

CHAPTER 4 – RESULTS AND DISCUSSIONS

This chapter will document the results and analyze the performance test of the end product. The chapter will then move on to discuss possible improvements for the project so that in the future, it can be developed better.

4.1 Testing and Experimental Plans

To achieve the thesis objective, several tasks and tests will be done from testing the sensors used to testing for the actual last-mile delivery application in real-time.

- Linorobot Teensy firmware porting from ROS to ROS2 and reliability check.
- IMU MPU9250 and rotary incremental encoder HE50B calibration and accuracy test.
- RTK-GNSS module reliability test on several occasions.
- LiDAR measured value compared to the actual value.
- RTK-GNSS Latitude and Longitude Conversion to XYZ Coordinates Reliability Check
- Robot actuation using keyboard teleoperation with RTK-GNSS data visualized on Mapviz.
- Robot odometry and RTK-GNSS combined localization using EKF.
- Route planning reliability.
- Robot automated navigation testing on one way and return path.

4.2 Test Results and Discussions

4.2.1 Linorobot Teensy Firmware Porting from ROS to ROS2

As Linorobot is built based on ROS architectures and packages for ROS2 have not been created, porting Linorobot package to ROS2 is required as their architectures are different. Using the Micro-ROS Arduino library, all the required ROS2 libraries can be used directly inside the Linorobot package.

ROS architectures are mostly based on Python language meanwhile ROS2 architectures which prioritize reliability are based on C++ language. Thus, headers that are required to be included so that ROS2 can be used inside Micro-ROS Arduino should be changed into RCLCPP (ROS Client Library C++) which consists of all ROS2 architectures and the comparison of included ROS libraries for ROS and ROS2 can be seen in Figure 4-1. Creating topics in ROS2 is also different compared to ROS. Figure 4-2 shows the comparison in which how ROS and ROS2 create their publisher. Figure 4-3 shows the comparison in which how ROS and ROS2 create their subscriber and how they are assigned to callbacks (a function that triggers on a specific condition). For ROS2, more than one *executor* is needed to execute the callback functions and topics. Figure 4-4 shows the lists of topics created by Linorobot which will be used with other packages in ROS2.

```
//ROS LIBRARIES
#include "ros.h"
#include "ros/time.h"
//header file for publishing velocities for odom
#include "lino_msgs/Velocities.h"
//header file for cmd_subscribing to "cmd_vel"
#include "geometry_msgs/Twist.h"
//header file for pid server
#include "lino_msgs/PID.h"
//header file for imu
#include "lino_msgs/Imu.h"

#include <micro_ros_arduino.h>

#include <stdio.h>
#include <rcl/rcl.h>
#include <rcl/error_handling.h>
#include <rclcpp/rclcpp.h>
#include <rclcpp/executor.h>

#include <std_msgs/msg/int32.h>
#include <std_msgs/msg/float32.h>
#include <geometry_msgs/msg/twist.h> //header file for subscriber vel
#include <geometry_msgs/msg/vector3.h> //header for PID and DR, needs 3 Vector3 data

#include <sensor_msgs/msg/imu.h> //header for IMU
#include <geometry_msgs/msg/quaternion.h> //header for IMU
```

Figure 4-1 ROS (above) and ROS2 (below) Libraries Inclusion

```
nh.advertise<raw_vel_pub>;
nh.advertise<raw_imu_pub>;

lino_msgs::Imu raw_imu_msg;
ros::Publisher raw_imu_pub("raw_imu", &raw_imu_msg);

lino_msgs::Velocities raw_vel_msg;
ros::Publisher raw_vel_pub("raw_vel", &raw_vel_msg);

rcl_publisher_t publisher_vel; //Geometry Twist
rcl_publisher_t pub_gyro;
rcl_publisher_t pub_accel;
rcl_publisher_t pub_magnet;

RCHECK(rcl_publisher_init_default(
    &pub_gyro,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(geometry_msgs, msg, Vector3),
    "raw_imu_gyro"));

RCHECK(rcl_publisher_init_default(
    &pub_accel,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(geometry_msgs, msg, Vector3),
    "raw_imu_accel"));

RCHECK(rcl_publisher_init_default(
    &pub_magnet,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(geometry_msgs, msg, Quaternion),
    "raw_imu_magnet"));

RCHECK(rcl_publisher_init_default(
    &publisher_vel,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(geometry_msgs, msg, Twist),
    "raw_vel"));
```

Figure 4-2 ROS (left) and ROS2 (right) Creating Publisher

```

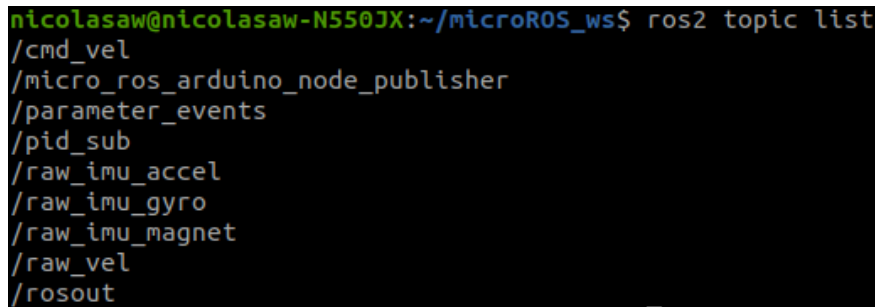
ros::Subscriber<geometry_msgs::Twist> cmd_sub("cmd_vel", commandCallback);
ros::Subscriber<lino_msgs::PID> pid_sub("pid", PIDCallback);

nh.subscribe(pid_sub);
nh.subscribe(cmd_sub);

RCCHECK(rcl_executor_add_subscription(&executor_vel, &subscriber_vel, &msg, &sub_vel_callback, ON_NEW_DATA));
RCCHECK(rcl_executor_add_subscription(&executor_pid, &subscriber_pid, &msg, &sub_pid_callback, ON_NEW_DATA));
rcl_subscription_t subscriber_vel; //Geometry Twist
rcl_subscription_t subscriber_pid; //Geometry Vector3

```

Figure 4-3 ROS (two above) and ROS2 (two below) Creating Subscriber



```

nicolasaw@nicolasaw-N550JX:~/microROS_ws$ ros2 topic list
/cmd_vel
/micro_ros_arduino_node_publisher
/parameter_events
/pid_sub
/raw_imu_accel
/raw_imu_gyro
/raw_imu_magnet
/raw_vel
/rosout

```

Figure 4-4 Linorobot Topic List

Subscribing or publishing a topic in ROS2 has a different method compared to ROS. Figure 4-5 shows the comparison in which how ROS and ROS2 subscribe to topics whenever new messages are published in the topic. Figure 4-6 shows the comparison in which how ROS and ROS2 publish values to topics. Figure 4-7 and Figure 4-8 show the program is run and the rqt_graph in which how Micro-ROS communicates with topics created from Linorobot and connects with ROS2 in PC respectively.

```

void sub_vel_callback(const geometry_msgs::Twist& cmd_msg)
{
    //callback function every time linear and angular speed is received from 'cmd_vel' topic
    //this callback function receives cmd_msg object where linear and angular speed are stored
    g_req_linear_vel_x = cmd_msg.linear.x;
    g_req_linear_vel_y = cmd_msg.linear.y;
    g_req_angular_vel_z = cmd_msg.angular.z;

    g_prev_command_time = millis();
}

void sub_vel_callback(const void * msgin)
{
    const geometry_msgs__msg__Twist * msg = (const geometry_msgs__msg__Twist *)msgin;
    //digitalWrite(LED_PIN, (msg->data == 0) ? LOW : HIGH);

    //callback function every time linear and angular speed is received from 'cmd_vel' topic
    //this callback function receives cmd_msg object where linear and angular speed are stored
    g_req_linear_vel_x = msg->linear.x;
    g_req_linear_vel_y = msg->linear.y;
    g_req_angular_vel_z = msg->angular.z;

    g_prev_command_time = millis();
}

```

Figure 4-5 ROS (above) and ROS2 (below) Subscribing to Topic with Callbacks

```

void publishIMU()
{
    //pass accelerometer data to imu object
    raw_imu_msg.linear_acceleration = readAccelerometer();

    //pass gyroscope data to imu object
    raw_imu_msg.angular_velocity = readGyroscope();

    //pass accelerometer data to imu object
    raw_imu_msg.magnetic_field = readMagnetometer();

    //publish raw_imu_msg
    raw_imu_pub.publish(&raw_imu_msg);
}

void publishIMU()
{
    //pass accelerometer data to imu object
    msgVect3_accel = readAccelerometer();

    //pass gyroscope data to imu object
    msgVect3_gyro = readGyroscope();

    //pass accelerometer data to imu object
    msgVect3_magnet = readMagnetometer();

    RCSOFTCHECK(rc1_publish(&pub_accel, &msgVect3_accel, NULL));
    RCSOFTCHECK(rc1_publish(&pub_gyro, &msgVect3_gyro, NULL));
    RCSOFTCHECK(rc1_publish(&pub_magnet, &msgVect3_magnet, NULL));
}

```

Figure 4-6 ROS (above) and ROS2 (below) Publishing to Topics

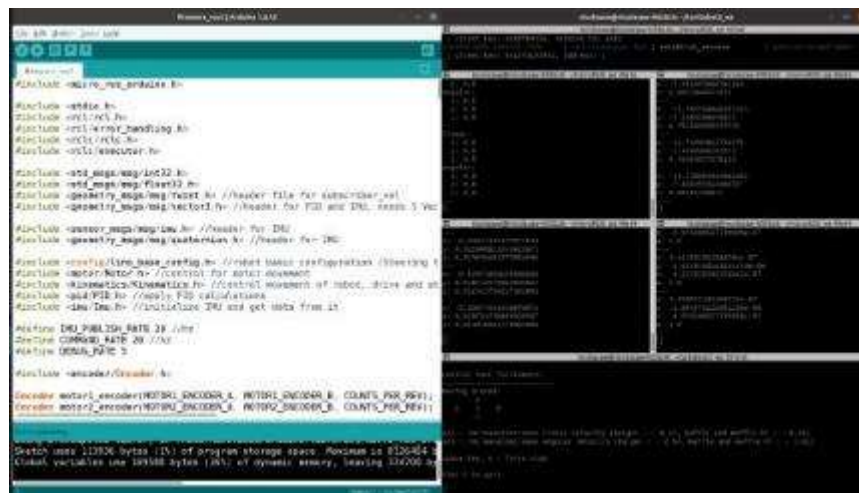


Figure 4-7 Executing Linorobot Firmware

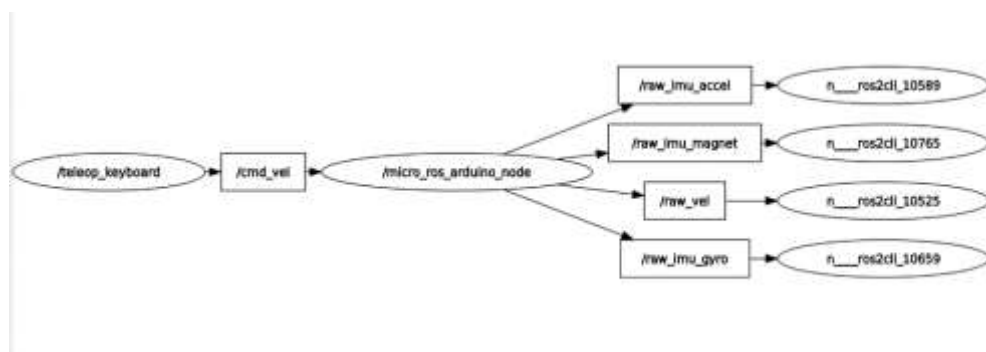


Figure 4-8 RQT Graph of Linorobot Software

Linorobot Teensy firmware has libraries for robot configuration, sensors needed, kinematic calculation, PID calculation, and motor controlling which can be seen included in Figure 4-9. In *lino_base_config.h*, robot basic configurations can be set such as robot base, motor driver, IMU used, PID values, maximum wheel RPM, rotary encoder resolution, wheel diameter, robot dimensions, and pin configurations. In *imu.h*, it will read the value of IMU readings such as accelerometer, gyrometer, and magnetometer values. In *encoder.h*, it will count the number of pulses in a certain time and return the RPM of the motor. In *kinematics.h*, it will calculate the required RPM needed using robot kinematic after Linorobot subscribed */cmd_vel* topic which consists of required velocity. It can also return the actual velocity of the robot using readings of a rotary encoder. In *pid.h*, it will calculate the required PWM for the motor to reach the required RPM value. Lastly, *motor.h* will give PWM output to the controller to drive the motor.

```
#include <config/lino_base_config.h> //robot basic configuration
//(Steering type, Motor driver, IMU, PID, Robot specs, Pins assignment)
#include <motor/Motor.h> //control for motor movement
#include <kinematics/Kinematics.h> //control movement of robot, drive and steering.
//Sets velocity and acceleration
#include <pid/PID.h> //apply PID calculations
#include <imu/Imu.h> //initialize IMU and get data from it
```

Figure 4-9 Header Libraries for Linorobot Firmware

4.2.2 Linorobot ROS Applied on ROS2

Linorobot in ROS is a robot system used for indoor applications. It has a complete system from the microcontroller side using Linorobot firmware as well as on the PC side which consists of codes from sensing to navigation. For sensing, it has libraries for IMU calibrations, EKF localization, LiDAR packages, amcl (adaptive Monte Carlo localization) to locate robot position using laser scans in respect to the map, 3D camera packages as well as SLAM (Simultaneous Localization and Mapping) using LiDAR. For navigation, it has *move_base* which will make the route plan of the robot and actuate to reach the target. In total, Figure 4-10 will show the completed *rqt_graph* of the Linorobot system in ROS.

[illegible]

Figure 4-11 Linorobot Firmware Used in ROS2

4.2.3 IMU MPU9250 Calibration Test

IMU calibration is important to be able to obtain an accurate measurement from the IMU. Without an IMU calibration, the device will experience drifting in a certain direction as it does not know the zero position of the IMU which is at yaw, roll, and pitch at zero degrees. By doing an IMU calibration, offsets and biases of the zero position of IMU can be measured as well as it eliminates magnetic disturbance for the magnetometer.

From the previous research that uses Linorobot software, IMU calibration test can be done using package under Linorobot software which is *imu_calib* node. For this project, IMU calibrations will not be done instead calibrations done by Wilvan Gunawan Wirjono, SGU Mechatronics Batch 2016 will be referred which uses *imu_calib* node from Linorobot software.

Wilvan mounted the IMU on the robot before calibration as the process requires the robot to be rotated in the X, Y, and Z-axis which will record the yaw, pitch, and roll of the robot. Figure 4-12 and Figure 4-13 show the process of the calibration and the results respectively. After the calibration had been completed, the magnetometer of the IMU will be tested by comparing the robot angle difference with the readings from the magnetometer which results are shown in Table 4.1. Based on the results, the MPU9250 has a measurement error of 0.01733% with a reliability of 99.98267%. (Wirjono, 2020)



```
wilvan@wilvan-GF75-THICC:~/thesis_ws/src/linorobot/param/imu$ roslaunch imu_calib do_calib
Orient IMU with X+ axis - Front side of the robot facing up. Press [ENTER] once done.
Calibrating! This may take a while....
Done.
Orient IMU with X- axis - Rear side of the robot facing up. Press [ENTER] once done.
Calibrating! This may take a while....
Done.
Orient IMU with Y+ axis - Left side of the robot facing up. Press [ENTER] once done.
Calibrating! This may take a while....
Done.
Orient IMU with Y- axis - Right side of the robot facing up. Press [ENTER] once done.
Calibrating! This may take a while....
Done.
Orient IMU with Z+ axis - Top side of the robot facing up. Press [ENTER] once done.
Calibrating! This may take a while....
Done.
Orient IMU with Z- axis - Bottom side of the robot facing up. Press [ENTER] once done.
Calibrating! This may take a while....
Done.
Computing calibration parameters... Success!
Saving calibration file... Success!
```

Figure 4-12 IMU Calibration Process

```

bias:
- 0.3820744757146852
- 0.1850884103215002
- -0.559582312278731
SM:
- 0.9992554908830635
- -0.04258160463142015
- 0.006785431529667215
- 0.0210131196096866
- 1.000808445249334
- -0.003439003042586958
- -0.02348692826552792
- 0.01450584173198626
- 0.9815301549248474
    
```

Figure 4-13 IMU Calibration Results

Real Angle (°)	Measured Angle (°)
0	-0.012
5	5.04
10	9.86
15	14.98
20	20.11
25	24.97
30	30.01

Table 4-1MPU9250 Magnetometer Testing Results

4.2.4 Rotary Incremental Encoder HE50B Accuracy Test

A rotary incremental encoder needs to be run on an accuracy test to determine how high of accuracy can the output be. In this test, the angular velocity of both encoders will be recorded using the program in Linorobot software and compared to the output from a tachometer. For this test, the robot will run with no load as the robot will be on a higher level platform in which the wheels do not touch the ground as seen in Figure 4-14. Measured value from the encoder will be recorded by running the Linorobot firmware as seen in Figure 4-15.



Figure 4-14 Encoder Test Scenario

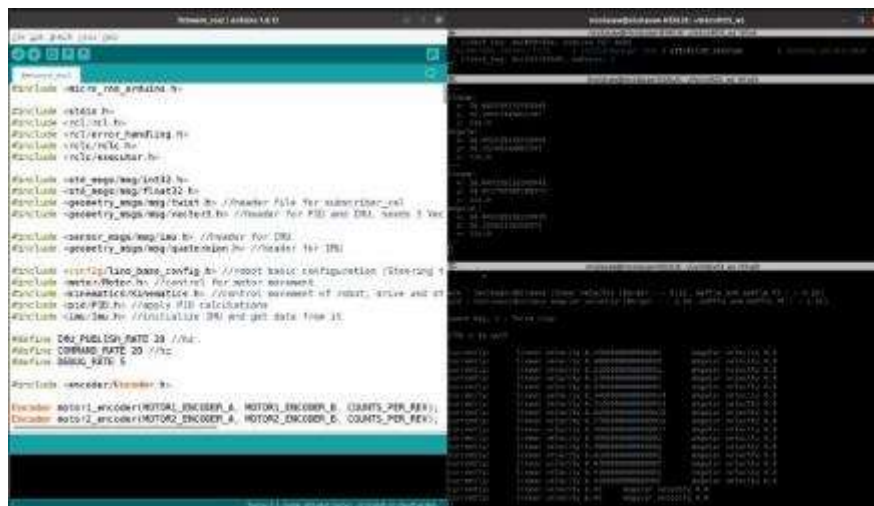


Figure 4-15 Linorobot Firmware Enncoder Data Taking

The test will record several required angular velocities for the system with the PWM on each motor. Table 4.2 shows the result of the data taking. From the results show, there are slight differences in the PWM on the motors with the right motor always slightly higher than the left motor. This could be caused by the difference in the motor itself which causes the right motor to need more PWM to reach the same required angular velocity. There is a slight difference between the set value from the program and encoder reading which the error average for the right encoder is at 0.52% and for the left encoder, the average difference is 0.64 is at 1.29%. Thus, from these results, it is concluded that the encoder has a high accuracy output despite the slight difference

and reliability to be used in this project. The graph comparison can be seen in Figure 4-16.

Required Value (RPM)	Tachometer Left (RPM)	Encoder Left (RPM)	Error (Percentage)	Required Value (RPM)	Tachometer Right (RPM)	Encoder Right (RPM)	Error (Percentage)
26.04	25	25.8	0.94	26.04	25.3	25.9	0.55
24.31	23.3	24.1	0.86	24.31	23.6	24.3	0.04
22.57	21.6	22.2	1.64	22.57	22	22.5	0.31
20.84	20	20.7	0.65	20.84	20.2	20.8	0.17
19.10	17.3	18.8	1.57	19.10	18.4	19	0.52
17.36	16.8	17.2	0.93	17.36	16.9	17.3	0.36
15.63	14.8	15.5	0.81	15.63	15.2	15.6	0.17
13.89	13.3	13.6	2.09	13.89	13.4	13.8	0.65
12.15	11.7	12	1.27	12.15	11.8	12	1.27
10.42	9.9	10.2	2.11	10.42	10	10.3	1.15

Table 4-2 Encoder Angular Velocity Test

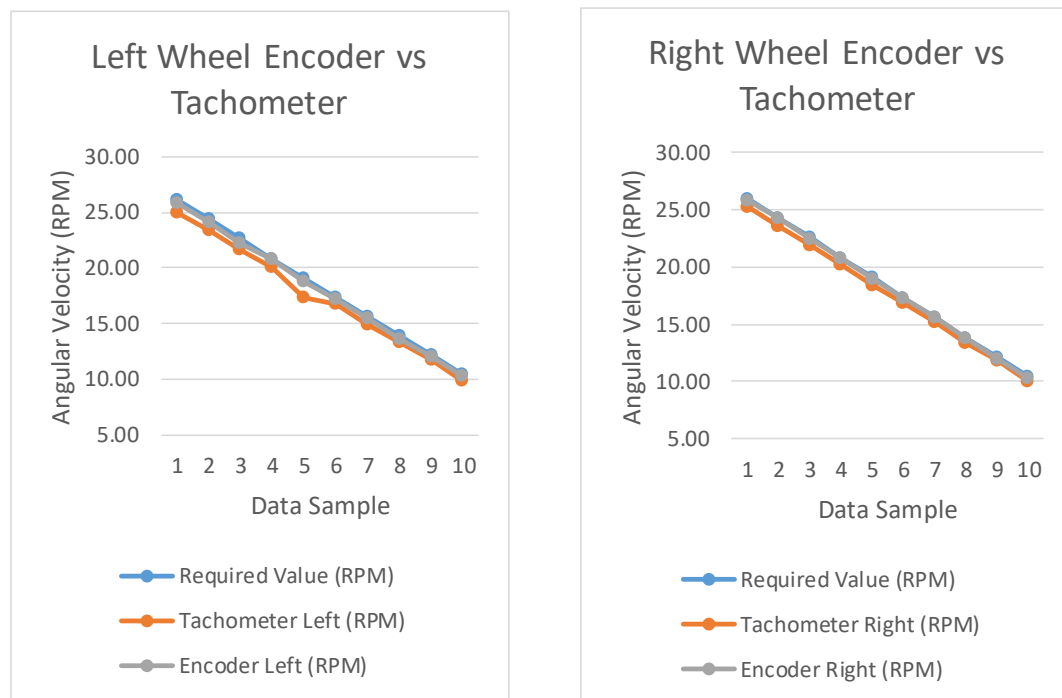


Figure 4-16 RPM Comparison on Tachometer vs Encoder Value

4.2.5 Linorobot Firmware Testing on Robot

For this test, it will be more like a quality check of the Linorobot firmware whether it works fine from the encoder RPM output, IMU readings, ROS2 topic publish and subscribes, kinematic calculation, PWM calculation with PID as well as actuation onto the motor. The firmware will be tested on the robot indoors using keyboard teleoperation by Turtlebot3 as the firmware requires `/cmd_vel` topic to subscribe so that it can calculate the required kinematics and PWM that the robot needs to be actuated in. Figure 4-X shows MicroROS connects the Teensy with ROS2 architectures, ROS2

subscribes the angular velocity of the wheels as well as Turtlebot3 Keyboard Teleoperation publishing `/cmd_vel` topic to the firmware. Figure 4-17 shows how the robot is tested. The robot can actuate smoothly indoors however when it rotates on its axis, the front wheels tend to skid while rotating.



Figure 4-17 Robot Keyboard Teleoperation Test

4.2.6 RTK-GNSS Reliability Test

As the GPS-RTK2 module is used in this thesis, the module needs to be run on several performance tests to ensure its reliability in receiving geospatial position. The tests are:

- Does the rover receive RTCM corrections from the base station if the base is not in RTK Fixed mode?
- Do smaller required position accuracy deviations reduce outliers in the rover's geo-spatial position readings?
- Does weather affect the rover's geo-spatial position readings?
- Does Survey-In of 1 hour of the base station is still accurate for a longer period of rover's geospatial position readings?
- How reliable are the RTCM corrections from the base station to the rover?

The test will be done in Taman Palem Lestari D10 No. 58, Cengkareng, DKI Jakarta 11730, Indonesia with the placement of the antenna of the base station and rover as

seen in Figure 4-18 and Figure 4-19 respectively and the modules inside the house as seen in Figure 4-20.



Figure 4-18 Position of Base Station Antenna for Testing A until D



Figure 4-19 Position of Rover Antenna for Testing A until D



Figure 4-20 Base Station and Rover Module Placements

A) Base Station Fails to Enter RTK Fixed Mode

For this test, the base station will be set with a short observation time of five minutes with a required position accuracy of 0.5 meters as seen in Figure 4-21 so that it would not reach RTK Fixed mode as seen in Figure 4-22. The base station has exceeded the designated observation time at five minutes and the module has not reached the required position accuracy which is still at 0.8127 meters thus not reaching RTK Fixed mode. When the rover is started, it is not able to receive RTCM corrections from the base station as it is not able to enter 3D RTK mode with a position accuracy of still two meters as seen in Figure 4-23. If the rover can reach 3D RTK mode, it will have an accuracy of five centimeters. On the module itself, there is an RTK LED that will show whether it has received RTCM corrections and transitioned to RTK mode. As the RTK LED is wired to the ground, it will be on when it is not in RTK mode. Once the module reached under five centimeters accuracy, the RTK LED will blink and enters RTK Float mode. Once it reaches 14 millimeters accuracy, it will turn off and enters RTK Fix mode.

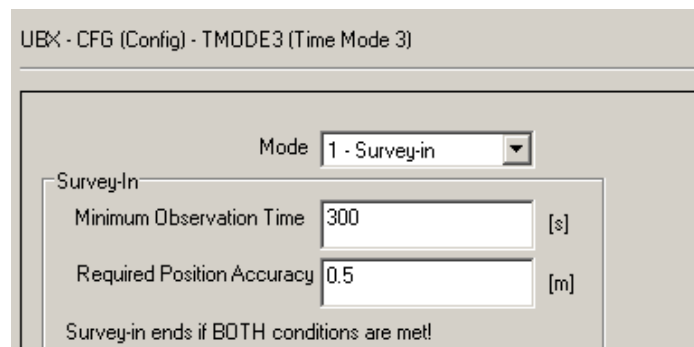


Figure 4-21 Base Station Setup 5 Minutes with 0.5 Meter Accuracy

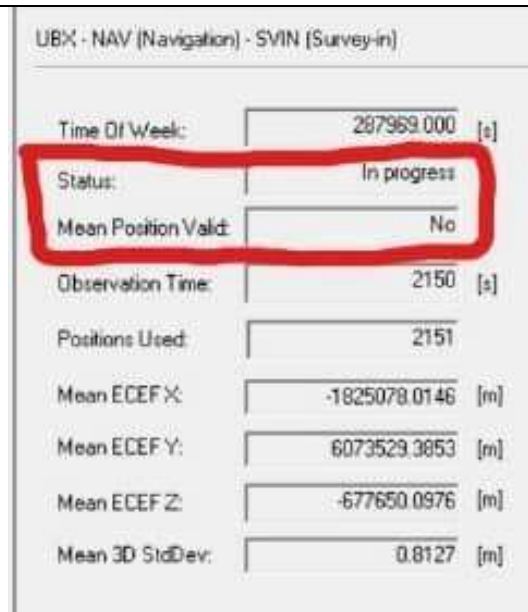


Figure 4-22 Base Station Does Not Reach RTK Fixed Mode

UBX - NAV (Navigation) - PVT (Navigation PVT Solution)		
Param	Value	Units
GPS Time Tag	287978.000	[s]
UTC Date and Time	12/ 5/ 2021 07:59:20 -000101796	
UTC Date and Time Confirmation Status	Date: CONFIRMED, Time: CONFIRM...	
UTC Time Accuracy	1008	[ns]
Position Fix Type	3D Fix	
Fix Flags	FixOK	
PSM state	n/a	
Position Latitude, Longitude, Height, MSL	-6.1398875, 106.7253670, 21.0, 19.3	[deg,de
Invalid Position Latitude, Longitude, Height, MSL	No	
Position Accuracy Estimate Horizontal, Vertical	2.0, 2.7	[m,m]
Velocity North, East, Down	-0.035, 0.011, -0.044	[m/s,m,
Velocity, Heading Accuracy Estimate	0.294, 180.0	[m/s,de
Speed over Ground	0.037	[m/s]
Heading of Motion, Heading of Vehicle	0.0, n/a	[deg,de
Magnetic Declination, Declination Accuracy Estim...	n/a, n/a	[deg,de
PDOP	2.62	
#SVs Used	16	
Carrier Range Status	Not used	

Figure 4-23 Rover Does Not Receive RTCM Corrections

B) Required Position Accuracy Comparison for Base Station

In this test, the base station will be set with two different required position accuracy which is at five meters and one meter. Before the test is started, it is required to check whether the base station can enter the required position accuracy in the determined observation time. Figure 4-24 shows the Survey-In progress to set up base station both at five meters and also one meter which shows the minimum time to reach one-meter accuracy. From the results, it is determined that the suitable and preferable observation time is at one hour.

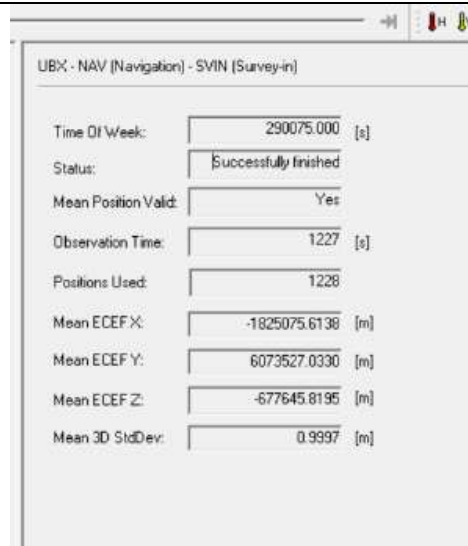


Figure 4-24 Minimum Time to Reach 1 Meter Accuracy for Base Station

There will be two sets of one-hour data, one set will be for five-meter required position accuracy and the other will be for one meter. This test will determine whether bigger required position accuracy will produce my outliers (data that are significantly different from other observations). Figure 4-25 shows the data taken for five-meter required position accuracy and Figure 4-26 shows the data taken for one-meter required position accuracy.

From these two sets of data, it can be seen from the left graph which consists of latitude and longitude in respect to time, at the start of the data taking, there are many outliers at the beginning of the data taking. For the five-meter sets of data, the outliers are far off which reaches up to three to five meters meanwhile the one-meter sets of data despite having outliers, they are less off compared to the five meters reading which is mostly around the one-meter range. For the position accuracy estimate, as can be seen in Table 4-3, both sets of data do not have that much of a difference with the five-meter set has an average of 3.22 centimeters and the one-meter set has an average of 2.77 centimeters. From this test, it can be concluded that having smaller required position accuracy will not eliminate outliers at the beginning of data taking but will significantly reduce how far the outliers are. It is proven true as required position accuracy determines the accepted value readings of the estimated position which means if it is bigger than the required

position accuracy, the data will not be accepted. Although it does reduce outliers, having smaller required position accuracy does not necessarily increase the position accuracy estimate.

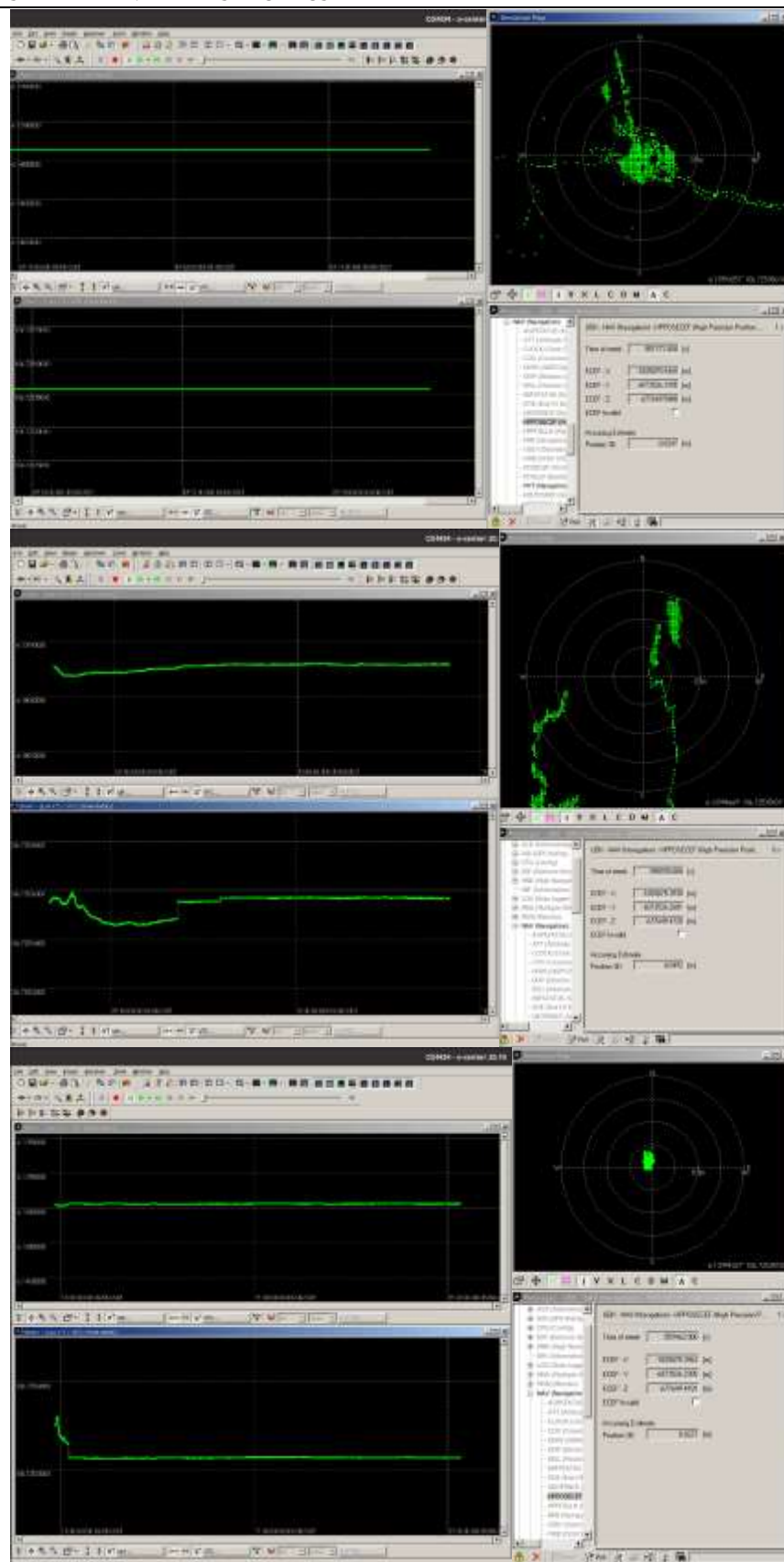


Figure 4-25 Five Meters Data for Outliers Test (RTK-GNSS)

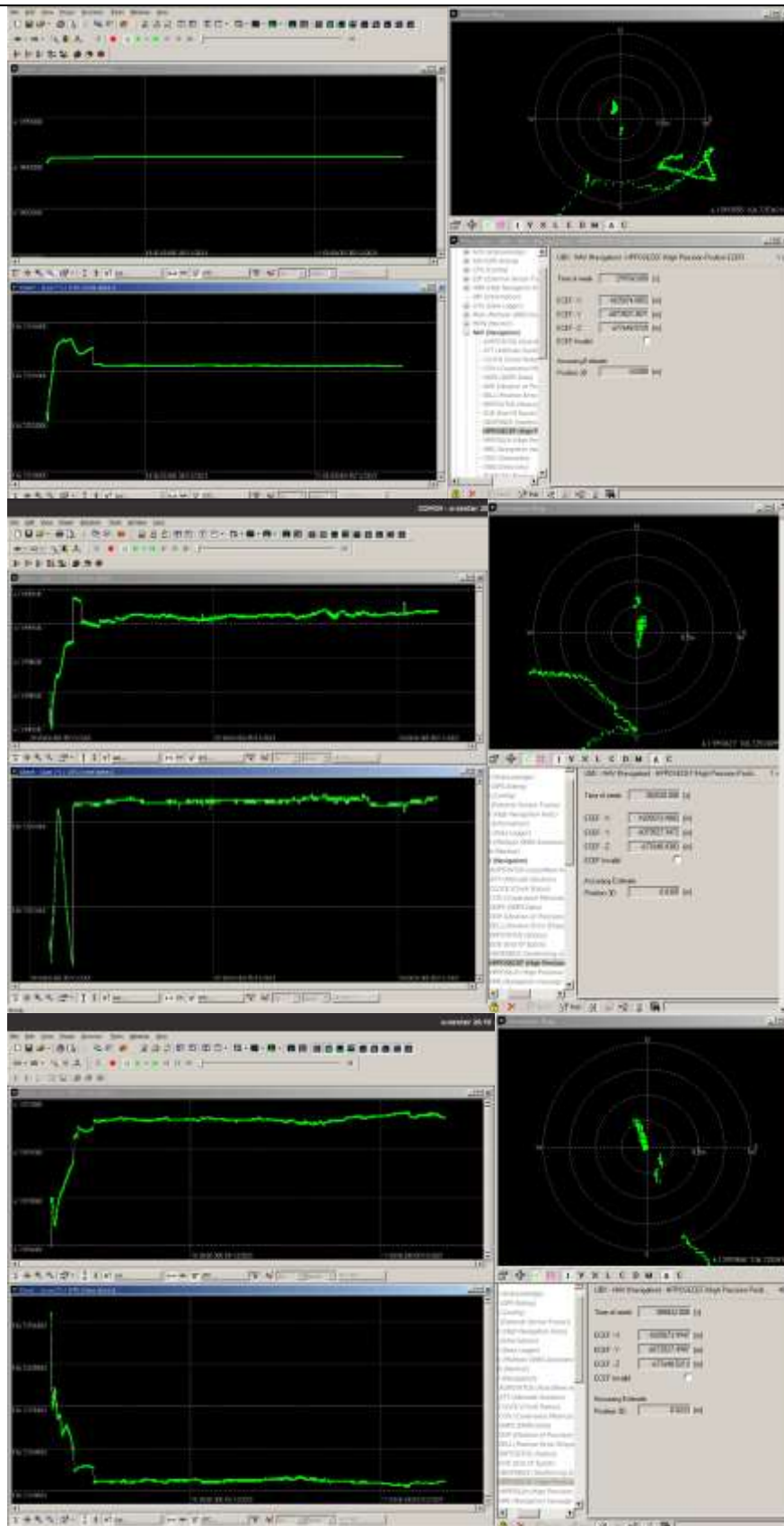


Figure 4-26 One Meter Data for Outliers Test (RTK-GNSS)

Lower Number = Higher Accuracy	Errors of Rover Position with Different Base Station Settings Expected Accuracy	
	5 Meter Required Accuracy (cm)	1 Meter Required Accuracy (cm)
	2.47	3.05
	4.92	3.05
	2.27	2.23
AVERAGE	3.22	2.78

Table 4-3 5 Meter and 1 Meter Position Accuracy Estimate on Base Station Settings Comparison

C) Position Accuracy Estimate on Different Weathers

In this test, there will be three kinds of weather (sunny, cloudy, and rainy) with an additional night time to be tested with each will be having three data, The goal of this test is to compare whether the different weathers affect the position accuracy estimate in one hour of data taking. For the base station, the settings are on one hour observation time and one meter required position accuracy. Both the base station and the rover will be in the same weather condition for each data taking.

Figure 4-27 shows the data taken when the weather is sunny, Figure 4-28 shows the data taken when the weather is cloudy but with light clouds, Figure 4-29 shows the data taken when it is raining with light and heavy rain in transitions, and lastly Figure 4-30 shows the data taken when it is night time. Table 4-4 shows the position accuracy estimate of each data set. From the data taken in sunny weather, it has an average position accuracy estimate at 2.57 centimeters. Meanwhile, for cloudy weather, it has an average position accuracy estimate at 2.77 centimeters. On rainy weather, it has an average position accuracy estimate at 4 centimeters. Last but not least, at night time, it has an average position accuracy estimate at 2.81 centimeters.

From this thesis, we can conclude that despite the weather difference, the position accuracy estimate received by the rover has not much difference ranging from 2-5 centimeters with a total average of 3.04 centimeters. Despite the outliers on sunny weather data, it does not affect the position accuracy estimate that much as the latitude and longitude graphs are in a small grid on the Y-axis.

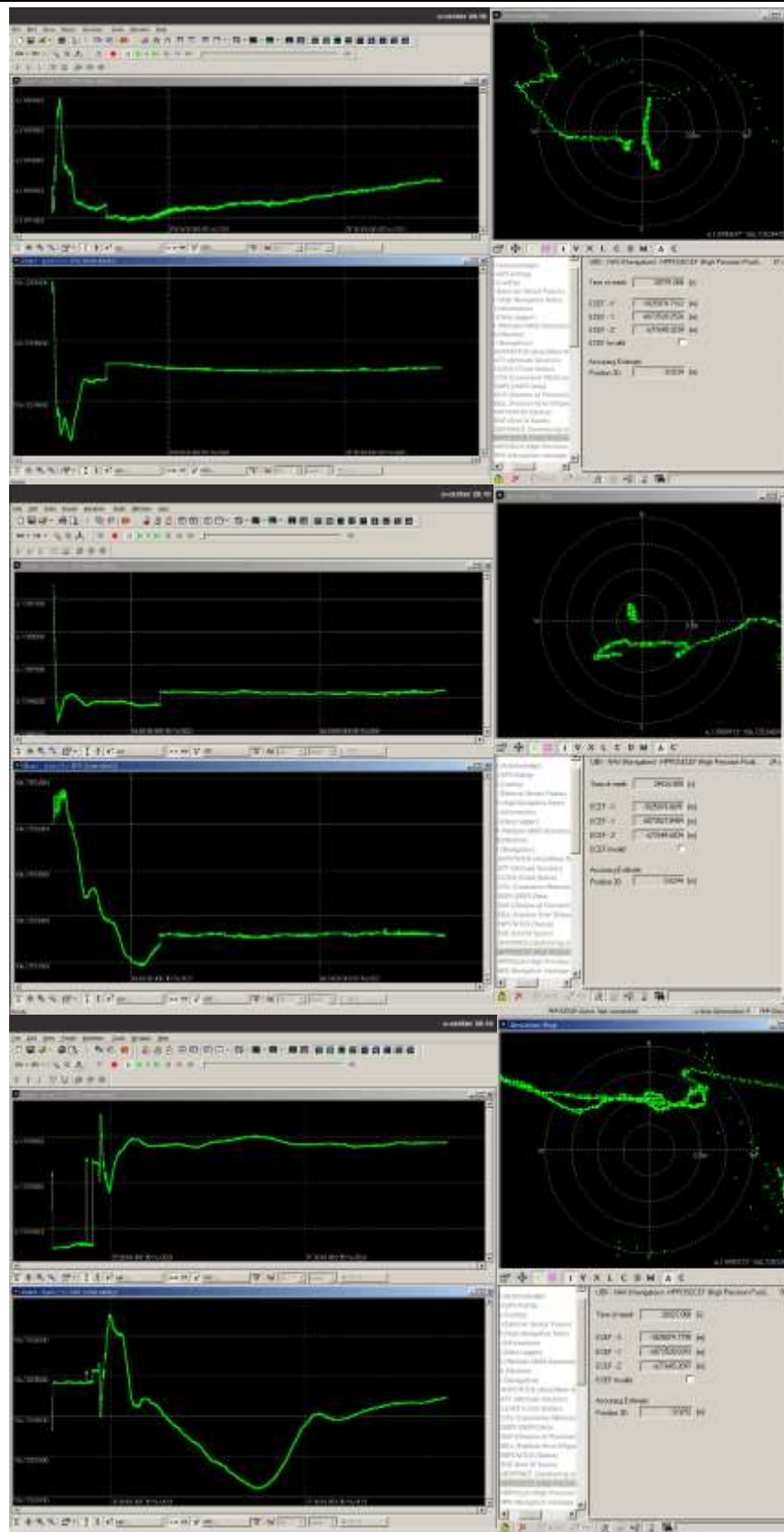


Figure 4-27 Data Taken on Sunny Weather (RTK-GNSS)

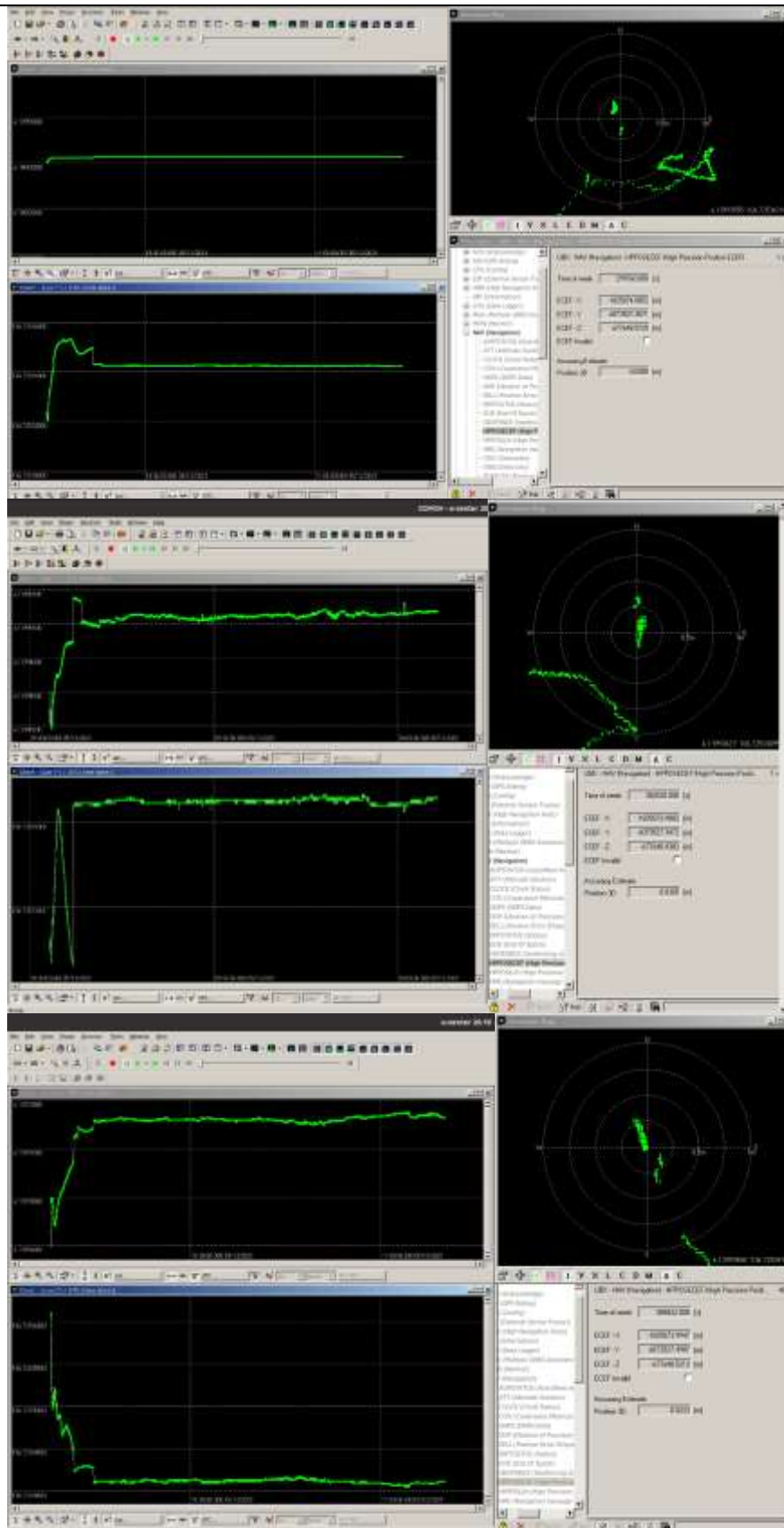


Figure 4-28 Data Taken on Cloudy Weather (RTK-GNSS)

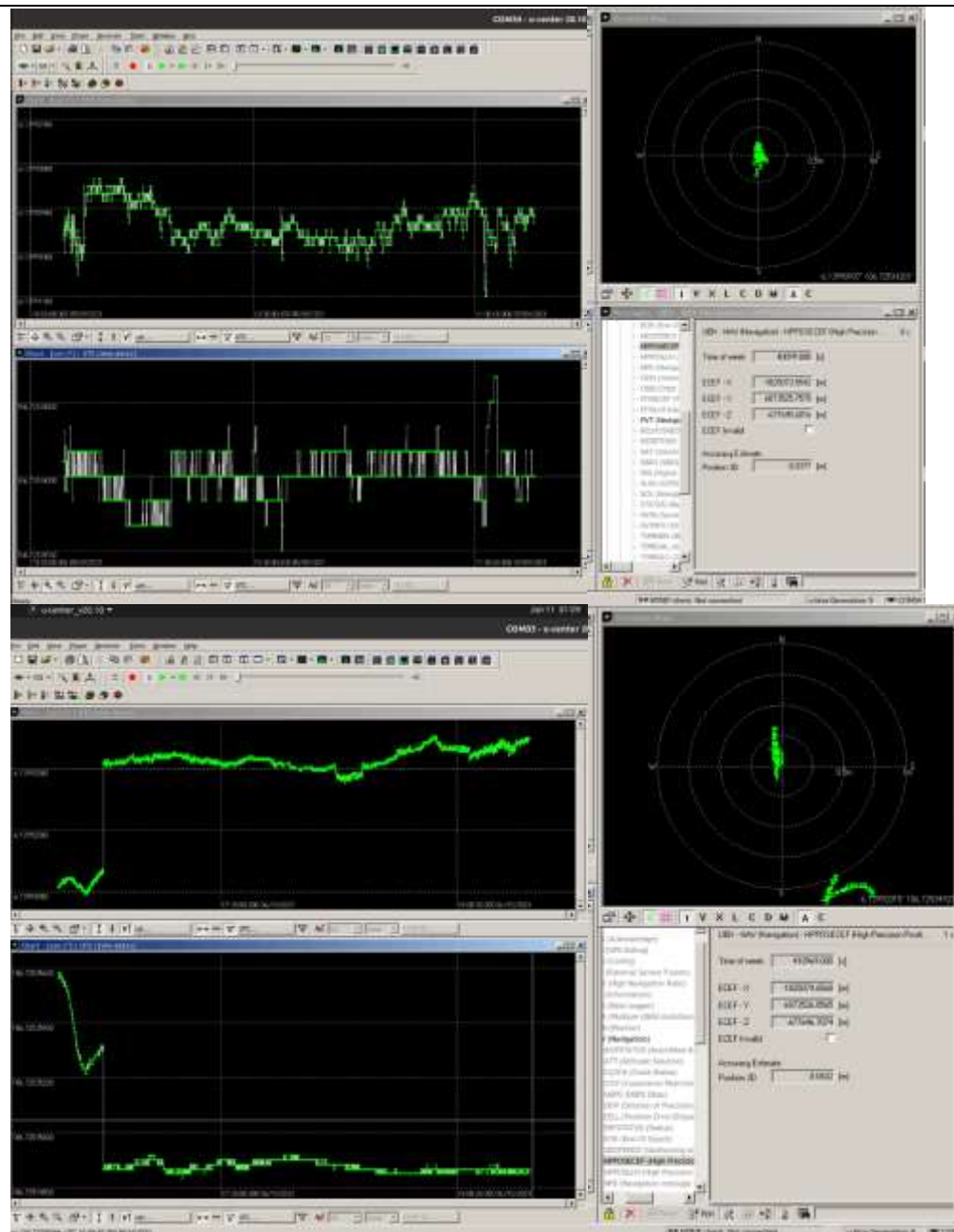


Figure 4-29 Data Taken on Rainy Weather (RTK-GNSS)

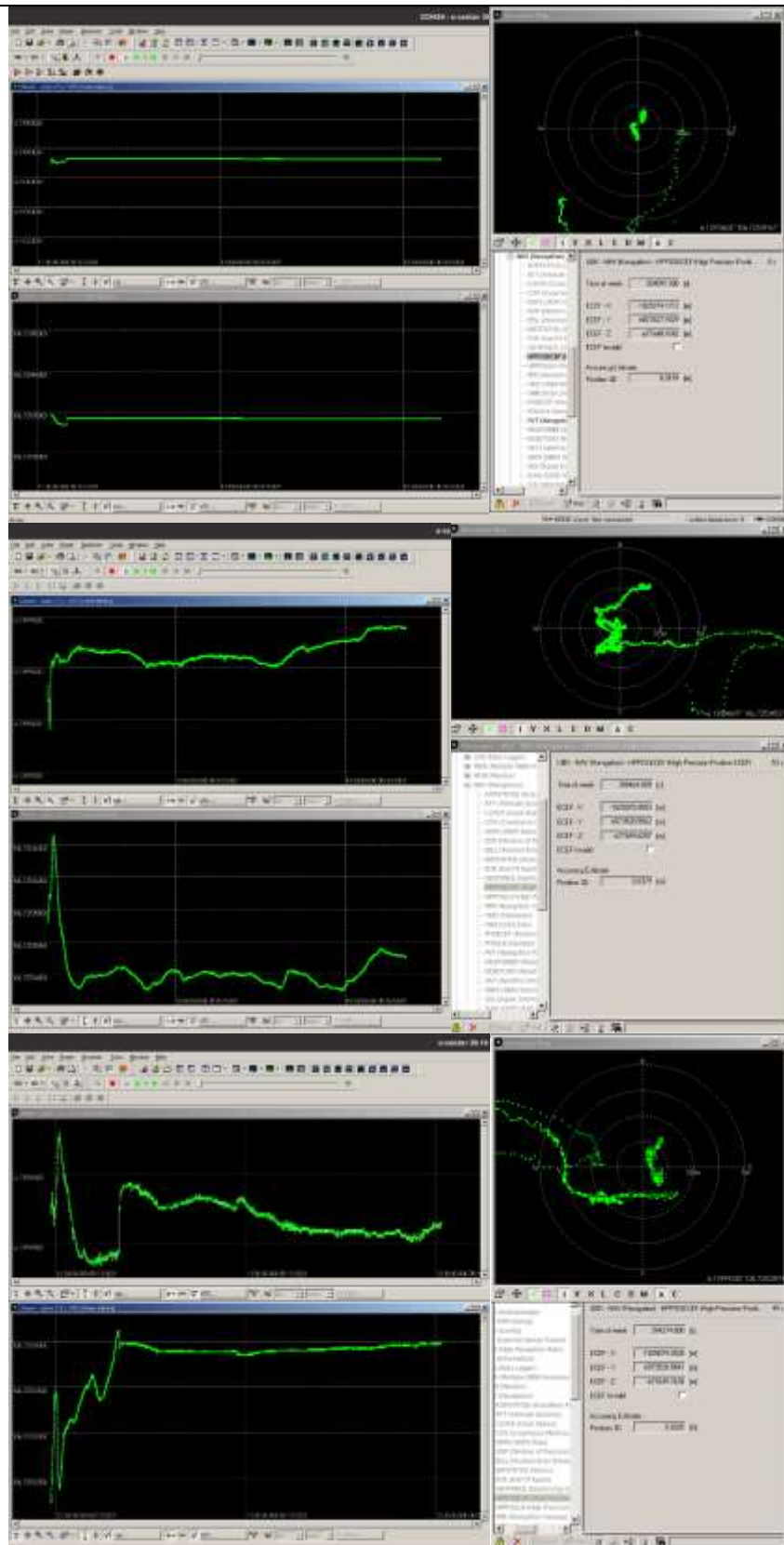


Figure 4-30 Data Taken on Night Time (RTK-GNSS)

Lower Number = Higher Accuracy	Errors of Rover Position on Different Weather Condition			
	Sunny Weather (cm)	Cloudy Weather (cm)	Rainy Weather (cm)	Night Time (cm)
	2.34	3.05	3.77	1.99
	2.44	3.05	4.22	3.79
	2.92	2.23		2.65
AVERAGE	2.57	2.78	4.00	2.81
TOTAL	3.04			

Table 4-4 Different Weather Position Accuracy Estimate of Rover Comparison

D) Position Accuracy Estimate on Longer Base Station Usage

In this test, the base station will be set up with one hour observation time on one meter required position accuracy. The data will be taken for one hour long each a total of three times consecutively on one base station setup. The goal of this test is to test the durability of base station setup data on how long will it still be able to give accurate RTCM corrections to the rover.

Figure 4-31 shows the data of three consecutive data takings for one hour each. From Table 4-5, it has a position accuracy estimate of 2.34, 2.44, and 2.92 centimeters. In average position accuracy estimate of 2.57 centimeters. From this data, we can conclude that despite the base station setup used for three hours of data taking, the position accuracy estimate is not much affected as the first data has 0.1 centimeters difference with second data and second data has 0.48 centimeters difference with third data. With an average position accuracy estimate of 2.57 centimeters, compared to 0.1 and 0.48 centimeters are minor differences.

Errors of Rover Position on Longer Base Station Usage			
First Hour (cm)	Second Hour (cm)	Third Hour (cm)	Average
2.34	2.44	2.92	2.57

Table 4-5 Three Hour Consecutive Data on Longer Base Station Usage Comparison

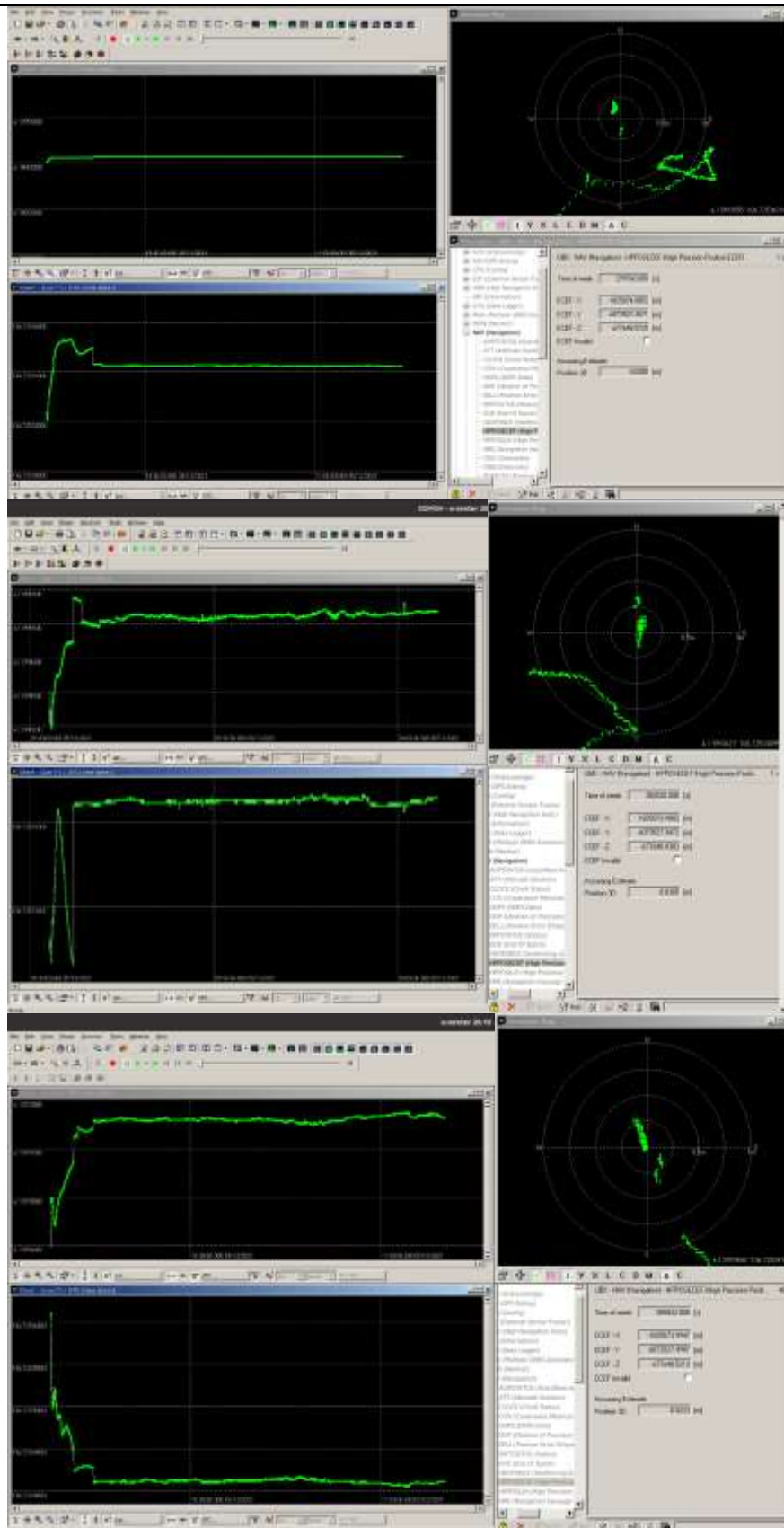


Figure 4-31 Three Hours Consecutive Data (RTK-GNSS)

E) Position Accuracy Visualized on Mapviz on Real Car

In this test, the base station will be placed in a higher place as the test sequence will be done outdoors as seen in Figure 4-32. The tests will be done by putting the rover on a real car and taken for a spin multiple times and data will be taken for each cycle visualized in Mapviz visualization. There will be also a test to check how far the RTCM corrections can be sent from the base station to the rover.



Figure 4-32 Base Station for Outdoor Testing

Figure 4-33 shows two data (left and right) with test taken sequence when the speed of the car is ranging from 10 to 15 km/h. Figure 4-34 shows two data (left and right) with test taken sequence when the speed of the car is ranging from 20 to 25 km/h.



Figure 4-33 Data Taken Visualized on Mapviz at 10-15 km/h on Real Car

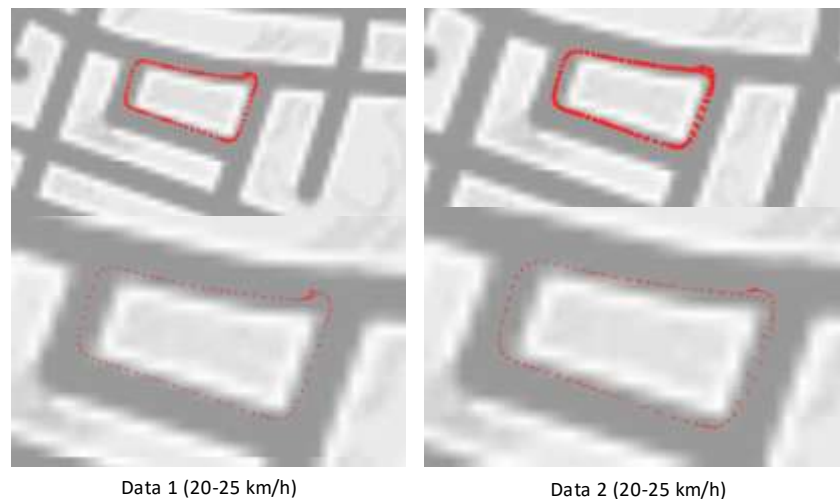


Figure 4-34 Data Taken Visualized on Mapviz at 20-25 km/h on Real Car

From the two sets of data, it is concluded that rover readings are still accurate as it still receives RTCM corrections from the base station. The thing that differs from the data taken at 10-15 km/h to 20-25 km/h is that when the rover is moving faster, it will receive fewer data compare to when the rover is moving slower. As this data is taken using ROS2 package `nmea_navsat_driver`, this will depend on how fast the node can subscribe to `/fix` the topic which consists of the GNSS geospatial position. By subscribing at a slower rate, it will also receive less data at a certain period. Regardless of the different amount of small dots which indicate the GNSS geospatial position, the accuracy of the RTK-GNSS is still accurate and reliable to be used in this project.

Figure 4-35 shows the data when the rover is driven to a further place in which the base station and rover are covered with buildings in between. Despite the smooth data shown, the module shows that there are no longer RTCM corrections received by the rover which makes the rover operating only as an ordinary GNSS. Thus, from this test, we can conclude that so that the RTCM corrections can be sent from the base station to the rover, the base station must be placed in a high location so that the radio telemetry can reach the rover. When the rover works as an ordinary GNSS, the exact accuracy cannot be visualized in Mapviz as it outputs the same visual accuracy as when the GNSS operates in RTK or ordinary mode. However, when the GNSS operates in RTK mode, it can reach the accuracy of centimeters but when it operates in ordinary mode, it can only reach the accuracy of meters.

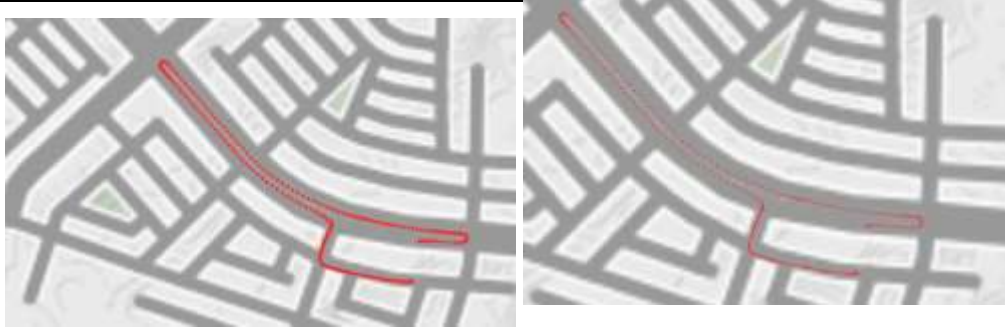


Figure 4-35 Data Taken Visualized on Mapviz when Rover No Longer Receives RTCM Corrections

4.2.7 LiDAR Accuracy Test on RViz2

In this test, the accuracy of LiDAR will be tested by measuring the actual distance of objects compared to the measured distance from LiDAR. Since ROS2 emphasizes its architecture for RTOS (Real-Time Operating System) applications, higher-level message types such as `sensor_msgs` require to publish and subscribe under a certain policy called QoS (Quality of Service). QoS allows the system to tune communication between nodes which includes publishers, subscriptions, service servers, and clients. However, if different QoS profiles are used, they might be incompatible thus preventing the delivery of messages. In this case, for ROS2, adding to the requirements for QoS, as ROS2 relies on RTOS applications, higher computation capabilities are required to read `sensor_msgs` data such as `/scan` topic of `sensor_msgs/msg/LaserScan` message type as the complete array that the topic send are not able to be subscribed entirely. Thus, this test will include two scenarios, first comparing the measured distance from LiDAR and the actual value, and second is a visualization of objects on RViz2.

Figure 4-36 shows how data is taken to measure the distance of objects from LiDAR. ROS2 package is used to display the output of `/scan` topic on a specific angle. Figure 4-37 shows the scenario of the data taking. Table 4-6 shows the comparison between actual distance and measured distance from LiDAR.

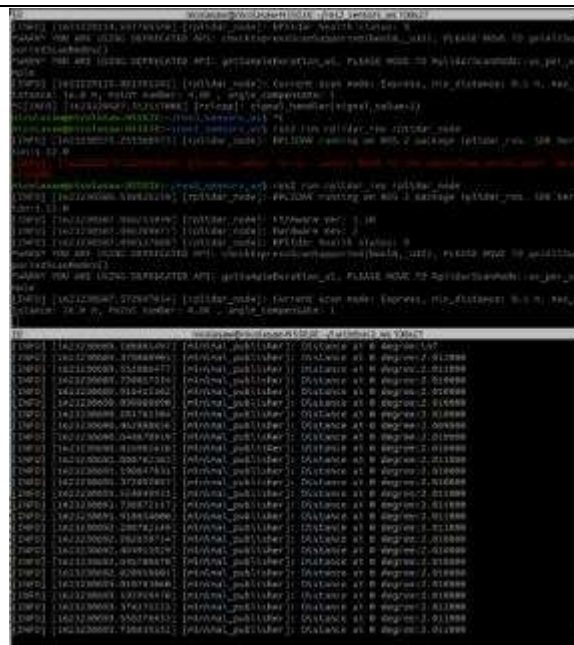


Figure 4-36 LiDAR Distance Measuring at 0°



Figure 4-37 LiDAR Distance Measuring Scenario

Actual Value (cm)	Measured Distance (cm)
10	inf (cannot measure)
12	12.3
15	14.5
20	19.7
30	29.8
40	39.9
50	49.8
60	60
70	69.9
80	79.9
90	90.5
100	100.2
110	110.5
120	120.6
130	130.3
140	140.2
150	150.1
160	160.6

Table 4-6 LiDAR and Actual Distance Measurement Comparison

From the data above, the distance measured from LiDAR compared to the actual value has a minimum difference of 0 centimeters with a maximum difference of 0.6 centimeters which has an average of 0.105882 centimeters and a standard deviation of 0.320791 centimeters.

The second data taking will only visualize how similar is the output of LiDAR laser scans compared to the actual position of objects. Figure 4-38 shows the arena where the LiDAR will be tested without any obstacles inside. Figure 4-39 and Figure 4-40 show the arena with several obstacles inside it and how is it visualized in RViz.

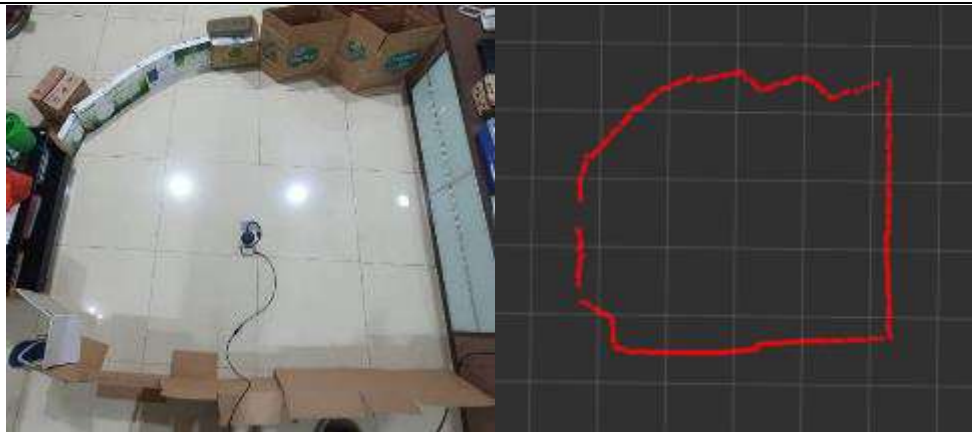


Figure 4-38 Empty LiDAR Arena for Object Detection Visualizaton

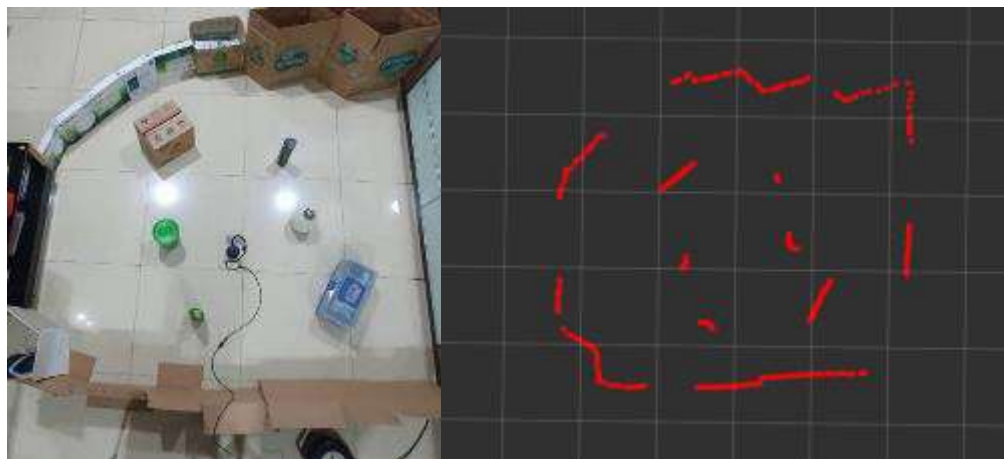


Figure 4-39 LiDAR with First Obstacles Setup on Arena

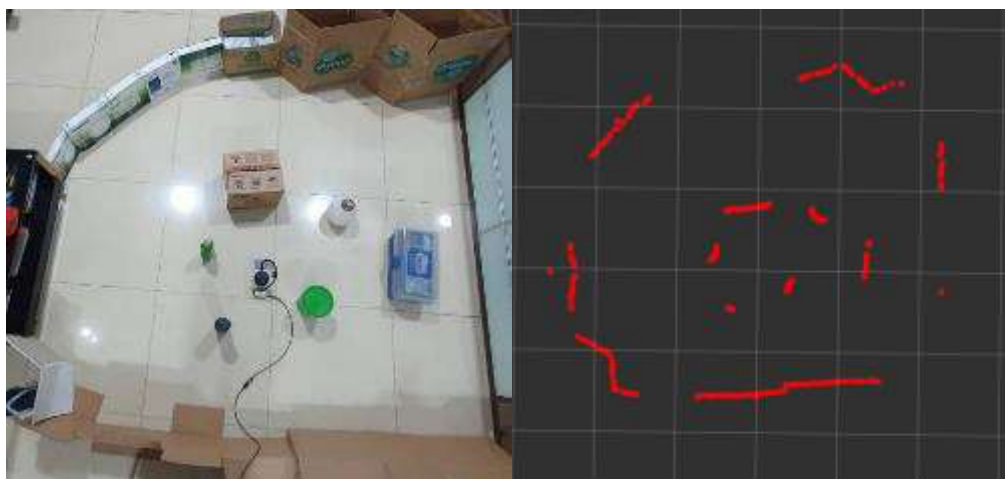


Figure 4-40 LiDAR with Second Obstacles Setup on Arena

4.2.8 Robot Actuation with Keyboard Teleoperation Visualized on Mapviz

In this test, both the capabilities of the robot and the accuracy of RTK-GNSS will be tested. As the robot can actuate smoothly with minor problems indoors, the limitations of the robot need to be tested outdoors. In the meantime, the accuracy of RTK-GNSS will be tested as well and visualized on Mapviz. Figure 4-41 shows the condition of how the robot will be tested. The robot will be driven with keyboard teleoperation for eight meters in a straight line for several repetitions.



Figure 4-41 Robot Movement Outdoors Testing Scenario

Figure 4-42 until Figure 4-47 shows respectively data 1 until data 6 for this test which shows the initial position visualized on Mapviz as well as its geospatial position and the final position also visualized on Mapviz with its geospatial position. The left figure is the initial state meanwhile the right figure is the final state. Table 4-7 shows the geospatial position of all the six data.



Figure 4-42 Data 1 Robot Movement with Keyboard Teleoperation Visualized on Mapviz



Figure 4-43 Data 2 Robot Movement with Keyboard Teleoperation Visualized on Mapviz



Figure 4-44 Data 3 Robot Movement with Keyboard Teleoperation Visualized on Mapviz



Figure 4-45 Data 4 Robot Movement with Keyboard Teleoperation Visualized on Mapviz



Figure 4-46 Data 5 Robot Movement with Keyboard Teleoperation Visualized on Mapviz



Figure 4-47 Data 6 Robot Movement with Keyboard Teleoperation Visualized on Mapviz

Data	Initial Latitude	Initial Longitude	Final Latitude	Final Longitude
1	-6.14018667	106.72533717	-6.14025633	106.72531683
2	-6.14020033	106.72534050	-6.14026967	106.72532050
3	-6.14020000	106.72534300	-6.14026950	106.72532017
4	-6.14020006	106.72535000	-6.14026960	106.72532800
5	-6.14020030	106.72534000	-6.14026967	106.72531857
6	-6.14019200	106.72534583	-6.14026017	106.72532350

Table 4-7 Geospatial Position Data Comparison between Initial and Final

From the results obtained as seen in Table 4-8, the latitude difference has an average of -0.00006926° and a standard deviation of 0.0000005° . For the longitude difference, it has an average of -0.00002149° and a standard deviation of 0.00000103° .

Data	Latitude Difference	Longitude Difference
1	-0.00006967	-0.00002033
2	-0.00006933	-0.00002000
3	-0.00006950	-0.00002283
4	-0.00006954	-0.00002200
5	-0.00006936	-0.00002143
6	-0.00006817	-0.00002233
AVERAGE	-0.00006926	-0.00002149
STD. DEV	0.00000050	0.00000103

Table 4-8 Average and Standard Deviation from Data 1 to 6

4.2.9 RTK-GNSS Accuracy with Robot Actuation

In this test, the accuracy of the RTK-GNSS will be tested with it equipped on the robot. The robot will be driven every two meters and the geospatial position will be recorded. From the recorded initial and final geospatial position, the distance between the two geospatial positions can be calculated. There will be five different data taken for this test.

Before the test is done, a calculation to convert geospatial positions into distances is required. Haversine formula will be used to do this conversion. Haversine formula calculates the spherical distance between two points on a sphere given their latitudes and longitudes. Thus, the following calculation can be used:

Known:

Earth's Radius $R = 6378.137$ km

Lat_{in} = Initial Latitude; Lat_f = Final Latitude;

Long_{in} = Initial Longitude; Long_{f} = Final Longitude

First, calculate the difference between final and initial latitude and longitude which has been converted from degree to radian. ($\text{Rad} = \pi / 180^\circ$)

$$d\text{Lat} = \text{Lat}_{\text{f}} - \text{Lat}_{\text{in}} ; d\text{Long} = \text{Long}_{\text{f}} - \text{Long}_{\text{in}}$$

Then, variable “a” needs to be calculated.

$$a = \sin(d\text{Lat}/2) * \sin(d\text{Lat}/2) + \cos(\text{Lat}_{\text{in}}) * \cos(\text{Lat}_{\text{f}}) * \sin(d\text{Long}/2) * \sin(d\text{Long}/2)$$

Thus, distance d can be obtained from:

$$D = 2 * R * \arcsin(\sqrt{a})$$

Figure 4-48 until Figure 4-52 shows respectively data 1 until data 6 for this test which shows the initial position visualized on Mapviz as well as its geospatial position and the final position also visualized on Mapviz with its geospatial position. The left figure is the initial state meanwhile the right figure is the final state. The initial position and final position are expected to be two meters difference. Table 4-9 shows the geospatial position as well as the distance in meters.



Figure 4-48 Data 1 Robot Two Meters for Manual Calculation



Figure 4-49 Data 2 Robot Two Meters for Manual Calculation



Figure 4-50 Data 3 Robot Two Meters for Manual Calculation



Figure 4-51 Data 4 Robot Two Meters for Manual Calculation



Figure 4-52 Data 5 Robot Two Meters for Manual Calculation

Data	Initial Latitude	Initial Longitude	Final Latitude	Final Longitude	Distance (m)	Error Compared to Actual (Percentage)	Actual Distance = 2 m
1	-6.14019383	106.72534783	-6.14020900	106.72534300	1.77	11.55	
2	-6.14019433	106.72534850	-6.14020983	106.72534367	1.80	9.78	
3	-6.14019483	106.72534883	-6.14021050	106.72534367	1.83	8.33	
4	-6.14019533	106.72534900	-6.14021033	106.72534383	1.76	11.85	
5	-6.14019500	106.72534917	-6.14021017	106.72534400	1.78	10.97	
AVERAGE					1.79		
STD. DEVIATION					0.026		

Table 4-9 Geospatial Position Data and Distance between Initial and Final Position

From the data taken, the distance measured has a minimum of 1.76 meters and a maximum of 1.83 meters with an average of 1.79 meters and a standard deviation of 0.026 meters. In this test, the measured distance is supposed to be two meters thus the average error of the calculation from the actual distance is at 10.5%. There could be several reasons the measured distance is not at two meters. One of the reasons is as the actuation of the robot is done with keyboard teleoperation, the movement and the halt of the robot are controlled in the program in which a PID controller is used. As the PID controller has not been tuned, the value of the PID is not yet optimal for the system. Thus, the final distance of the robot is not exactly at two meters. The other reason is the inaccuracy of the RTK-GNSS readings as a ROS2 node is used to subscribe `/fix` topic which consists of RTK-GNSS spatial position, the ROS2 node is unable to obtain the whole exact geospatial position up to the last decimals which will reduce the accuracy of the readings despite using RTK-GNSS.

4.2.10 GNSS Geospatial Position to XYZ Coordinates

Before this test is done, further changes need to be done for the GPS2XYZ package by ShantihBlah as the structure of the code is unsuitable to be used in the system. The ROS2 package will be able to capture the initial geospatial position and set it as the zero coordinate of the robot. It will also be able to measure the changes in geospatial position and display the distance compared to the zero coordinate of the robot in ENU (East North Up) coordinates.

The test will be done by going on square routes to measure the distance. Figure 4-53 until Figure 4-57 shows data taking for square routes movement. Table 4-shows the comparison of the distance from the GPS2XYZ package and calculated from the Haversine formula. Since the GPS2XYZ package calculates the distance with ENU

[illegible]

Figure 4-53 Square Route 1 Data Taking with GPS2XYZ



Figure 4-54 Square Route 2 Data Taking with GPS2XYZ

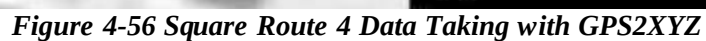




Figure 4-57 Square Route 5 Data Taking with GPS2XYZ

Data	Measured Distance 1-0 (m)	Haversine Distance 1-0 (m)	Measured Distance 2-1 (m)	Haversine Distance 2-1 (m)
1	6.91	6.93	8.02	8.01
2	6.42	6.45	5.81	5.81
3	5.77	5.51	6.07	6.10
4	5.69	5.74	6.78	6.78
5	7.53	7.57	6.94	6.93
AVERAGE	6.46	6.44	6.73	6.72
DEVIATION	0.64	0.69	0.71	0.70

Measured Distance 3-2 (m)	Haversine Distance 3-2 (m)	Measured Distance 4-3 (m)	Haversine Distance 4-3 (m)	T O T A L
7.34	7.32	8.21	8.12	
6.27	6.31	7.98	7.98	
5.28	5.30	7.16	7.16	
5.21	5.24	6.53	6.53	
4.74	4.73	7.40	7.39	
5.77	5.78	7.46	7.43	
0.85	0.84	0.55	0.53	6.60
				0.69

Table 4-10 Data Comparison for Square Route with GPS2XYZ

In this test, distance 1-0 and 3-2 should be equal as well as distance 2-1 and 4-3 should be equal. For distance 1-0, the measured value has an average of 6.46 meters and a standard deviation of 0.64 meters meanwhile haversine value has an average of 6.44 meters with a standard deviation of 0.69 meters. For distance 2-1, the measured value has an average of 6.73 meters with a standard deviation of 0.71 meters meanwhile

haversine formula has an average of 6.72 meters with a standard deviation of 0.7 meters. For distance 3-2, the measured value has an average of 5.77 meters with a standard deviation of 0.85 meters meanwhile haversine formula has an average of 5.78 meters with a standard deviation of 0.84 meters. For distance 4-3, the measured value has an average of 7.46 meters with a standard deviation of 0.55 meters meanwhile haversine value has an average of 7.43 meters with a standard deviation of 0.53 meters. Comparing the average of calculation from haversine formula and the measured value as seen in Table 4-11, the average error is at 0.37%. This shows that the readings of the geospatial position of the rover are not accurate as not only the distance from the GPS2XYZ package but also from the Haversine formula does not make a square as both widths do not have the same length, as well as both heights, do not have the same length.

Data	Total Average Measured	Total Average Haversine	Error Compared to Haversine Distance (Percentage)
1	7.62	7.59	0.33
2	6.62	6.64	0.23
3	6.07	6.02	0.91
4	6.05	6.07	0.30
5	6.65	6.65	0.07

Table 4-11 Data Average Comparison Between Measured and Haversine Calculation

4.3 Discussion of Results and Error Diagnoses

In this subchapter, the results of the testing will be discussed as well as errors that happen during the testing.

4.3.1 Linorobot Reliability on ROS2

From the results from Linorobot firmware porting from ROS to ROS2, it shows that because ROS2 is considered new, many of its features are not stable yet and require more development. Adding to the lack of documentation available, the development of a robot based on ROS2 is considered difficult. As ROS mostly uses Python language to create its nodes and launch files, ROS2 will have a hard time matching as most ROS2 architectures are built on C++ language. Thus not everything in ROS can be accessed directly on ROS2 for example, not the whole Linorobot software can be used in ROS2 as only Linorobot firmware is used which is on the Teensy side. Adding to the QoS features that were added in ROS2 which do not exist in ROS, these features make significant changes to publishing and subscribing topics in ROS2 which when QoS is

not compatible, publishing and subscribing topics may do not occur. Linorobot firmware for ROS2 used in this thesis is self-ported as no previous documentation has ported Linorobot software from ROS to ROS2.

Referring to results from Linorobot software, MicroROS which is used to connect Teensy and ROS2 is not ready to use commercially as it is always really hard to connect Teensy to ROS2. Whenever the MicroROS agent is started, it needs several repetitions for Teensy to be able to connect to ROS2 which number of repetitions cannot be specified.

4.3.2 IMU and Rotary Encoder for Odometry

From the results for IMU and rotary encoder calibrations and data taking, IMU MPU9250 is reliable as it is accurate enough to have a similar measurement compared to the actual value. Despite its high reliability, MPU9250 will need to be calibrated once a week to avoid the drifting of the data. For rotary encoder HE50B, the data shows that it has slight accurate readings compared to tachometer readings. Since the encoder used has a resolution of 360 pulses per rotation, it is not as sensitive as it will give a pulse every one degree. With a higher resolution encoder, the more sensitive it will be and the more accurate and detailed output it will give. All in all, both MPU9250 and HE50B are reliable and can be used for robot odometry as accelerations, headings, and robot displacements can be obtained from these two sensors to be used for localization.

4.3.3 Localization with RTK-GNSS and Mapviz Visualization

For RTK-GNSS, several tests are done such as base station and rover connections reliability, base station settings requirements, data taking on different weather, longer duration data taking, as well as the base station and rover connections on high-speed applications on a real car. The GPS-RTK2 module by U-Blox provides a feature that eases monitoring whether the rover has communicated with the base station through RTCM. If the base station is not on RTK-Fixed mode, the rover will not enter RTK mode which guarantees the rover not operating on RTK mode. For the base station, the smaller the accuracy requirements that are set, the fewer outliers received in the rover. For different weather, the geospatial position accuracy estimate does not vary much ranging from one to five centimeters with rainy weather having the worst position

accuracy estimate. However, different weather does not affect much on the position accuracy estimate, and having centimeters accuracy is reliable for AMR applications. For long period data taking, the base station survey-in for one hour will persist after three hours of data taking, however, it will slowly reduce its accuracy as for longer period data taking.

For testing on real cars with Mapviz, it can be seen that the data still shows a smooth movement of the rover on the map and despite driven at 10-30 km/h, the data is still reliable to use despite the lack of points for faster movements.

4.3.4 LiDAR for Object Detection

For LiDAR, two different kinds of tests are done in which first to measure the accuracy of LiDAR measuring distance using laser scan compared to the actual value. For the distance measured by the LiDAR, it can be seen that the inaccuracy compared to the actual value is small which can be compensated with the program for object detection. Next, the capability of LiDAR to capture surrounding objects is tested and it can be seen from the results that for bigger surface objects, it can detect most of the reflections of the laser. But for smaller and curvy objects, it is harder to capture the reflections thus having small points only on that object.