2024).

[7] P. Misra and P. Enge, Global Positioning System: Signals, measurements, and performance ..., https://ieeexplore.ieee.org/document/1044515/ (accessed Feb. 12, 2024).

[8] Real Python, “Python exceptions: An introduction,” Real Python, https://realpython.com/python-exceptions/ (accessed Feb. 11, 2024).

[9] “Working with text data,” Working with text data - pandas 2.2.0 documentation, https://pandas.pydata.org/docs/user_guide/text.html (accessed Feb. 11, 2024).

[10] “How to convert OpenDocument spreadsheets to a pandas DataFrame?,” Stack Overflow, https://stackoverflow.com/questions/17834995/how-to-convert-opendocument-spreadsheets-to-a-pandas-dataframe (accessed Feb. 11, 2024).

[11] Csb-Comren, “CSB-comren/Pynmea2: A python parser for NMEA strings. fork of https://github.com/knio/pynmea2 with some extra types and other changes,” GitHub, https://github.com/csb-comren/pynmea2 (accessed Feb. 11, 2024).

All code and data files can be found on GitHub

GitHub Link: https://github.com/sazgar001/EECE5554/tree/main/gnss/src