Spine

Acad Year 2018/2019

Project No. (C018)

LIDAR INERTIAL ODOMETRY LOCALIZATION OF UAV IN TUNNEL-LIKE ENVIRONMENT

cover

Lidar Inertial Odometry Localization of UAV in Tunnel-like Environment

SCHOOL OF MECHANICAL AND AEROSPACE ENGINEERING

NANYANG TECHNOLOGICAL UNIVERSITY

Year (2018/2019)

Title Page

Lidar Inertial Odometry Localization of UAV in Tunnel-like Environment

SUBMITTED

BY

CHOW JIUN FATT

SCHOOL OF MECHANICAL AND AEROSPACE ENGINEERING

A final year project report
presented to
Nanyang Technological University
in partial fulfillment of the
requirements for the
Degree of Bachelor of Engineering (Mechanical Engineering)
Nanyang Technological University

Year 2018/2019

ABSTRACT

Mobile robots are capable of various applications, ranging from simple delivery tasks to advanced surveillance operations. These abilities rely heavily on its localization ability, which provides stable and accurate pose estimations within the environment.

Localization can be achieved by different sensors and techniques such as GPS, WIFI, ultrasonic sensors. In this paper, we focus on the laser-based localization method of UAV (drone) in a GPS-denied environment such as depth tunnel. Due to the limited payload for sensors on a UAV, a 2D LIDAR sensor is used to perform localization with the fusion of inertial measurement unit (IMU).

IMU is an additional sensing input from the onboard flight control unit – Pixhawk, to increase the reliability and accuracy of localization performance. In this paper, we present a comparative analysis of the three most common laser-based 2D Simultaneous Localization and Mapping (SLAM) approached: Hector SLAM, Gmapping, Cartographer, in terms of localization against the precise ground truth and mapping quality.

To evaluate the performance, experimental datasets were collected under the same conditions and compared to ground truth value in Motion Capture Lab. The experimental results were analysed in the aspect of accuracy and mapping quality via MATLAB. This study investigates these open-source packages and the real-time evaluations on the SLAM performance (i.e. localization and mapping) in surrounded environments.

ACKNOWLEDGEMENT

Firstly, I would like to thank my supervisor, Professor Gerald Seet for providing me with a great opportunity to work on this project. Furthermore, I sincerely thank my mentor, Mr. Kocer Basaran Bahadir who has always been helpful and provide me a lot of guidance and advice throughout my project.

I would like to offer my special thanks to Mr. Dogan Kircali, Mr. Iuldashov Nursultan, Mr. John Hawery, for their pieces of advice and assistance in setting up the experiments. Not to mention the technical staffs in Mechatronic lab who also have provided me support on resources and guidance. Furthermore, I sincerely appreciate the School of Mechanical and Aerospace Engineering for financial support in purchasing the necessary components needed in this project.

Finally, I wish to thank my parents for their support and encouragement throughout my study.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Background

Mobile robots have become extremely commonplace in many fields, for instance, industrial, military and commercial settings. Compared to stationary robots that are fixed in position, mobile robots have the advantage of their flexibility and mobility within the operating environments. Therefore, they can perform applications such as transportation, inspection, and even entertainment. Mobile robots can be further classified by their travel methods: 1) Unmanned Ground Vehicles (UGVs), 2) Unmanned Aerial Vehicles (UAVs), also known as a drone, and 3) Unmanned undersea vehicles (UUVs). In particular, UAVs have become a valuable platform for specific tasks such as inspection, surveillance, aerial photography and mapping due to their ability to avoid obstacles [1]. In 2017, the Land Transport Authority (LTA) has researched intensively on the use of UAV technology for more efficient and safer tunnel inspections [2]. To perform these tasks, localization plays a crucial role in determining the current position and orientation of the mobile robot with respect to a reference frame [3].

1.2 Problem and Challenges

In recent years, various algorithmic approaches have been presented for the mobile robot's localization, combining different sensors configuration, such as visual-based localization like a stereo camera and optical flow sensor, laser-based localization and Global Positioning System (GPS) [4]. Localization configuration can be further varied based on the environmental conditions, which are indoor and outdoor. Outdoor localization can easily be achieved using GPS while indoor localization mainly relies on a stereo camera, radio waves or beacons configuration. In recent years, simultaneous localization and mapping (SLAM) technique is experiencing more demands and attention because of its capability to construct an unknown map while instantaneously locate its position for both outdoor and indoor [5]–[7]. Especially, SLAM using cameras is referred as visual-based SLAM (vSLAM) which based on visual information only while SLAM using the LIDAR sensor is referred as laser-based SLAM which relies on laser scan information [8]. From then on, the

SLAM technique becomes superior than GPS technique in a tunnel-like environment where signal interference or signal denial occurs. However, a poor light condition in such an indoor environment can be a severe problem to visual-based SLAM algorithm which mainly replies stereo camera to detect the surrounding features. Especially for aerial mobile robots which only have onboard inertial measurement unit (IMU) to provide angular rates and translational accelerations, compared to those landed mobile robots which have encoder that provide odometry information. Therefore, a laser-based SLAM algorithm is proposed for UAV localization in such tunnel-like environment [9].

1.3 Objective

The aim of this project is to implement a potential laser-based SLAM technique which meets our desired tolerance of 15cm with LIDAR sensor as main perception input. On-board IMU is used to further improve both reliability and accuracy of pose estimation by eliminate unusable laser scan (caused by rolling and pitching). Three potential approaches, namely Hector SLAM, Gmapping, and Cartographer, were implemented and configured to be tested on the UAV platform. The mapping and localization performances were then compared with ground truth data in the motion capture lab, and the pose estimation error of both approaches was evaluated and discussed in the paper. The evaluations from this project might potentially determine which approach is more suitable for UAV localization in a tunnel-like environment.

1.4 Scope

The scopes covered in this project:

- To select suitable LIDAR sensor
- To review potential SLAM approaches
- To implement and configure each approach
- To improve SLAM result with the aid of IMU data
- To construct a setup for testing and comparison
- To evaluate the accuracy of each compared to the ground truth value

- To build a whole UAV platform with necessary components
- To fuse altitude for 6 DoF position information

Noted, offline experiments are conducted via a hand-hold manner instead of flying the real drone remotely due to safety factor and uncertainty of the selected package's performance. The on-board experiment is then conducted in Motion Capture Lab once the system is properly set up and piloted by professional personal.

Worth to mention, autonomous piloting is not within the scope of this project, therefore only localization and mapping performance will be covered.

2. REVIEW OF THEORY AND RELATED WORK

2.1 Localization and Mapping

Localization of mobile robot is defined as the method to determine the pose information with various sensors configuration and mathematical algorithm within the environment, either in 3 DOF or 6 DOF. With the advanced innovation in localization technology, Global Positioning System (GPS), becomes most widely used in the world to perform localization, such as the automotive navigation system, aircraft tracking, self-navigating. However, GPS is not suitable for all environments, for instance, a tunnel-like environment is basically signal denial or signal interference. Henceforth, there are many approaches developed to achieve localization but with various sensors. For example, LIDARs, ultrasonic sensors, stereo cameras are common sensors configuration mentioned in [4]. Especially LIDAR-based localization is more superior when working in visually degraded environments (e.g. darkness).

Figure 1: Deepest tunnel in Singapore

In autonomous industry, Simultaneous Localization and Mapping (SLAM) is introduced as the most widely researched topic. It is useful for creating and updating maps within an unknown environment, while keep tracking the position of mobile robot instantaneously within the same system. Therefore, there has been a wealth of researches into SLAM algorithms, with reliably working solutions for typical office-like indoor environments and outdoor environments using particle filters like Gmapping. Visual odometry estimation is also widely used for landed-robots [10] due to the direct translation information getting from the odometry encoder. Most of

them are being available as open software for individual development in [5]–[7], [9], [11], [12].

Since various of SLAM algorithms has increased in the past few years, the need to compare different approaches depending on the mobile robot and the characteristic of the working environments increases tremendously. Visually inspection of the resulting maps cannot lead to a correct comparison, therefore evaluation of the results in a quantitative scale is necessary. For instance, in [13], the comparison of SLAM algorithms is conducted by considering the poses of the robot during data acquisition. This fact allows comparison between approaches with different outputs. Also, the proposed method is not affected by the sensor configuration of the mobile robot.

2.2 Inertial Measurement Unit (IMU)

IMU is an electronic device that determines and measures an object's orientation, angular velocity, linear acceleration, and sometimes the magnetic field surrounding the body. A typical IMU consists of three accelerometers and three gyroscopes mounted in a set of three orthogonal axes, sometimes also magnetometers. It is commonly used in pose estimation for dynamically moving an object in all three dimensions at a high sampling rate, typically higher than 100 Hz [14]. However, it suffers from severe noise during relatively slow-moving motion, caused accumulated error - drifting.

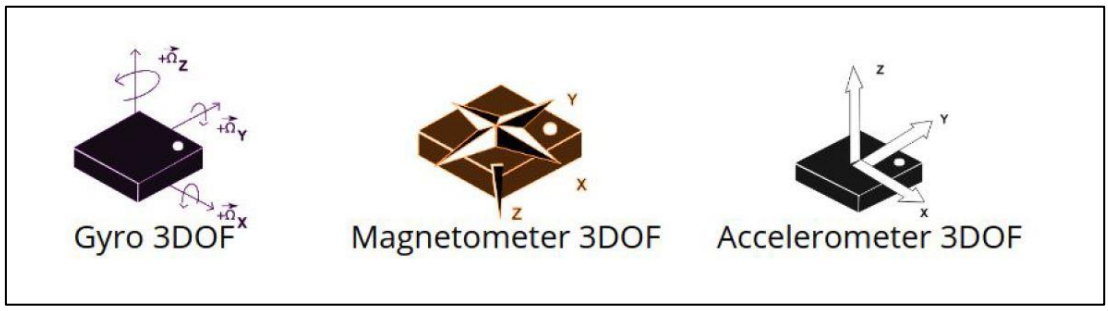

Figure 2: Build-in sensors within IMU

The gyroscope measures the angular velocity of each axis from the Coriolis Force which applied to a vibrating element in the sensor. Magnetometer applies the principle of Hall Effect detect earth magnetic field which affects the current flows

within the sensor. Similarly, accelerometer contains small masses that enable detection to linear acceleration.

Technically, IMU can be found in most UAVs application, mainly for control unit system. In our project, Pixhawk is the flight controller powered by ARM Cortex M7 and low-noise Bosch Sensor Tec IMU. It supports multiple flight stacks to pilot various multi-copters, fixed-wing aerial platforms, and even ground-based mobile robots [15].

With the characteristics of IMU sensor, especially the absolute output values of the IMU orientation information (RPY) is extremely reliable, we can utilize and fuse it with SLAM techniques to further increase the accuracy. During the flying motion of UAV, we can except large motion in RPY, therefore by introducing rotation matrix transformation we can obtain truth orientation of the body.

Figure 3: Pixhawk Flight Control Unit

2.3 Transformation Frames for Mobile Platforms (TF tree)

General rigid body motion (or general motion of reference frames) consists of a combination of translation and rotation. By simple definition, translation involves the changing of a linear distance, whereas rotation governs the changing of an angular displacement (or an angle). Therefore, a mobile robot that moves in a 3D space has 6 attributes to represent its pose information, particularly XYZ (position) and Roll

Pitch Yaw (orientation). Before diving straight into 3D transformation, we first understand a 2D transformation, a top-down planar view of the environment. Essentially, only 3 attributes are needed to represent the pose in a planar perspective, there are coordinates X, Y, and yaw. Table 1 describes a general definition of each term according to [16].

Table 1: List of general terms and definitions

Name	**Definition**
Laser_link	position of the sensor with respect to the robot base
Base_link	rigidly attached to the mobile robot base, affected by the roll, pitch angles
Base_stabilized	transformed frame which aligns with the orientation of map frame, provides height information
Base_footprint	provides no height information and represents the 2D pose of the robot (pose and orientation)
Map	a world fixed frame, with its Z-axis pointing upwards. The pose of the mobile platform is relative to the map frame. In a typical setup, the origin of the map frame is the beginning position of mobile robots before localization

The transform from base_link to map is computed by a localization technique. However, the localization component does not broadcast the transform from map to base_link directly. Instead, it first receives the transform from base_stablized to base_link and uses this information to determine the position.

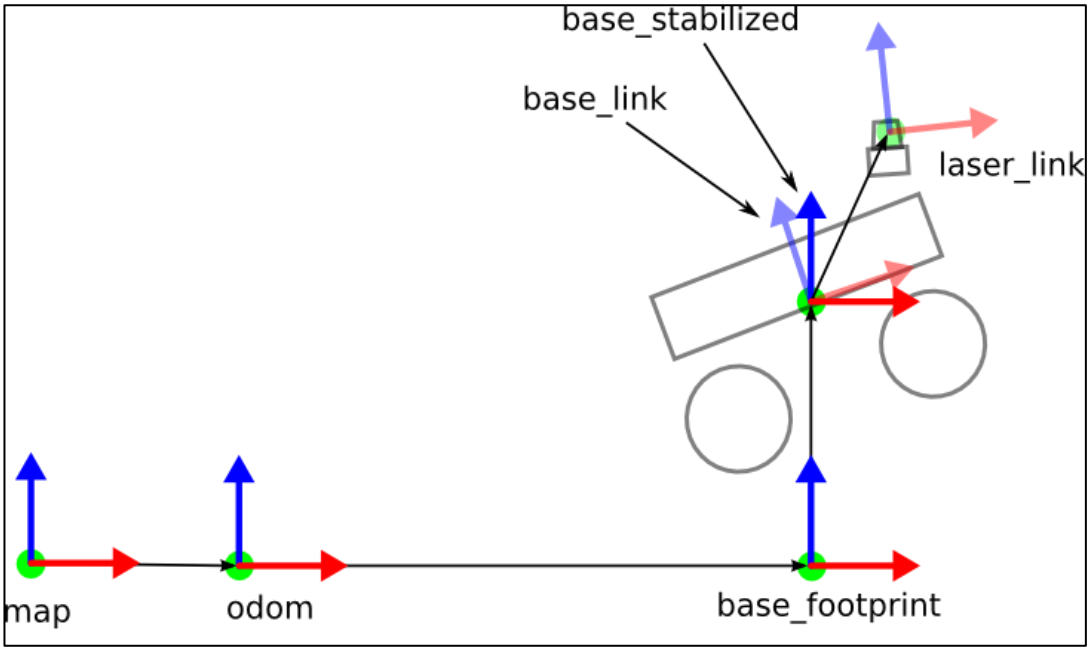

Figure 4: General relationship between the frames

2.4 Software framework

The Robot Operating System (ROS) [17] framework is widely used as a preferred working environment to implement the programming codes. In this project, ROS Kinetics was extensively used since it is the current stable version and it allows us to interconnect programs written in different languages and communicate among multiple machines seamlessly.

The fundamental concepts of the ROS framework are nodes, messages, topics, and services. In our project, only the first three concepts were extensively applied. Nodes are basic processes that perform the computation. Under a complex system, multiple nodes are running concurrently and interchange data.

Nodes communicate with each other in a several ways: Messages, Services, or Actions. Frequently, messages are sent in a specific channel (called topics) with a non-blocking way which means the same message can be received by multiple servers at the same time. A node that is interested in a specific data will subscribe to the appropriate topic. There may be a single node publish and/or subscribe to multiple topics, or the other way around.

ROS also supports generic logging and playback functionality which allows us to record the simulation data and reuse it for examination purposes. Any ROS message (e.g. laser scan point) can be dumped to a local folder and later replayed. Importantly, time properties will also be recorded. For example, the following dataset could be quickly played back into a different node, which runs another algorithm to proceed it.

A single node is not sufficient for the whole project. Third party nodes created by other developers are necessary and is used to examine the results. These are:

- **Rviz**: It is a visualization node used to observe the messages generated by nodes. For example, pose information, laser scan, map construction etc.

- **static_transform_publisher**: It publishes a static coordinate transform to offset the position of sensors to the base frame

3. LIDAR SELECTION

This section presents several laser sensors (LIDAR) that are suggested to be the main sensor input in our platform. The comparison of the LIDARs shortlisted are presented in Table 2 and they will be evaluated based on the following design requirements: detecting range, scanning rate, weight etc.

3.1 Laser Sensors (LIDAR)

LIDAR (Light Detection and Ranging) is a remote sensing method that uses light in the form of a pulsed laser beam to measure ranges (variable distance). The capabilities of the LIDAR sensor selected is extremely critical in our project because it serves as the main sensing input for localization algorithm. Therefore, all specifications and requirements of the project will be discussed in the following section.

3.1.1 RPLidar A1

The RPLidar A1 [18] is a low cost 360-degree omnidirectional 2D laser scanner, contains a range scanner system and a motor system. The system can perform a scan within a 12-meter range. Its scanning frequency reached 5.5Hz with 360 sampling points each round and it can be configured up to 10 Hz with 8000 sampling points per second. RPLidar A1 based on laser triangulation ranging principle and works excellent in all kinds of indoor environment and the outdoor environment without sunlight. It can be connected via micro USB with ROS integration.

Figure 5: RPLIDAR A1

3.1.2 RPLidar A2

As an upgraded version of A1, RPLidar A2 [19] also performs a 360-degree omnidirectional laser scan ranging from 0.20 m to 12 m/18m while thickness is reduced to only 4cm. The system adopts the low power infrared laser light as its light source and drives it by using a modulated pulse. It measures distance data 8000 times per second and also has an excellent performance in a long distance. Compared to the traditional belt drive mode, RPLIDAR A2 uses the self-designed brushless motor to reduce the mechanical friction in running. Therefore, the RPLIDAR A2 can run smoothly without any noise.

Figure 6: RPLIDAR A2

3.1.3 LDS-01

The LDS-01 [20] has a physical size of 69.5mm x 95.5mm x 39.5mm, a weight of 125g and four mounting points on the bottom. It has a scan angle of 360 degrees, a maximum scan range of 3.5m with an angular resolution of 1 degree. Scan settings are adjustable through a 3.3V UART connection. It is designed for indoor usage with a scan rate of 300 rpm or a sampling rate of 1.8 kHz.

Figure 7: LDS-01

3.1.4 Hokuyo URG-04LX

The Hokuyo URG-04LX is a high-end laser brand with a high reputation in the industrial sector. It has a physical size of 50mm x 50mm x 70mm with a weight of 160g. Unlike previous models, it can only scan with 240 degrees but with more precise resolution of 0.36 degrees, scan ranging from 0.06m to 4.0m. It is powered by 12V DC and specially designed for indoor application.

Figure 8: Hokuyo URG-04LX

3.1.5 Hokuyo UTM-30LX

The Hokuyo UTM-30LX is the most widely used laser scanner for mobile robots' application. It has a physical size of 60mm x 60mm x 85mm, a weight of 210g. Compared to the previous model, it has a scan range of 270 degrees with an angular resolution of 0.25 degrees. Its scanning range can reach up to 60m and adjustable through the USB connection. The device is designed for both indoor and outdoor environments with an IP rating of IP64.

Figure 9: Hokuyo UTM-30LX

3.2 Comparison of Lasers

Each of the sensors listed in section 3.1 has its advantage and disadvantage. Therefore, the table below tabulates each sensor in the form of specific characteristics such as scan range, field of view (FOV) etc.

Table 2: LIDARs hardware specifications

Laser model	Range (m)	Field of view (degree)	Scanning frequency (Hz)	Angular resolution (degree)	Weight (g)	Price (S$)
RPLidar A1	[0.15, 12]	360	10	1	200	180
RPLidar A2	[0.20, 12]	360	10	1	190	560
LDS-01	[0.12, 3.5]	360	5	1	125	250
Hokuyo URG-04LX	[0.06, 4]	240	10	0.36	160	1484
Hokuyo UTM-30LX	[0.1, 30]	270	36	0.25	370	3000

In our project, a UAV is expected to carry a LIDAR sensor less than 250g due to the payload constraint. To perform the SLAM algorithm properly, an omnidirectional laser scan is also desired. The working environment is an indoor space like a sewerage tunnel with 6m wide, referring to [21]. Most importantly, we only have a budget of S$500 under the FYP scheme, therefore we can eliminate the sensors which are overbudgeted. From the remaining two sensors, RPLidar A1 and LDS-01, we decided to stay with the former as it has superior range, resolution, and scanning rate of 10 Hz.

3.3 SDK of RPLidar A1

As mentioned, RPLidar A1 is compatible with ROS integration. Before we use it for any SLAM algorithm, we will test the basic functionality and visualize its scanning result via SDK provided from the official website. Figure 10 shows the visualization of laser beams via Rviz.

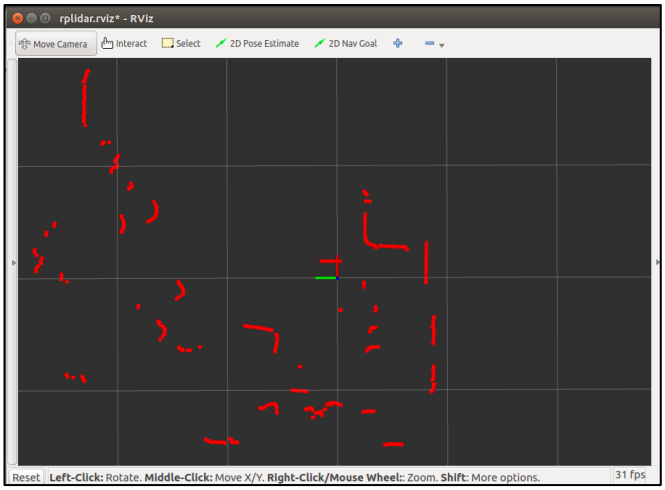

Figure 10: Visualization via Rviz

SLAMTEC company provides SDK in both Windows and Linux systems. For ease of convenience, we installed the source code from GitHub and followed the tutorials [22]. Additionally, we figured out that there are three possible scanning modes for A1 model, namely: Standard mode (2k sampling point), Stability mode (4k sampling point) and Boost mode (8k sampling point). To achieve the best scanning performance of the laser sensor, we configured it to Boost mode by launch script files. Figure 11 shows the default mounting orientation of RPLidar sensor.

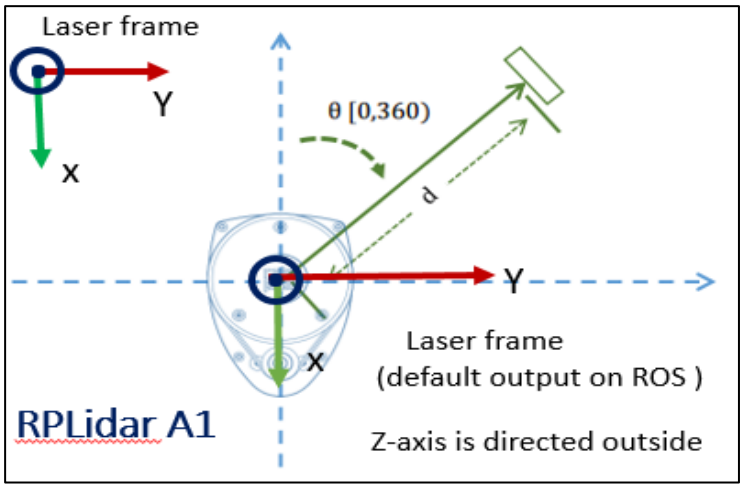

Figure 11: Suggested installment orientation of RPLidar A1

4. IMPLEMENTATION METHODS

This section discusses the characteristic of chosen SLAM techniques, the configuration and fine tuning of them to perform seamlessly with our LIDAR platform. To accomplish this goal, the individual installation and setup will be taken according to the online tutorials respectively. On the hardware, personal laptop is used to perform software implementation, with the following specifications:

- Intel® Core™ i5-4210U CPU@1.7GHz quad core
- 8GB RAM
- NVIDIA 840M GPU

4.1 Hector SLAM

Hector SLAM [5], [7], [12], [23] is generally a 2D SLAM approach that incorporates with 2D LIDAR sensor to generate a map from the laser scan. In contrast to other SLAM techniques (e.g. mostly used Gmapping), Hector SLAM does not require any auxiliary odometry sensor (e.g. wheel encoders) which directly measure the travel distance of a land-based robot, but only relies on the information from the laser scan matching approaches. Therefore, Hector SLAM is more suitable for aerial vehicles. Traditionally, Iterative Closest Point (ICP) is used to perform features scan matching, but it is computationally expensive [9]. Instead, in Hector SLAM, it takes advantage of the low distance measurement noise and high sampling rates of LIDAR for a fast scan-matching method [7]. Another advantage of Hector SLAM is its capability to generate multi-resolution grid maps to avoid singularity during scan matching.

A map is generated by Hector SLAM according to the endpoints of the LIDAR beams hit onto the walls. Then the transformation of the current scan is determined by the Gauss-Newton approach, which finds the best alignment of the current scan to the map generated previously. In its current implementation, Hector SLAM can only provide 2D pose estimation (x, y, and yaw) and a 2D map which can be visualised through Rviz.

4.1.1 Initial setup and configuration

Since Hector SLAM is an open source package that available from ROS.org, the most stable version was cloned into our workspace and built into an executable binary file according to tutorial [24] respectively. Detailed implementation steps will not be discussed in this paper, only the initial SLAM results and further configuration are shown as the following.

Basically, Hector SLAM is a meta-packages which includes many individual packages to perform multiple purposes. The main packages are:

1. Hector_mapping: The SLAM node which performs mapping and localization using laser scan information

2. Hector_geotiff: Saving of map into geotiff images files

3. Hector_trajectory_server: Convert pose information into tf based trajectory

To use Hector_mapping, a source of *sensor_msgs/LaserScan* data (e.g. from RPLidar) has to be subscribed. The static transformation of laser link to base link must also be known, by publishing via static_transform_publisher node mentioned in the earlier section. Figure 12 shows the relationship of coordinate frames in the current setup.

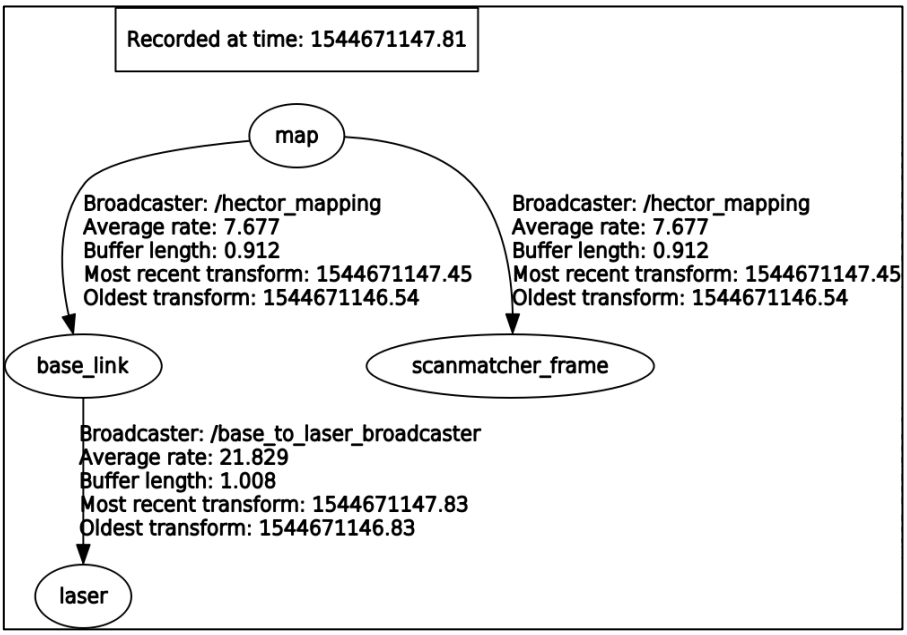

Figure 12: Hector SLAM's TF frame tree

According to the official API documentary [24], some parameters need to be set up before we can perform Hector SLAM. Firstly, both *base_frame* and *odom_frame* was set to be "base_link" while other parameters were the default value. The initial experiment was taken in NTU hostel double room. Figure 13 shows how each node of Hector SLAM cooperates seamlessly.

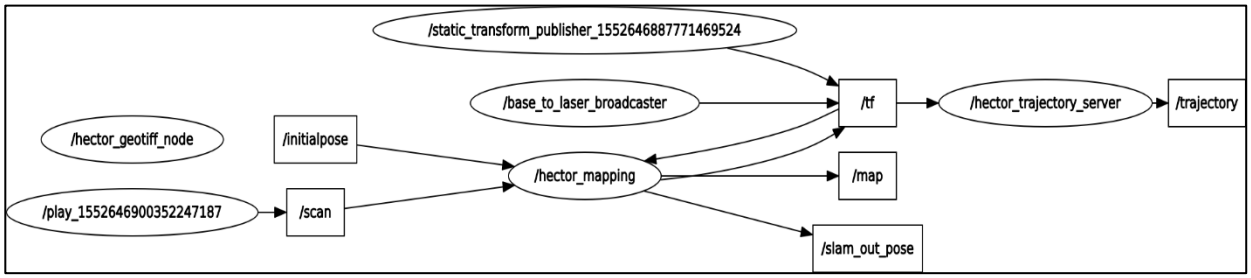

Figure 13: Dynamic graph of computation system. Circular box denotes running node. Rectangular box denotes active topics.

In Figure 15 to Figure 14, we can see that during the linear movement (without changing too much on the heading of RPLidar) the SLAM performance was decent. However, the severe overlapping issues occurred during circular movement. This can conclude that its current configuration cannot handle the huge changing of heading properly due to untuned parameters.

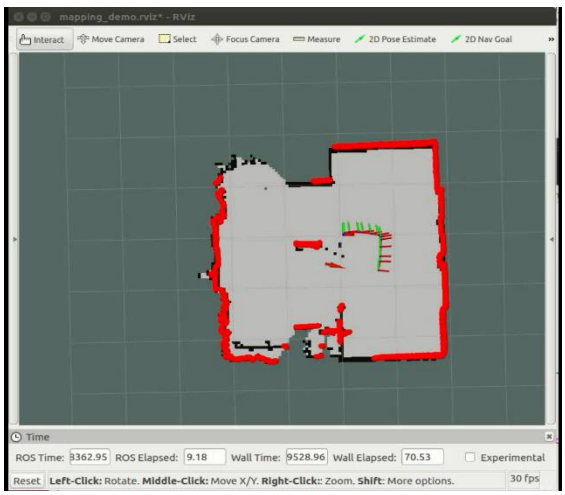

Figure 15: Demo of Linear Movement

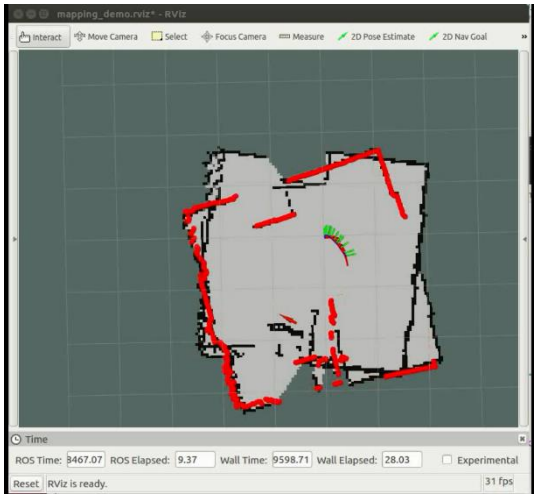

Figure 14: Demo of Circular Movement (Before)

Among those parameters, we only focus on three parameters that mainly affect the performance outcome, namely *~map_update_distance_thresh, ~map_update_angle_thresh, ~map_multi_res_levels.* The first parameter determines the threshold for performing map updates in meter while the second parameter is the

threshold in radians. The robot platform has to travel this far in meters or experience an angular change as described since the last update before a map update happens. As mentioned previously, Hector SLAM can create multi-resolution maps to optimise its SLAM performance, thus we can set the desired number of map multi-resolution grid levels via the above parameter. Since there is no formula or description on how to tune these parameters, possible combinations were adapted from the online suggestions. For the ease of convenience, developers made the

tutorial_imu.launch × static_transform.launch ×

```
<arg name="trajectory_update_rate" default="4.0"/>

<group if="$(arg bag)">
<param name="/use_sim_time" value="true"/>
<include file="$(find hector_slam_launch)/launch/static_transform.launch"/>
<!--arg name="path" default="/home/jchow/catkin_ws_fyp/rosbag" />
<arg name="play" default="Team_Hector_MappingBox_RoboCup_2011_Rescue_Arena.bag"/>
<node pkg="rosbag" type="play" name="hector_bag" output="screen" args="clock -q $(arg path)/$(arg
play)"/-->
</group>

<group unless="$(arg bag)">
<include file="$(find rplidar_ros)/launch/rplidar.launch">
  <arg name="scan_mode" value="Boost"/>
</include>

<include file="$(find mavros)/launch/px4.launch"/>
<param name="/use_sim_time" value="false"/>
</group>


<node pkg="rviz" type="rviz" name="rviz"
  args="-d $(find hector_slam_launch)/rviz_cfg/mapping_demo.rviz"/>

<include file="$(find hector_mapping)/launch/mapping_imu.launch">
  <arg name="base_frame" value="base_stabilized"/>
  <arg name="odom_frame" value="base_stabilized"/>
  <arg name="pub_map_odom_transform" value="true"/>
</include>

<include file="$(find hector_geotiff)/launch/geotiff_mapper.launch">
  <arg name="trajectory_source_frame_name" value="scanmatcher_frame"/>
  <arg name="map_file_path" value="$(arg geotiff_map_file_path)"/>
  <arg name="trajectory_publish_rate" value="$(arg trajectory_publish_rate)"/>
  <arg name="trajectory_update_rate" value="$(arg trajectory_update_rate)"/>
</include>
```

Figure 16: Sample of launch script file

parameters configurable within one single launch script file, shown as Figure 16. Table 3 shows the adequate values of these parameters used experimentally.

Table 3: Experimental values of parameters used in Hector SLAM

Parameters	**Default value**	**Best values**
map_update_distance_thresh	0.4	0.6
map_update_angle_thresh	0.9	0.9
map_multi_res_levels	2	3

After several of configuration, we successfully obtained a better SLAM performance even with rotational movement with the above parameters, shown in Figure 17.

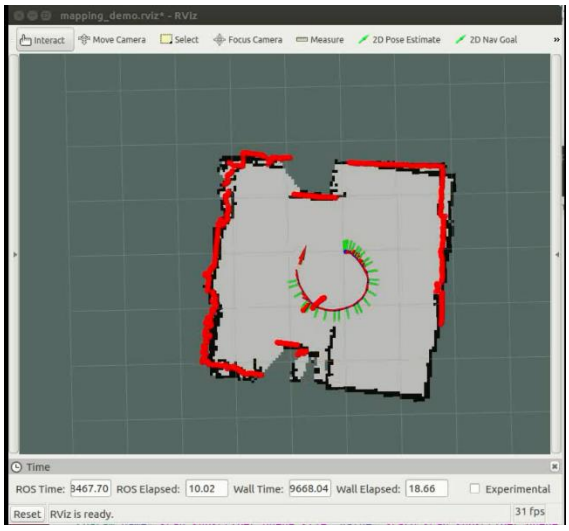

Figure 17: Demo of Circular Movement (After)

Additionally, we also discovered that Hector SLAM has a package called hector_imu_attitude_to_tf which is a lightweight node that can be used to publish the roll/pitch attitude angles reported via an IMU message to tf. In normal circumstance, aerial vehicle is expected to experience 3-dimension rotational motion (roll, pitch, yaw) compared to land-based vehicle. To improve the accuracy and reliability of Hector SLAM, we readjusted the coordinate frames setting according to Section 2.3, where we define “base_stabilized” as the robot base when its x-axis and y-axis parallel to the ground while “base_link” rotated around the roll/pitch angles. Figure 19 shows the updated TF tree and Figure 18 shows the active nodes and topics.

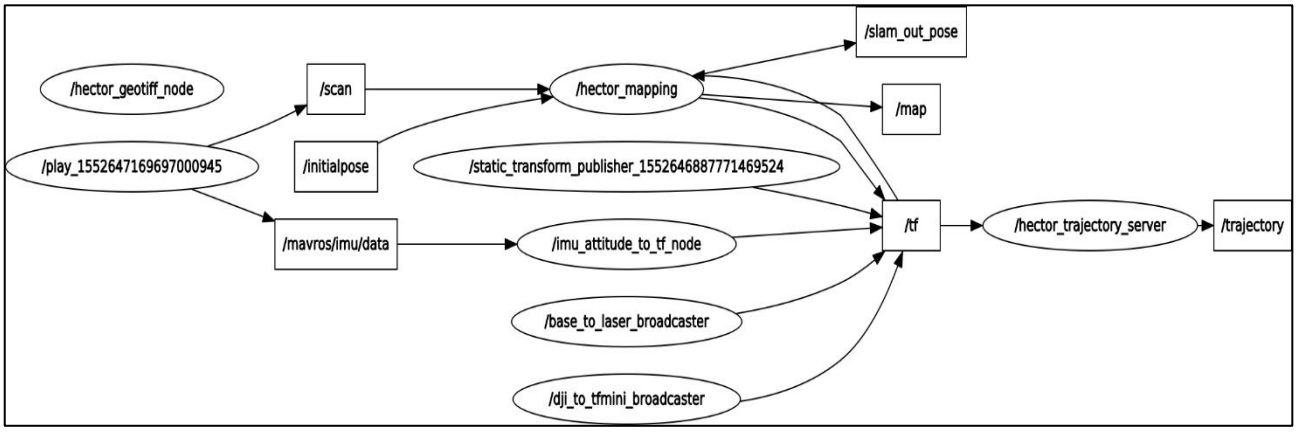

Figure 18: Dynamic graph of Hector SLAM (with IMU)

Notice that there was an additional frame “tfmini_lidar” in the TF tree, we shall explain in Section 7. In Figure 18, the IMU message was subscribed from topic */mavros/imu/data* which is generated by on-board Pixhawk (flight control unit).

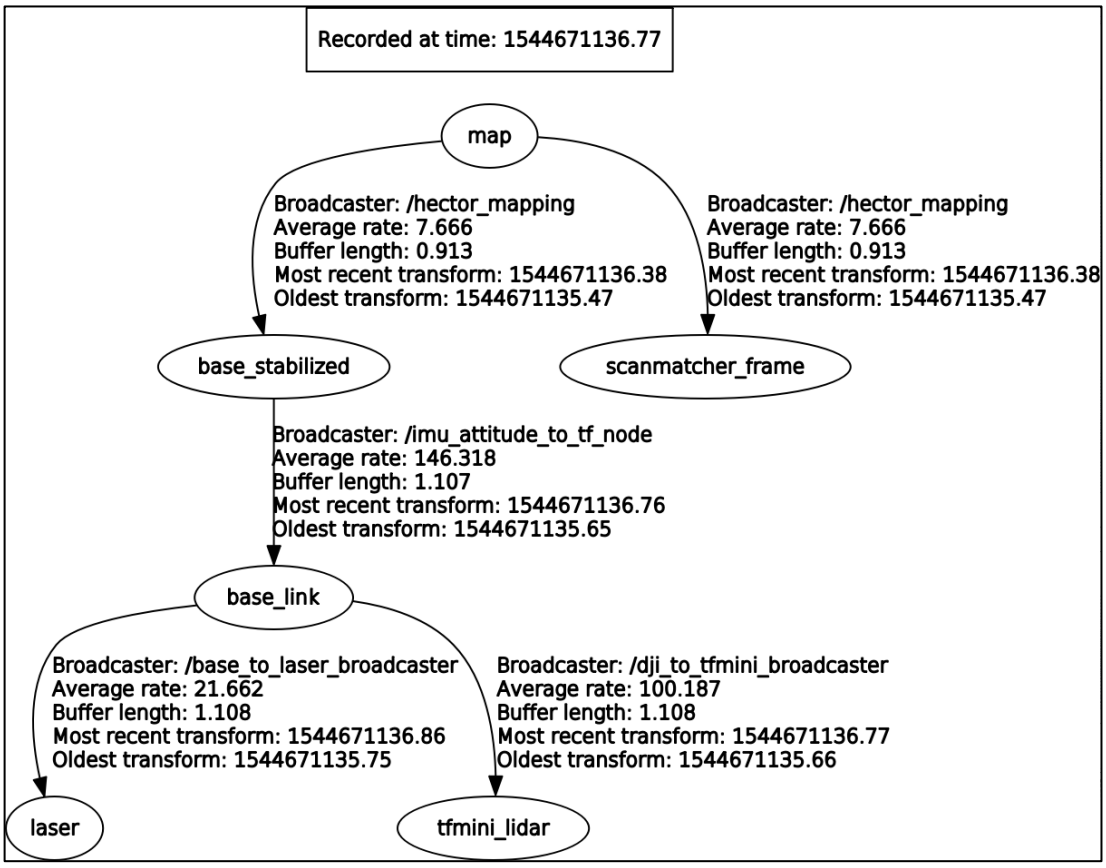

Figure 19: Hector SLAM’s TF tree (with IMU)

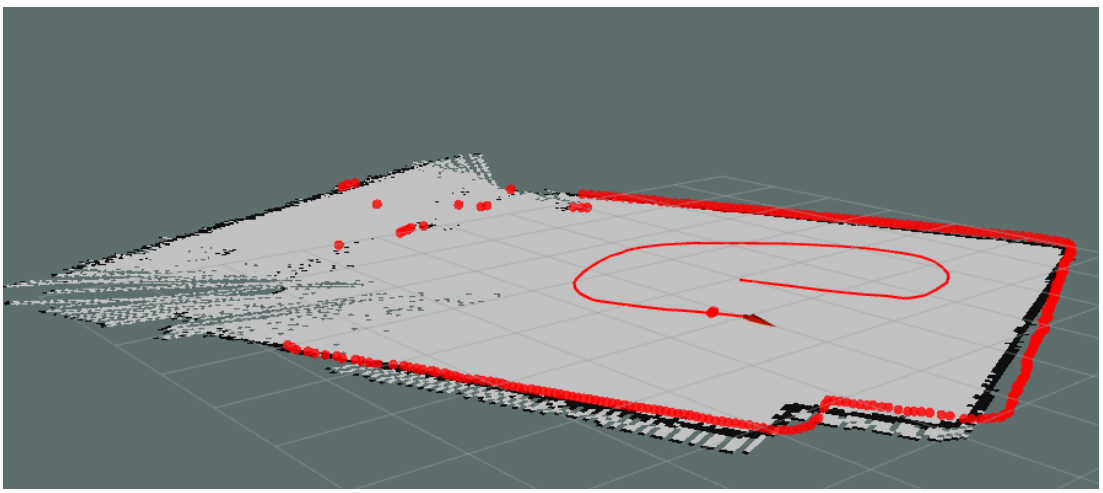

Figure 20: Planar view of Hector SLAM via Rviz (without IMU)

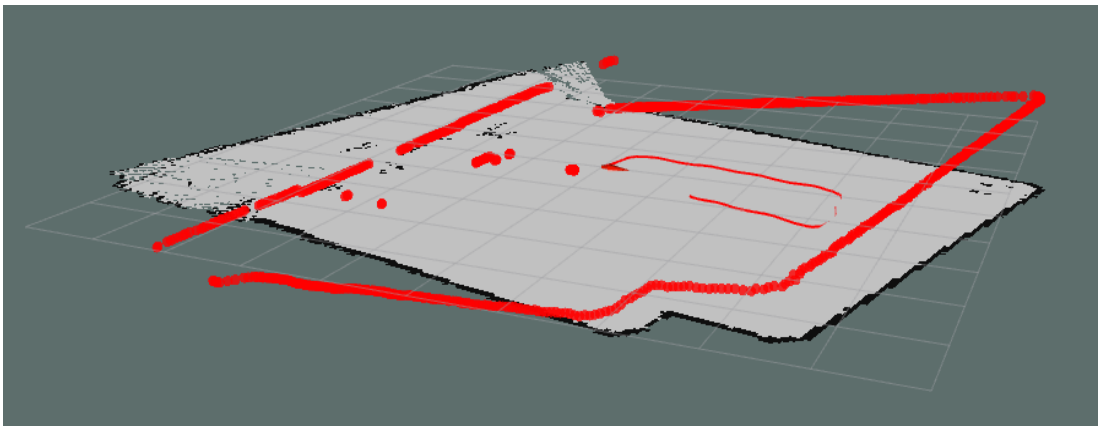

Figure 21: Planar view of Hector SLAM via Rviz (with IMU)

In Figure 20, the endpoints of laser beams were aligned with the boundary of the map in the same planar surface. In contrast, after we took the effect of roll/pitch by introducing IMU message, Figure 21 shows the real-time laser scan in more accurate way with the orientation of the body.

4.2 Gmapping

Gmapping [9], [23], [25] is a laser-based SLAM algorithm, which uses a Rao-Blackwellized Particle Filter SLAM approach. It is the most widely used SLAM method in robots worldwide, especially land-based mobile robots. Generally, the particle filter family of the algorithm requires a number of high sampling particles to obtain good results, therefore it has relatively bigger computational complexity. Also, the depletion problem associated with this method process decreases the algorithm accuracy. This happens when the elimination of a large number of particles from a sample set during the resampling step can lead to rejection/elimination of the correct hypothesis.

An adaptive resampling technique has been developed to minimize the depletion problem, since the resampling process is only performed when it is needed. Moreover, this approach takes into account not only the movement of the mobile robot, but also the most recent sensor observation with odometry information, therefore decreasing the uncertainty about the robot's pose in the prediction step of the particle filter.

As a result, the number of particles required is dramatically reduced since the uncertainty is lower, due to the quality of the laser scan matching process. In our experiment, the number of particles used was set as the default value of 30.

4.2.1 Initial setup and configuration

Similarly to the Hector SLAM package, the installation of Gmapping can refer to [25]. Within Gmapping package, only one main node is used to performance SLAM algorithm, namely *slam_gmapping*, which requires *sensor_msgs/LaserScan* message with odometry data (if possible). Since our platform is an aerial vehicle, we opted out the odometry input by changing the *odom_frame* parameter to "base_link". Again, static_transform_publisher is used to publish the tf transforms between laser to base link. Notably, in Figure 22, hector_trajectory_server was used to generate the trajectory due to the nature of Gmapping algorithm only determine the pose data in tf data format. But it is not a function of SLAM algorithm.

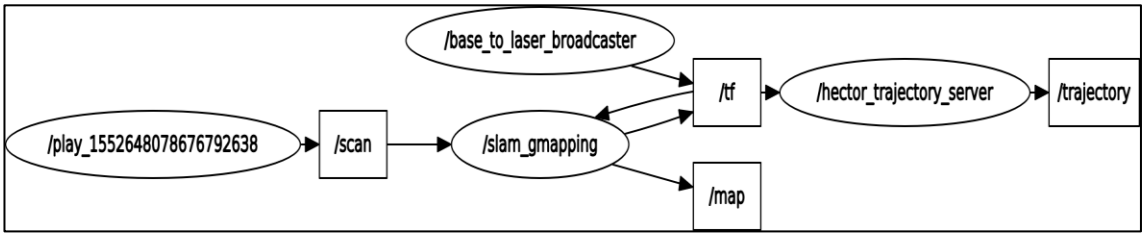

Figure 22: Active nodes and topics when running Gmapping

Firstly, without changing any default parameters, the laser scan message was directly fed into *slam_gmapping* node. Then both map and trajectory can be visualised via Rviz. In Figure 23 and Figure 24, the red solid line denotes the trajectory while the grey area is the occupied grid cells with a black boundary. Noticed that initial setup can handle translation movement but cannot deal with rotational movement well. Therefore, specific parameters: *~linearUpdate*, *~angularUpdate*, were tested experimentally to obtain a better outcome, shown in Table 4. The first parameter defines how far the robot translates before process a scan, the latter defines how far the robot rotates before process a scan.

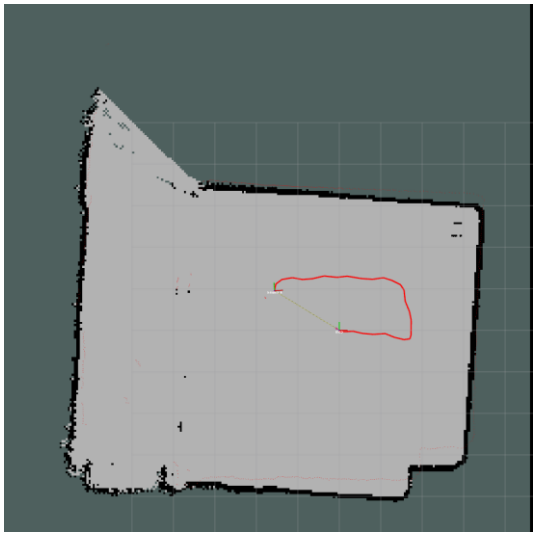

Figure 23: Initial result of translation movement

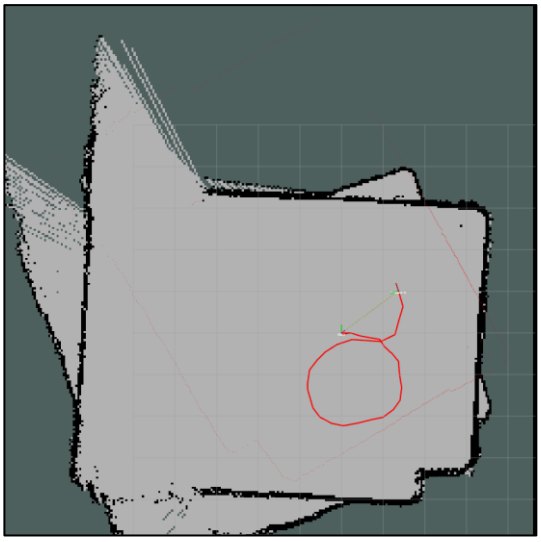

Figure 24: Initial result of circular movement (before)

Table 4: Experimental values of parameters used in Gmapping

Parameters	**Default**	**Best value**
linearUpdate	1.0	0.9
angularUpdate	0.5	0.6

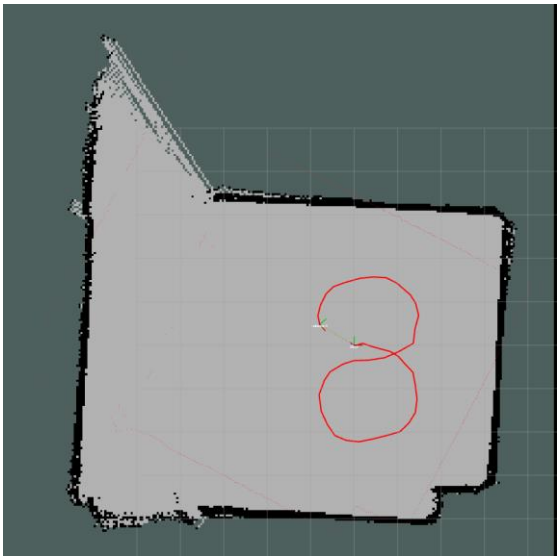

Figure 25: Initial result of rotational movement (after)

After tuning the parameters, we obtained much better SLAM performance compared to the previous one, shown in Figure 25. Although there was a minor mapping issue during rotational movement, this is the best setup we obtained from previous tuning.

4.3 Cartographer

Cartographer [26] is a system that provides real-time simultaneous localization and mapping (SLAM) in 2D and 3D across multiple platforms and sensor configurations. It is an open source library, developed by Google since 2016, which is also the latest SLAM algorithm. Worth to mention, Google Cartographer does not require a particle filter algorithm for mapping. It overcomes the issue of error accumulation during long iterations by pose estimation against a recent submap.

In 2D SLAM, Cartographer supports running the correlative scan matcher, which is normally used for finding loop closure constraints with a submap (at the best-estimated position) referred to as frames. In detailed, scan matching occurs at a recent submap, therefore only it depends on recent scans. After each submap is finished, that is no new scans will be inserted anymore, it automatically checks all submaps and scans for loop closure. A scan matcher starts to find the scan in the submap if the scans and submaps are close enough based on current pose estimates [23].

The paper [17] describes the conversion process from a scan into a submap. Submaps generated are presented in the form of probability grid point which contains all the endpoint of beams that are closest to that grid point. Whenever a scan is inserted into the probability grid, hits and misses are computed. For every hit, all nearest grid points are inserted into the hit set, but for every miss the grid point that intersects one of the rays between the scan origin and each scan point, is inserted in the miss set. Cartographer uses the Ceres scan matching approach to increase the accuracy of scan pose in the submap.

4.3.1 Initial result and configuration

Different from other SLAM packages, Cartographer has its own documentary and installation process mentioned in [27]. Unlike previous packages which all parameter can be directly configured under launch script files, Cartographer's ROS integration top-level options must be specified in the Lua configuration file. Firstly, we set *~tracking_frame, ~odom_frame* to "base_link" and *~published_frame* to "laser" since no odometry device is used in our platform. IMU message is optionally supported in 2D SLAM, therefore we also use it as input. Additionally, we enabled the setting *~publish_frame_projected_to_2d* because we want to restrict it to be a pure 2D pose. This prevents potentially unwanted out-of-plane poses in the 2D mode that can occur due to the pose extrapolation step.

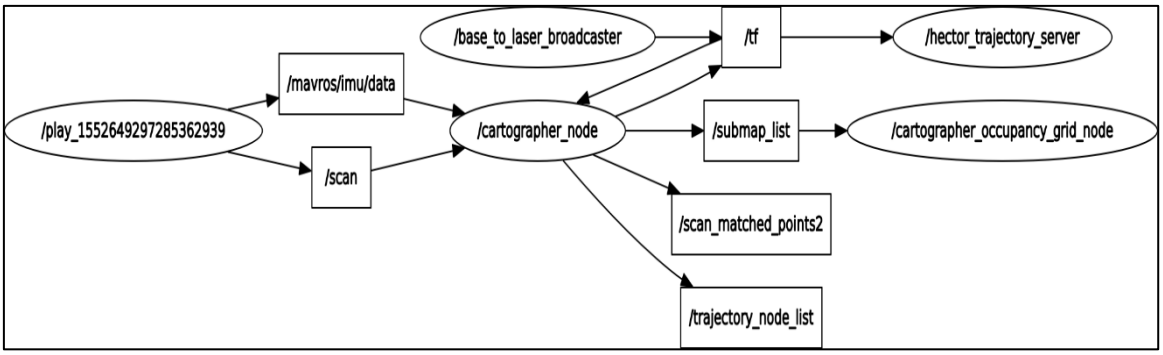

Figure 26: Active nodes and topics of Cartographer

In Figure 26, we can see that /base_to_laser_broadcaster is used to publish tf frame from the laser to base link while /hector_trajectory_server is to convert the pose information in the form of tf message to a trajectory. Figure 27 simply shows the tf frames of Cartographer.

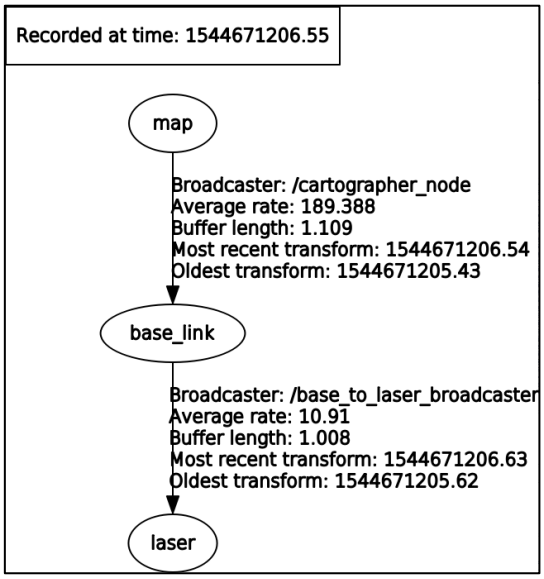

Figure 27: TF frames of Cartographer

Without further tuning Cartographer, we cannot obtain reliable SLAM performance during rotational movement (same as other SLAM algorithms), shown in Figure 28. Since Cartographer continuously generates submaps with the most recent scan, the map will overlap each other if it fails to match with the previous one.

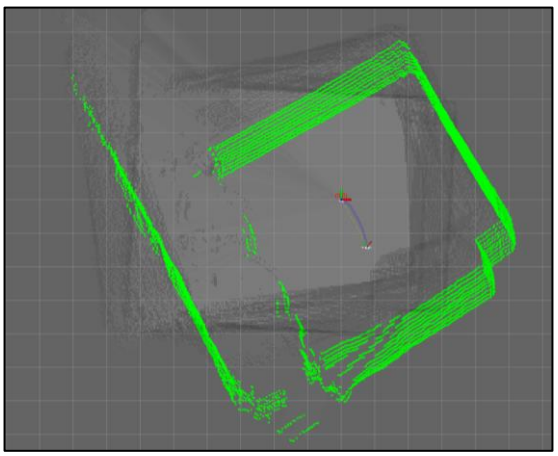

Figure 28: Initial result of Cartographer

Unfortunately, Cartographer has many parameters which affect each other. After several trials and researches, we identified the most important parameter which significantly affects the outcome of mapping quality, namely *~num_subdivisions_per_laser_scan*. It defines the number of point clouds to split each received laser scan into. Subdividing a scan makes it possible to unwarp scans

acquired while the scanners are moving. Table 5 shows the experimentally values tested to obtain better SLAM performance of Cartographer.

Table 5: Experimental values of parameter used in Cartographer

Parameter	Default value	Best value
num_subdivisions_per_laser_scan	1	12

Finally, we successfully obtained a satisfactory result from Cartographer SLAM algorithm in Figure 29. The green dotted line is the current laser scan, the white occupied area is the map generated.

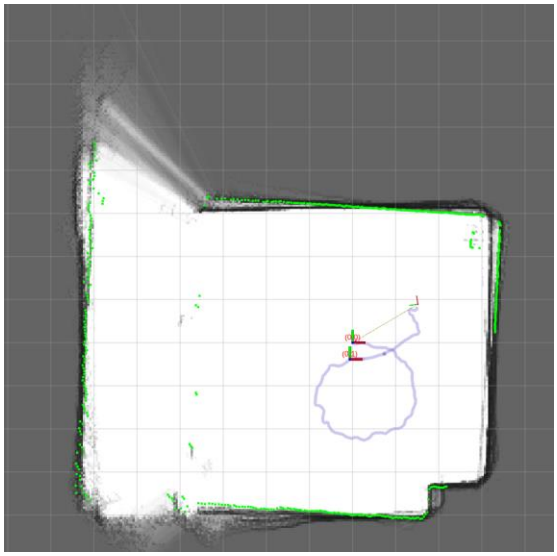

Figure 29: SLAM results of Cartographer after tuning

4.4 An overview of SLAM techniques

Before moving on to the offline experiment, a simple comparison of each SLAM technique is shown in Table 6. In common, they all require a 2D LIDAR sensor to perform a laser feature scan in order to build a map and localise itself within the working area. IMU device serves as auxiliary sensor input which detects the rotational motion (RPY) of the aerial vehicle, to improve the reliability of laser scan.

Table 6: Comparison of three different SLAM techniques/configuration

SLAM approaches	**Sensor Requirement**	**Odometry**	**IMU**
Hector SLAM	Only 2D LiDAR sensor	Not required	Optional
Gmapping	Mainly rely on the odometry	Required	Not required
Cartographer	Multiple sensors for a better performance	Optional	Optional

5. OFFLINE EXPERIMENTS

5.1 Design of the data collection experiment

After the implementation of the SLAM algorithms, an experimental platform of LIDAR/IMU integrated SLAM was designed and conducted to obtain the results of each performance. A 3D printed platform was designed and fabricated to mount RPLidar and Pixhawk (mainly served as IMU data publisher) in a top-down orientation which was similar to the actual arrangement of the on-board UAV platform. Figure 30 shows the initial design of the hand-hold bracket to mount the LIDAR sensor with Pixhawk. The reason of the 3D printing technique was selected instead of metal machining is because of lightweight, faster and easily fabricate in school (machine ready to use in Robotic Research Centre, NTU).

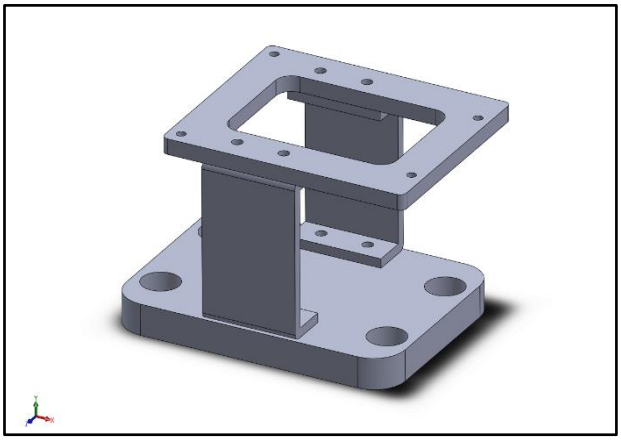

Figure 30: Assembly drawing in Solidwork

The overall data collection system is shown in Figure 31. The UBUNTU 16.04 Kinetics, an open source Linux platform, runs on a personal laptop with Intel® Core™ i5-4210U CPU@1.7GHz quad cores, on top of the robot operating system (ROS) in Ubuntu 16.04. ROS Kinetic is mainly served as the software implementation platform to run respective nodes that manage the connection of LIDAR and Pixhawk, receive data and publish the pose estimation.

The indoor experiment was conducted in the Motion Analysis Laboratory (N3-B2C-05A) which equipped OptiTrack Motion Capture (Mocap) systems, allow vehicles to navigate when a global position source is unavailable. Mocap uses a system of off-board cameras to get vehicle pose data (position and attitude) in a 3D space (i.e. it is an external system that tracks the position of reflective markers and tells its pose). Figure 32 shows the internal structure of the Mocap lab. The ground truth data was received and recorded via a customised node that subscribes to a specific topic published by Mocap systems.

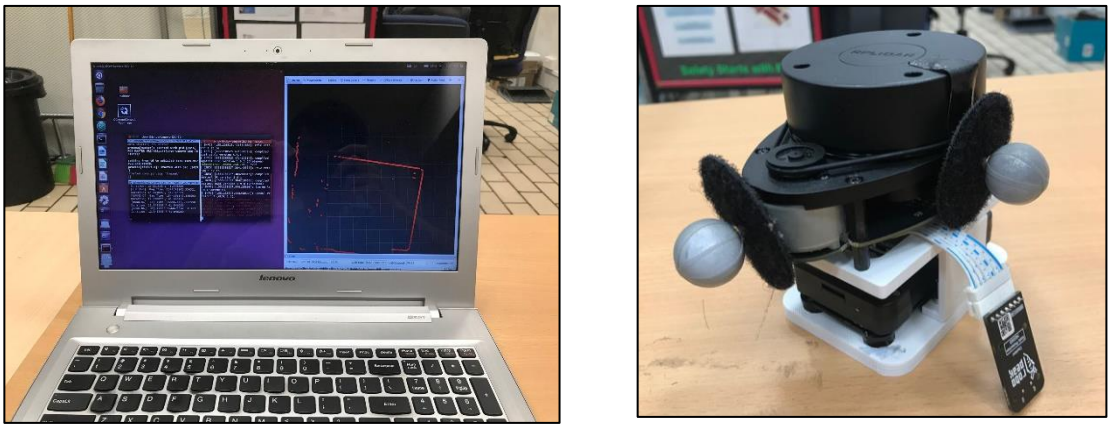

Figure 31: Main processing unit on the left, RPLidar and IMU mounted onto fabricated structure with reflective markers attached

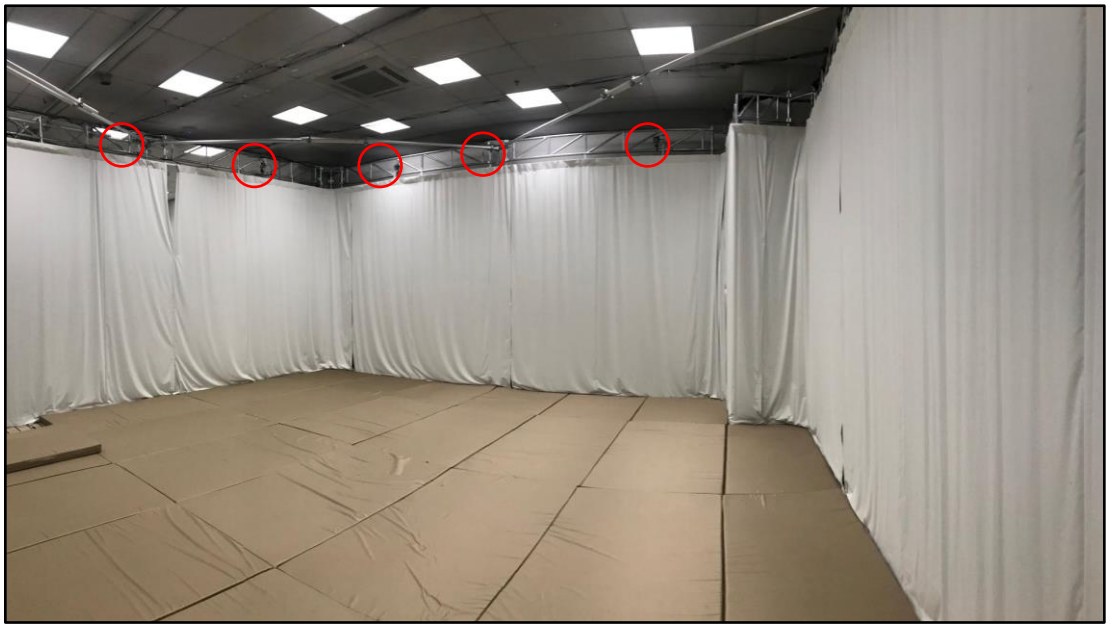

Figure 32: Indoor Experiment Environment, red circle denotes external cameras to detect the reflective markers

During the data collection in the indoor experimental process, the sensors and processor system were hand-held. The reason that a drone is not used to fly the system is that in the current stage our focus will be on determining the best SLAM algorithm before we can proceed to build a functional drone system. There are two trajectories with 2 different speeds (normal walking speed and fast walking speed), with the starting point set as the origin of the Mocap Coordinates system (the center of working area).

1. **Linear Rectangular Trajectory** – Starting from the origin, moving straight and then follow the rectangular path until it ends at the starting point. Throughout the trajectory, the heading of the platform was purposely remained facing West (X-direction), thus filtered the potential noise due to the large yawing angle. Figure 33 shows the geographical layout of the experimental area and the trajectory of the first scenario.

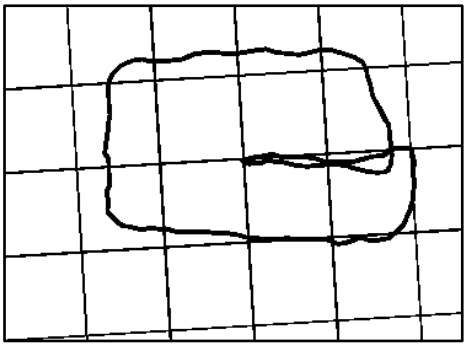

Figure 33: Linear Rectangular Path

2. **Circular Figure 8 Trajectory** – Starting for the origin, moving along figure 8 with the heading aligned with the trajectory, ending at the starting point. Figure 34 shows the trajectory of the second scenario.

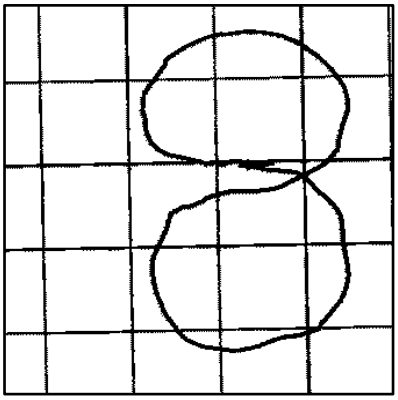

Figure 34: Circular '8' path

5.2 Validation of the Position Estimation of Different Algorithms

Initially, the Hector SLAM technique was launched to plot the trajectory on the screen in real-time while the ground truth pose was determined by Mocap systems. Specific topics were monitored and recorded with the rosbag functions, so that it can be replayed and feed into another SLAM algorithm with the input conditions. These topics were: /mavros/imu_data, /map, /slam_out_pose, /scan, /Robot_1/pose.

Additionally, these experiments were carried out again but with faster moving speed to test the capability of SLAM algorithms.

5.3 Pose Experiment

As mentioned earlier, essential data were recorded in *bag* format to be replicate the experimental conditions for the alternative techniques, namely Cartographer, and Gmapping. Both algorithms were tuned to optimum parameter settings (refer to Chapter 4) and fixed for all scenarios to compare the robustness of the SLAM system fairly.

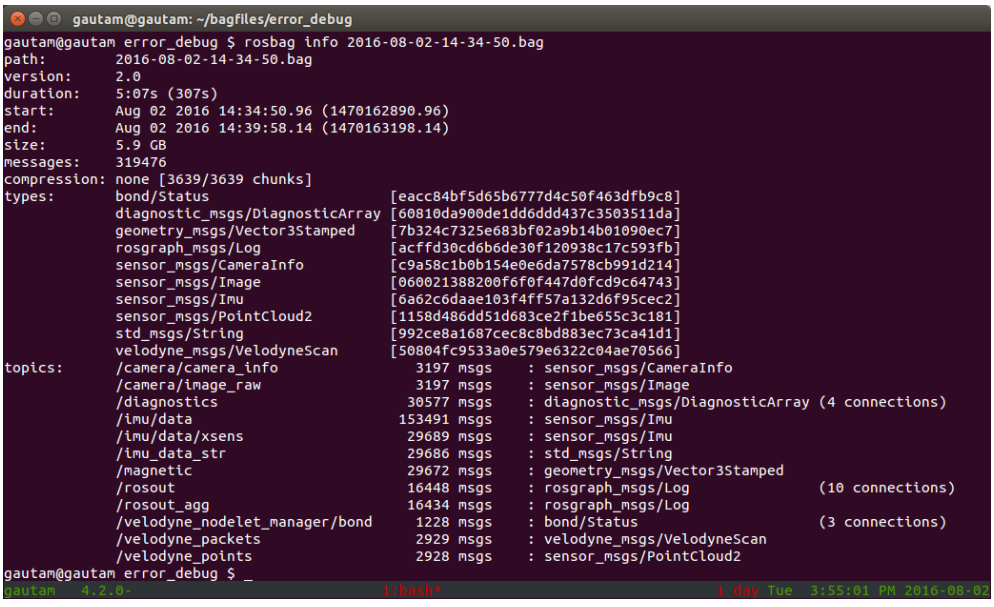

Figure 35: Rosbag Info

6. RESULT AND DISCUSSION

6.1 Performance analysis of the SLAM algorithm

A group of typical localization results in the indoor experiment is illustrated from the following aspects: trajectories, displacement of X-axis, displacement of Y-axis and yaw angle. In this section, MATLAB R2018b was used to analyse and plot the graphs because its compatibility to work with *rosbag* format within the ROS network and to plot the trajectories. Figure 36 shows the IDE of MATLAB and scripts to extract relevant data.

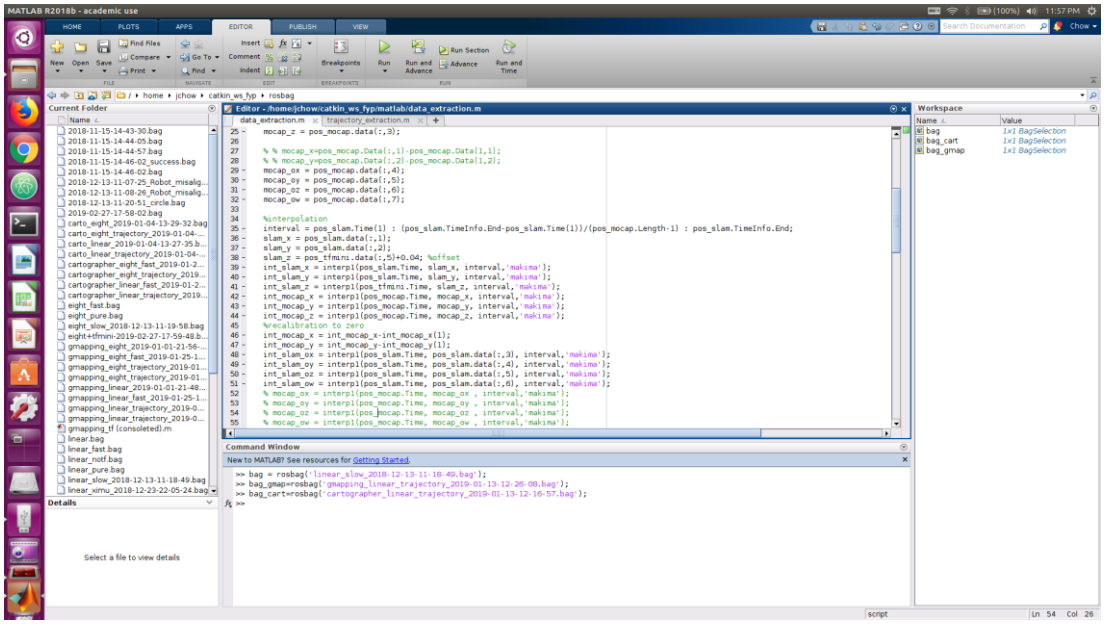

Figure 36: User Interface of MATLAB

6.1.1 Normal moving speed

Scenario 1: Linear Trajectory (Time of flight: 42s)

A ground of typical localization results in the indoor environment is illustrated from Figure 37 to Figure 44**Error! Reference source not found.**. All trajectories were plotted on the same graph to have a clearer comparison of several algorithms. In Figure 37, we can see that the localization results from Hector SLAM and Gmapping are comparably smoother than the outcome of cartographer which shows fluctuation and jerkiness. This result is possibly due to the characteristic of cartographer that

fuses multiple sensor data, but there is only one LiDAR sensor in our case. On the other hand, it is difficult to tell either Hector SLAM or Gmapping is more accurate only from observation.

According to our observation, localization performance of lased-based algorithm is affected by three variables, namely: the performance of the LIDAR sensor, the feature extraction within the environment and the matching algorithm.

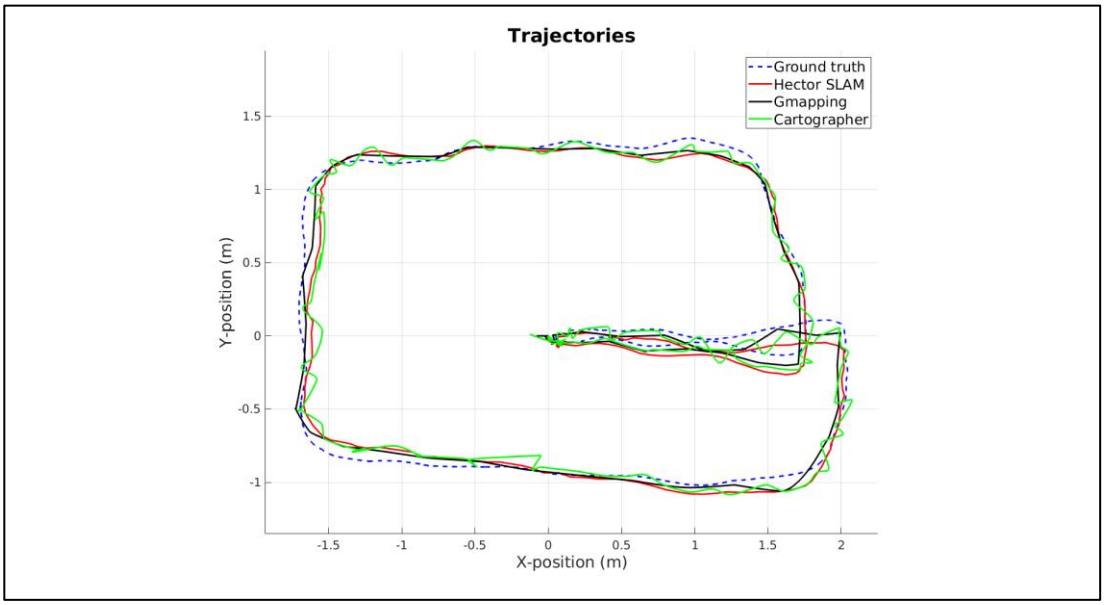

Figure 37: Comparison of trajectories with ground truth (blue dot line)

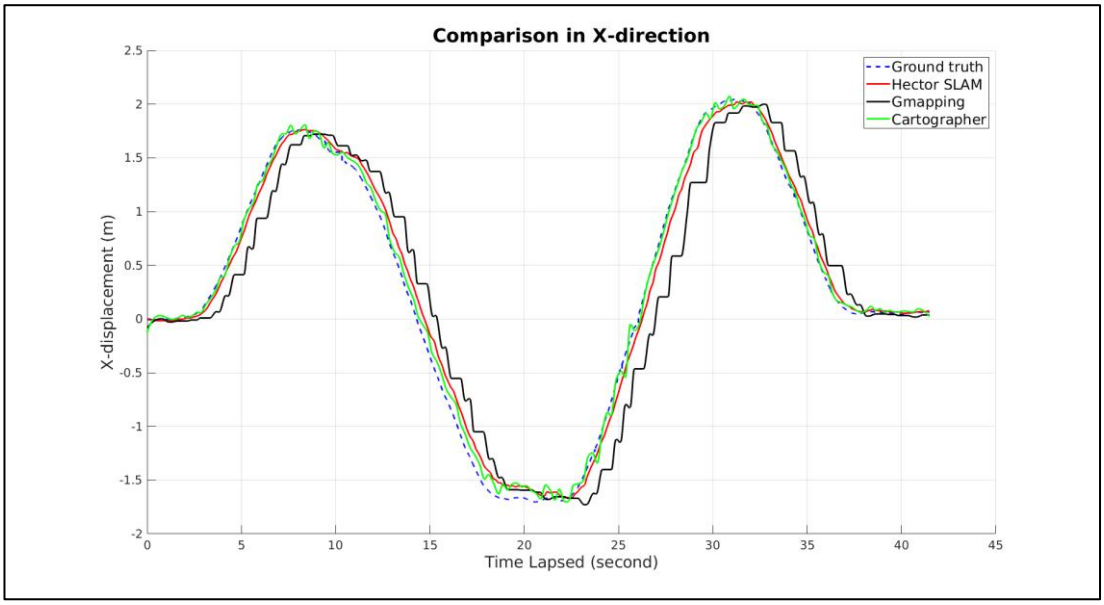

Figure 38: Displacement in X-direction

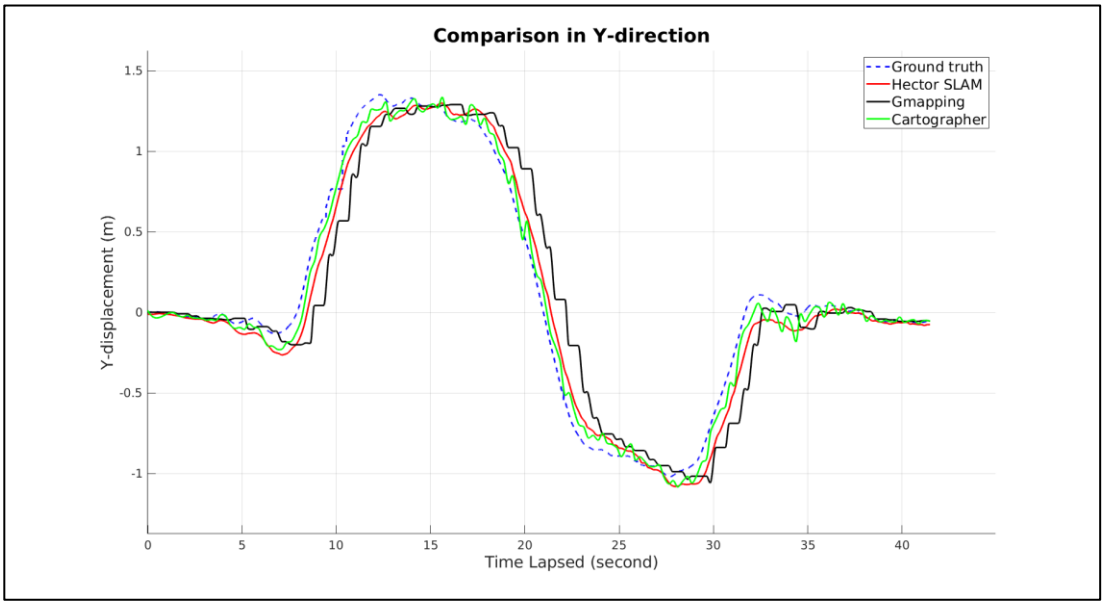

Figure 39: Displacement in Y-direction

From Figure 38 and Figure 39, we can observe that there was a significant time delay in the Gmapping algorithm, which also shows a discrete movement. This behaviour is mainly because of the slower updating frequency of the Gmapping algorithm compared to others. On the other hand, both Hector SLAM and Cartographer give reliable positioning results, but the latter algorithm shows fluctuation during each peak (while changing in direction). The same observation was obtained from orientation (Figure 40), Gmapping has a time delay increasing along the simulation

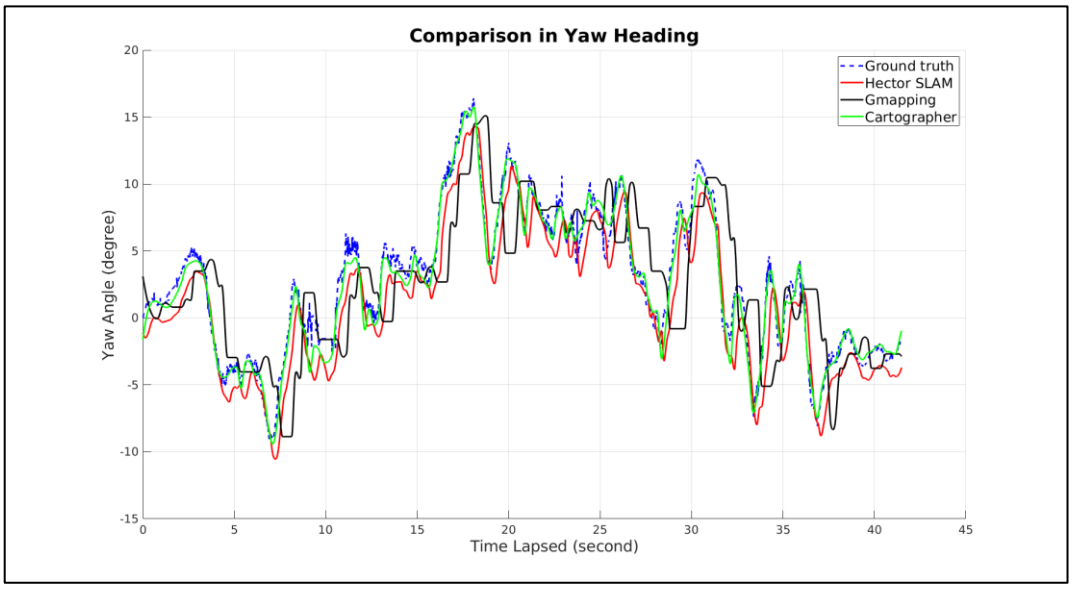

Figure 40: Yaw heading vs. Time lapsed

time. In conclusion, Hector SLAM has the best performance among three SLAM techniques based on smoothness, accuracy and time response.

Scenario 2: Circular Trajectory (Time of flight: 33s)

The same analysis was performed in the second scenario. The circular path was used to test the robustness of each SLAM technique when facing large changing in heading angle (yaw). The results are shown in Figure 41 to Figure 44. Once again, Hector SLAM had better results compared to cartographer which showed fluctuation

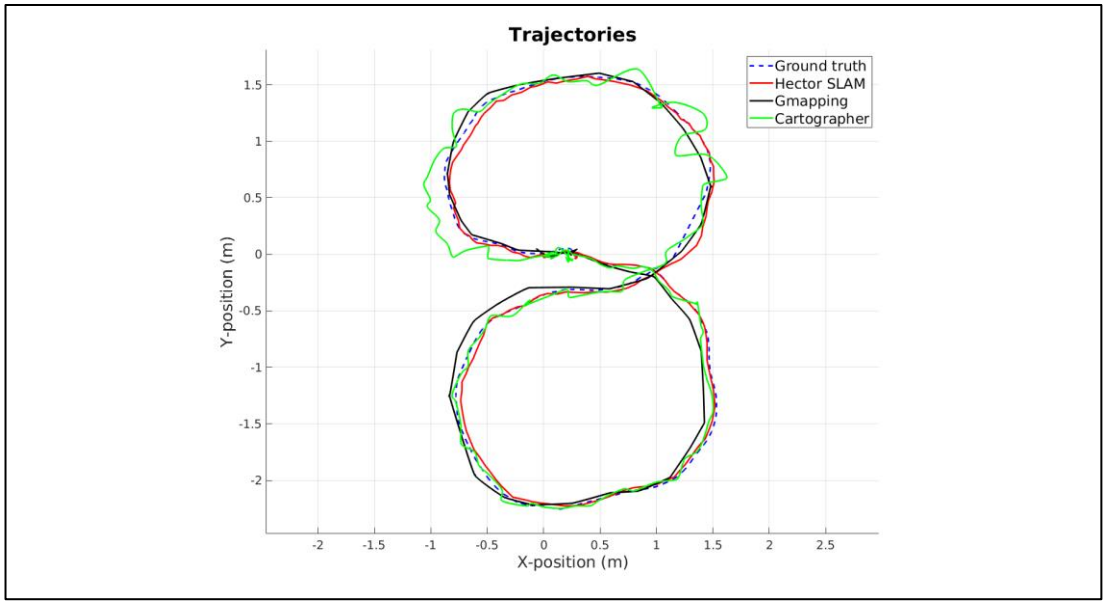

Figure 41: Comparison of trajectories

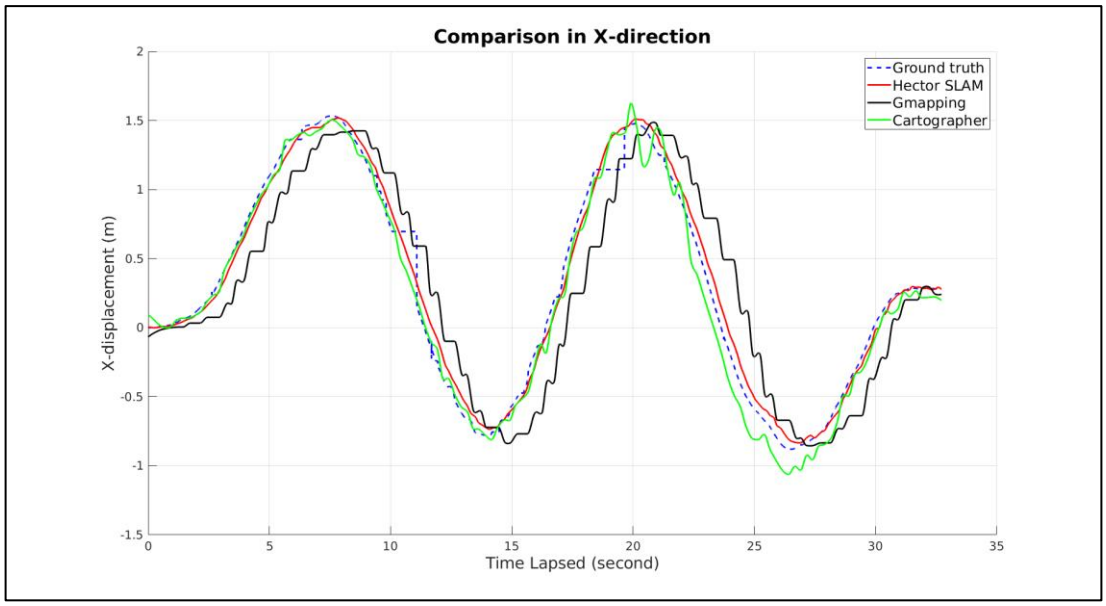

Figure 42: Displacement in X-direction

and Gmapping with high time delay. Notably, the time shift of Gmapping will be larger when simulation time increases. We can also observe that there were two sudden peaks from the yaw estimation from both Gmapping and Cartographer techniques.

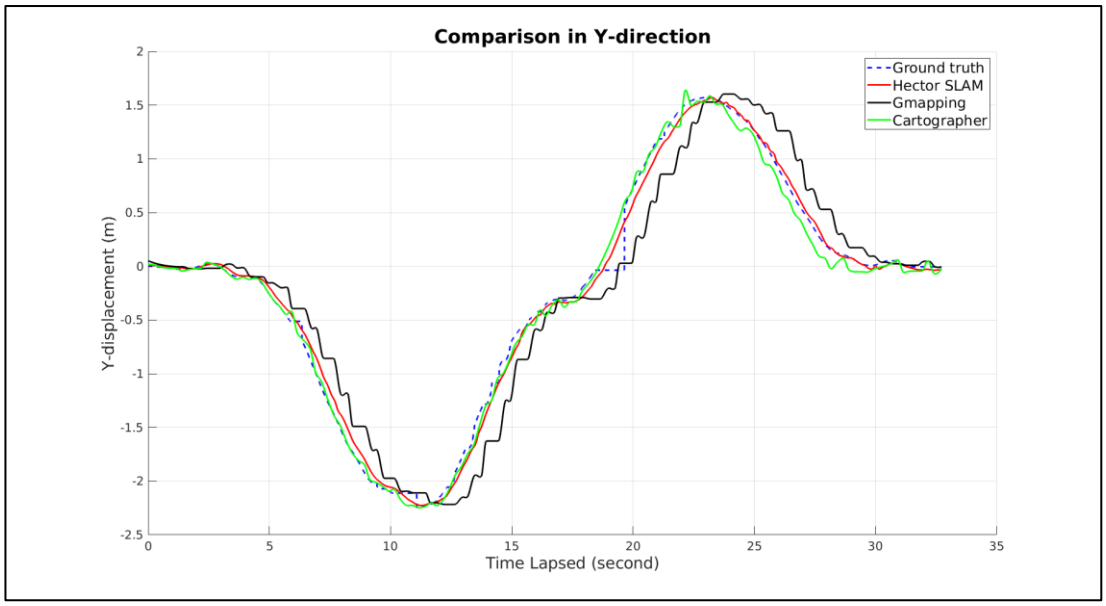

Figure 43: Displacement in Y-direction

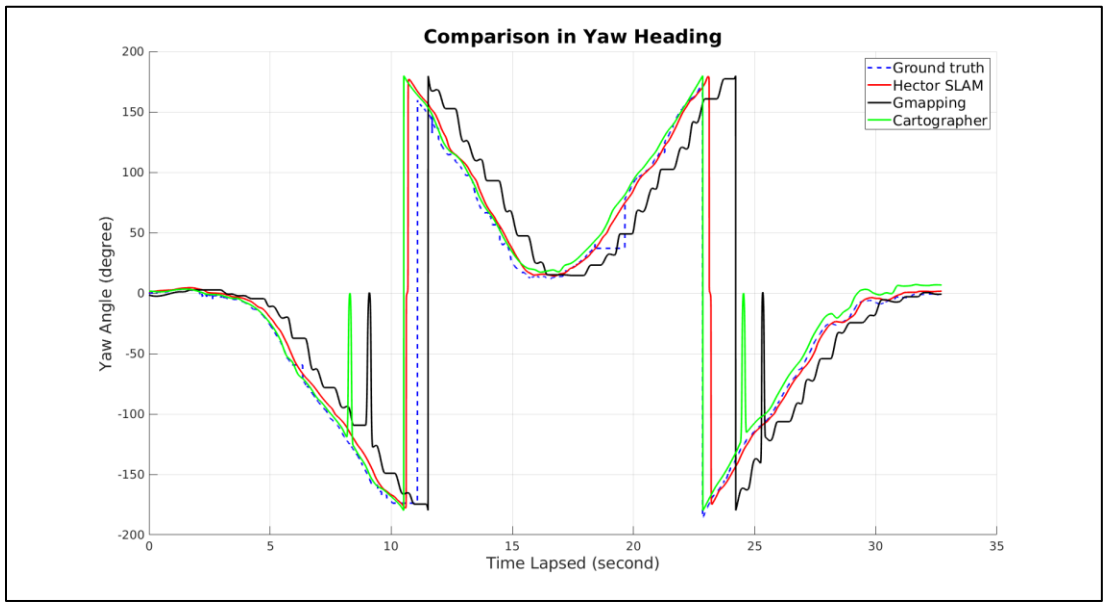

Figure 44: Yaw heading vs. Time lapsed

To further analyse the results, we use Root Mean Square formula (eq. 1) to determine the accuracy of each technique compared to ground truth. All calculations were calculated via MATLAB script and tabulated for comparison. Firstly, we examined the results of Scenario 1, then followed by Scenario 2.

$$RMSD = \sqrt{\frac{\sum_{i=1}^{n}(x_{truth} - x_{est})^2}{n}} \quad (1)$$

where x_{truth} is the ground truth value and x_{est} is the output from SLAM algorithm.

From Table 7, Hector SLAM has a smaller error ranges from 0.0539 to 0.0751 m position and 2.1945 degrees in yaw angle. Followed by cartographer technique which has an average position error less than 0.0787 m and 2.3522 degrees in yaw angle. In the second experiment (Table 8), Hector SLAM also showed a better accuracy, which was 0.0781 m error in X-direction, 0.0704 m in Y-direction. Unsurprisingly, Gmapping had the least accuracy, up to 0.3297 m error in Y-direction.

Table 7: RMS error in scenario 1 (normal speed)

	Hector SLAM	**Gmapping**	**Cartographer**
X-direction (m)	**0.0539**	0.3292	0.0787
Y-direction (m)	0.0751	0.2278	**0.0679**
Yaw angle (degree)	**2.1945**	4.5925	2.3522

Table 8: RMS error in scenario 2 (normal speed)

	Hector SLAM	**Gmapping**	**Cartographer**
X-direction (m)	**0.0781**	0.3297	0.1266
Y-direction (m)	**0.0704**	0.2733	0.0930
Yaw angle (degree)	44.0383	90.9150	39.8205

Noted that, all techniques have a large error in yawing angle, which is above 39 degrees. This is mainly due to the time shift and huge heading changes (from -180 degrees to 180 degrees) during the circular path. Therefore, we should not overemphasize on this value for accuracy estimation.

6.1.2 Fast moving speed

Scenario 1: Linear Trajectory (Time of flight: 24s)

A similar experiment was conducted but with faster moving speed to test the robustness of each algorithm. From Figure 45, Gmapping was not working properly and its mapping quality was totally unacceptable (will be shown in the mapping section). Again, Hector SLAM had the best pose estimation where else cartographer generated fluctuated trajectory. Notably, there was a few sudden discrete values for ground truth observed from Figure 46 and Figure 47. This is mainly due to the signal interruption in the motion capture lab. Since it is not significant, we can neglect the repercussion in RMS error calculation.

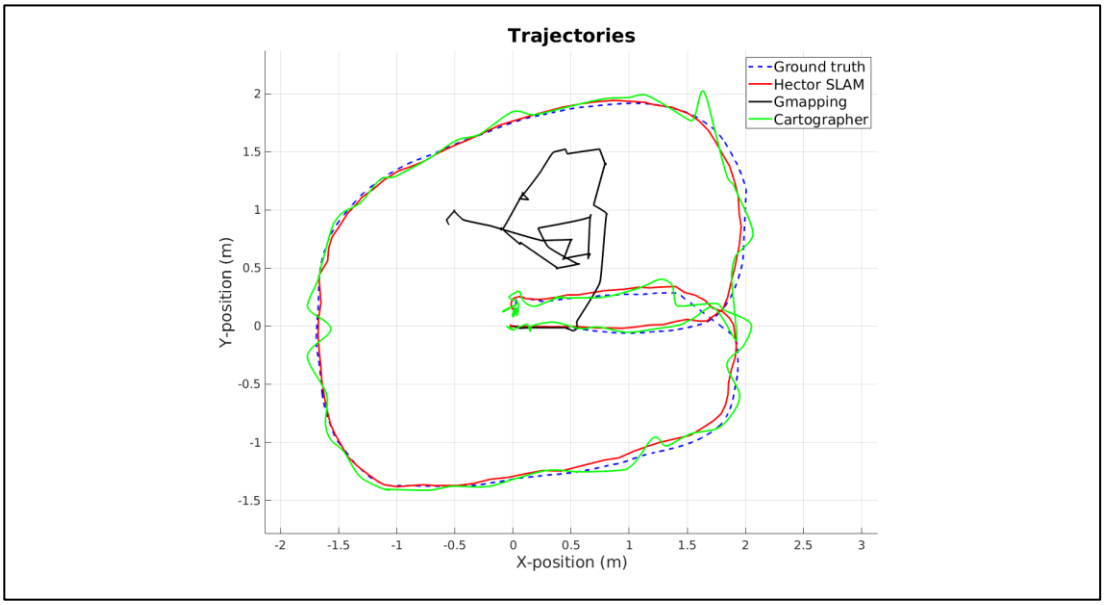

Figure 45: Comparison in Trajectories

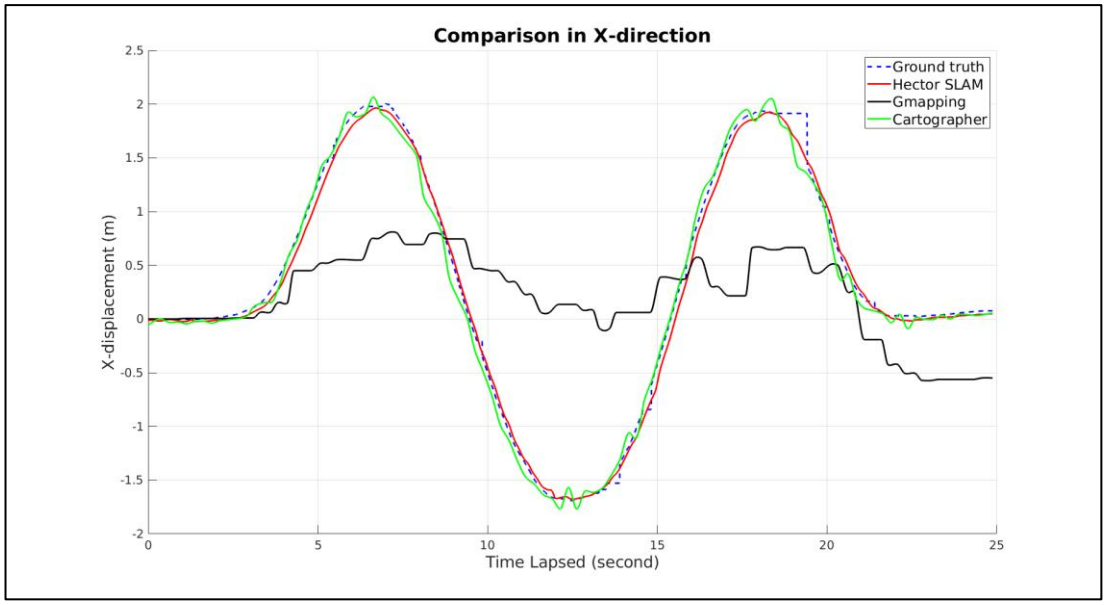

Figure 46: Displacement in X-direction

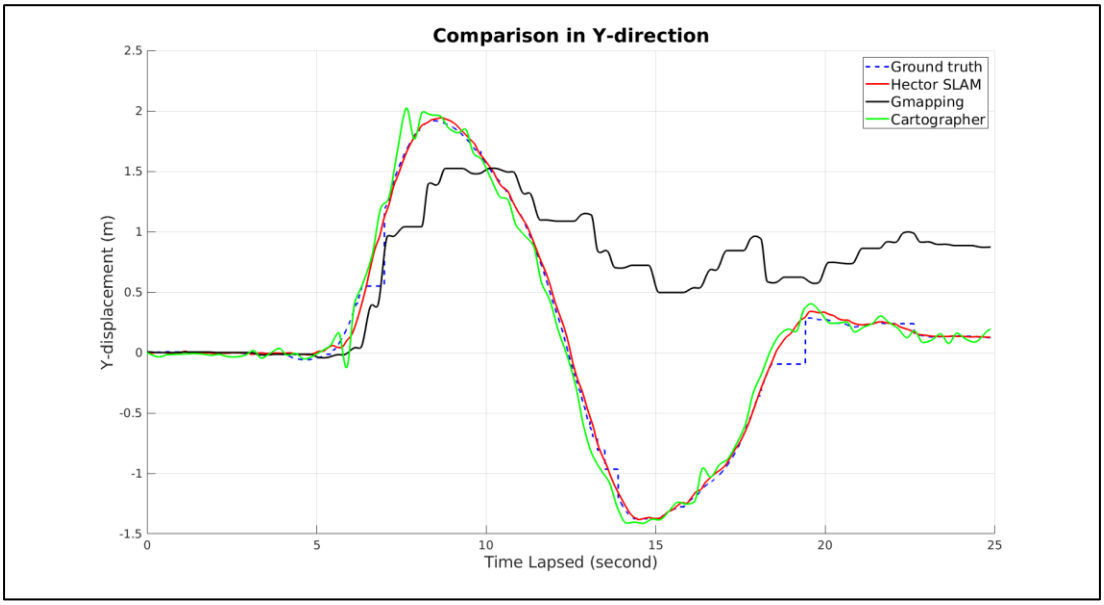

Figure 47: Displacement in Y-direction

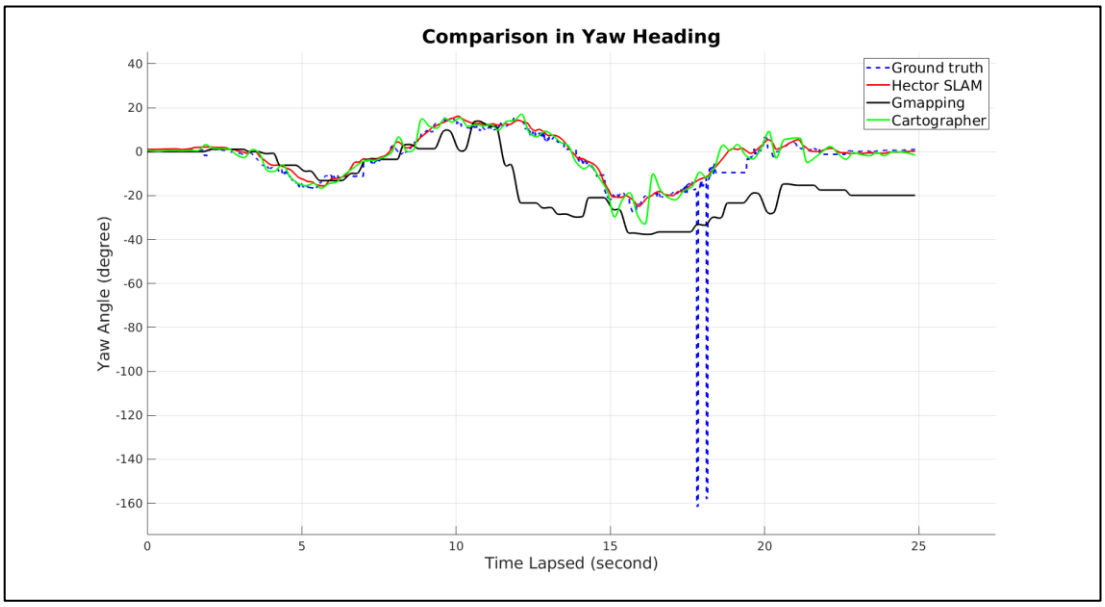

Figure 48: Yaw heading vs. Time lapsed

In Figure 48, there is unusual behaviour from ground truth value (at 18sec) because of the interrupted connection from Mocap system.

Scenario 2: Circular Trajectory (Time of flight: 23s)

In Figure 49, Gmapping (black line) failed to performance SLAM properly while Cartographer generated inconsistently and fluctuated with the green line shown. Only Hector SLAM's trajectory (red line) had the closest trajectory compared to ground

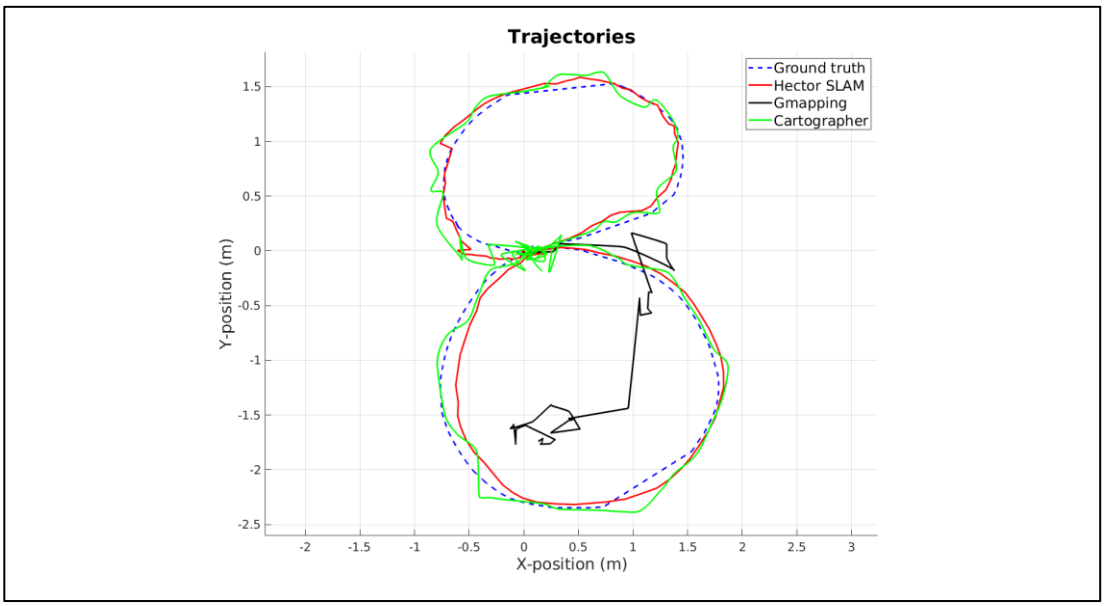

Figure 49: Comparison of trajectories

truth values (blue dotted line). Further comparisons in X and Y direction are shown in Figure 50 and Figure 51 respectively.

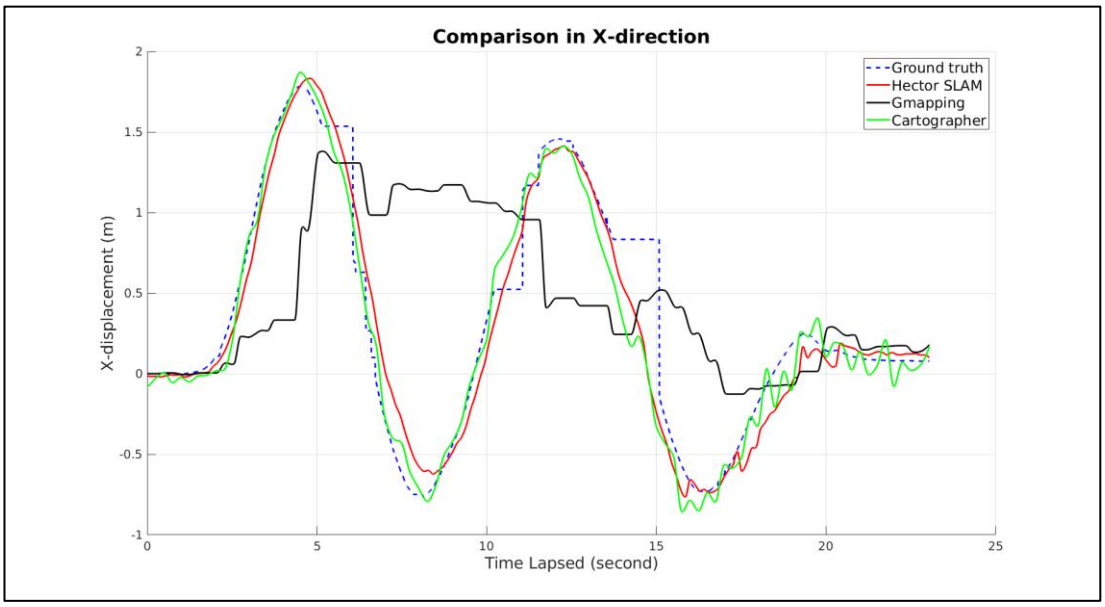

Figure 50: Displacement in X-direction

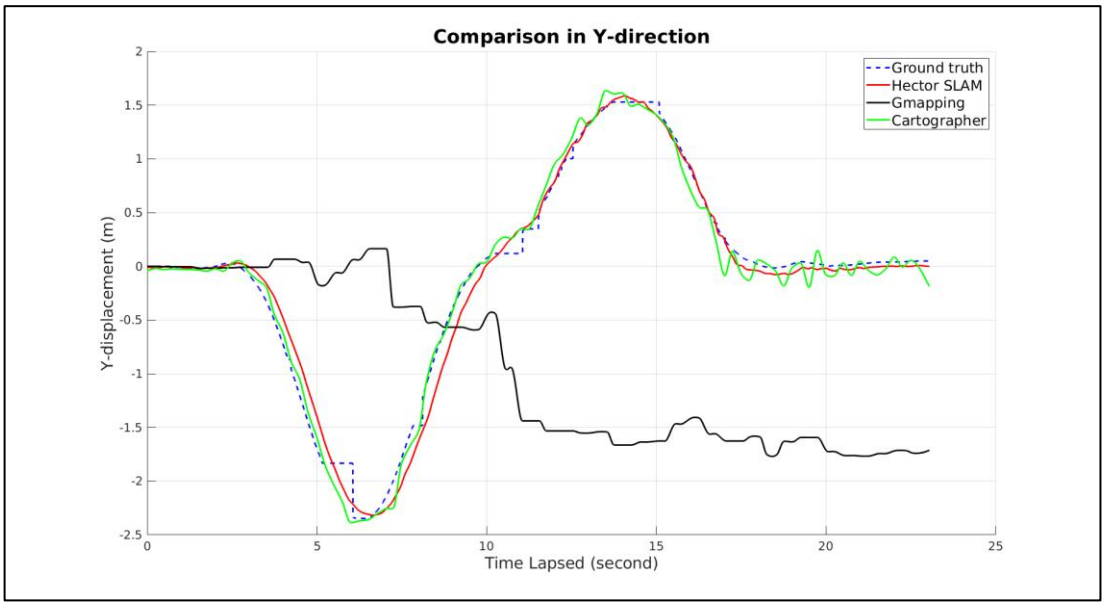

Figure 51: Displacement in Y-direction

Notably, Cartographer experienced sudden spike twice in determining the heading of our robot when around -120 degree of yaw, shown in Figure 52 (green line). There is also abnormal behaviour from ground truth value during the experiment, this might

be due to the blockade of reflective markers by the operator. Another possible explanation is that the ground truth data was interrupted which caused a delayed update at some instants.

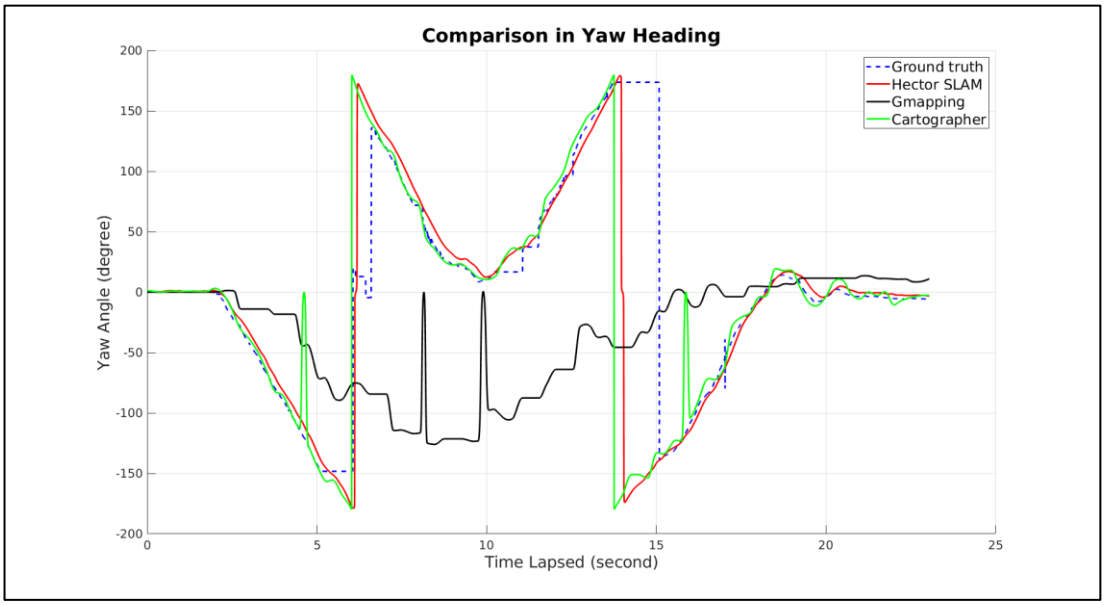

Figure 52: Yaw heading vs. Time lapsed

Repeatedly, RMS error of each algorithm in X, Y direction and yaw angle were tabulated. From Table 9, Hector SLAM gave us an overall average error of 0.08105 m in position. From the graph plotted previously, the Gmapping algorithm was totally unacceptable as it failed to perform SLAM in a faster-moving speed. This results in the highest error more than 1 m in position. In the second experiment, Hector SLAM continuously gave much reliable localization performance, followed by the Cartographer with an average RMS error of 17.92 cm in position.

Table 9: RMS error in scenario 1 (fast speed)

	Hector SLAM	**Gmapping**	**Cartographer**
X-direction (m)	**0.0783**	1.0062	0.1036
Y-direction (m)	**0.0838**	0.9865	0.1259

Yaw angle (degree)	**9.4032**	17.4318	9.7531

Table 10: RMS error in scenario 2 (fast speed)

	Hector SLAM	**Gmapping**	**Cartographer**
X-direction (m)	**0.2296**	0.8379	0.2365
Y-direction (m)	0.1372	1.8959	**0.1219**
Yaw angle (degree)	74.9432	110.9804	84.0971

Due to the unreliable RMS error in yaw angle (time delay further increases the difference with ground truth value), the evaluation will not be further discussed. Despite that, computational time was evaluated as one of the comparison properties among the three algorithms. Since each of them publishes pose information in tf format in common, we can use rostopic monitor to determine the update frequency (Hz) and processing time (ms). As shown in Table 11, both Hector SLAM and Cartographer had updating frequency up to 290 Hz while Gmapping only had half of their values. Although cartographer has the fastest computation time of 3.4ms, the difference compared to Hector SLAM is not significant.

Table 11: Computational performance of three algorithms

	Frequency (Hz)	**Processing time (ms)**
Hector SLAM	262	3.8
Gmapping	110	9.1
Cartographer	290	3.4

6.2 Mapping Quality

In the following section, mapping performance was evaluated instead of localization. Firstly, all techniques showed decent mapping quality during the linear path experiment. All the mapping was visualised via Rviz with the same scaling factor of 70%. The map resolution was configured as 0.025m (the length of a grid cell edge). The black colour line shows the outer bounded area in Mocap lab while the red solid line denotes the trajectory of each simulation.

6.2.1 Normal Moving Speed

Hector SLAM

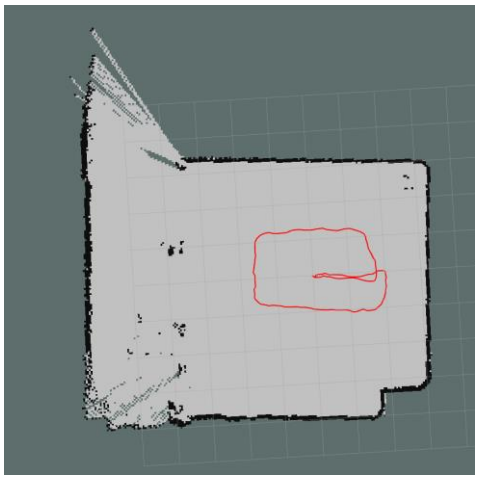

Figure 53: Map visualization of Hector SLAM (linear movement)

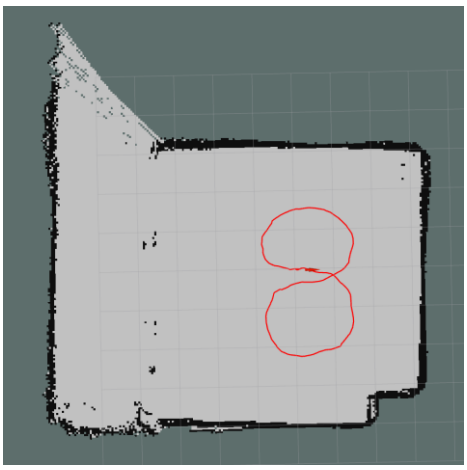

Figure 54: Map visualization of Hector SLAM (circular movement)

Gmapping

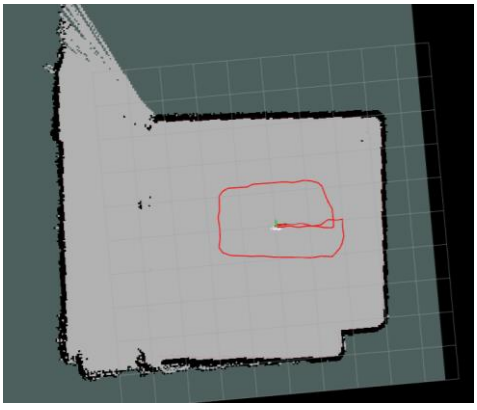

Figure 55: Map visualization of Gmapping (linear movement)

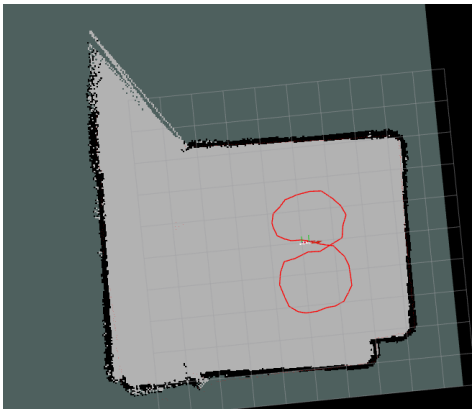

Figure 56: Map visualization of Gmapping (circular movement)

Cartographer

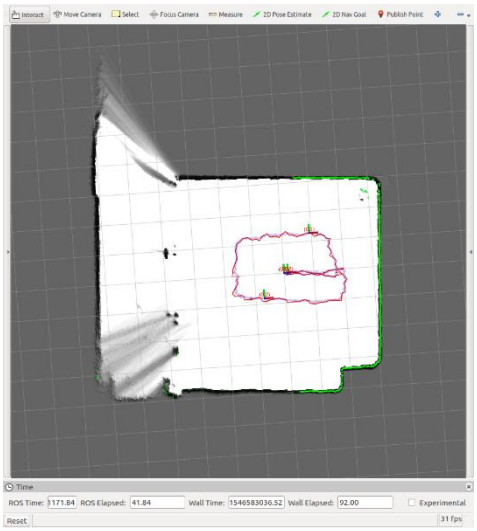

Figure 57: Map visualization of Cartographer (linear movement)

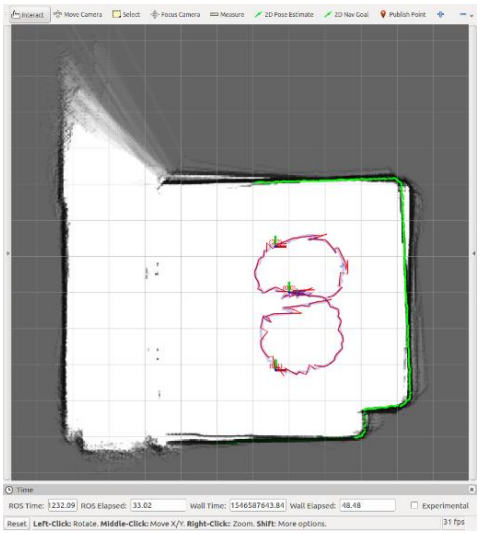

Figure 58: Map visualization of Cartographer (circular movement)

During linear movement, the laser scan deformation is negligible, giving stable mapping performance for all techniques. However, many vacancies were created on the left bottom corner of Figure 57 by the Cartographer algorithm. Surprisingly, Gmapping provided outstanding mapping quality even in the circular path, Figure 56. From Figure 54 and Figure 58, we can see that the laser measurements underwent a significant drifting, causing a severe overlapping issue with Hector SLAM and Cartographer during circular movement.

6.2.2 Fast Moving Speed

Hector SLAM

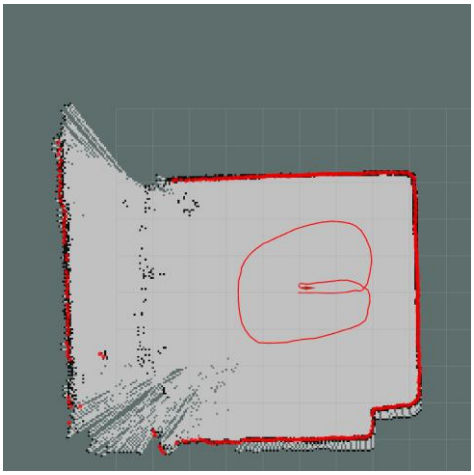

Figure 59: Map visualization of Hector SLAM (faster linear movement)

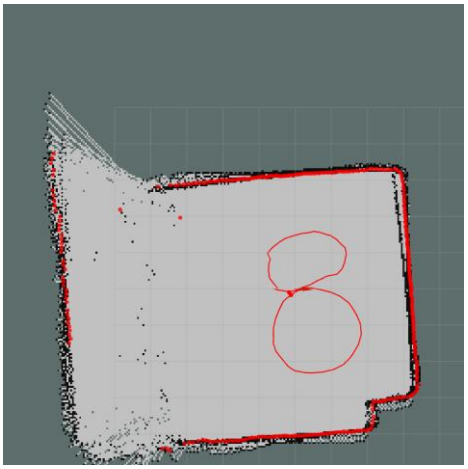

Figure 60: Map visualization of Hector SLAM (faster circular movement)

Gmapping

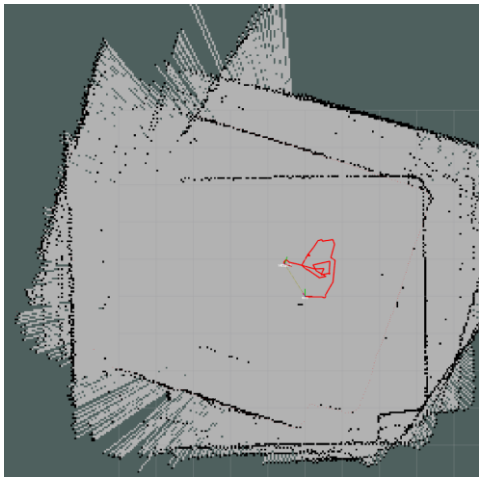

Figure 61: Map visualization of Gmapping (faster linear movement)

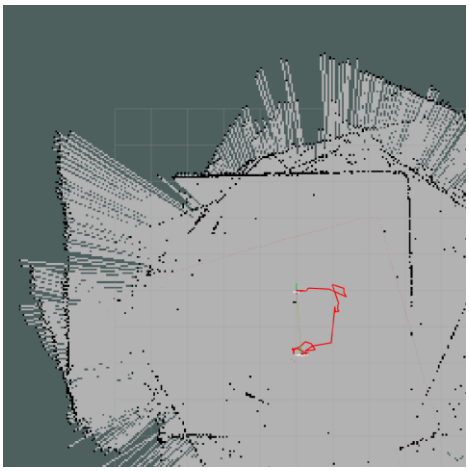

Figure 62: Map visualization of Gmapping (faster circular movement)

Cartographer

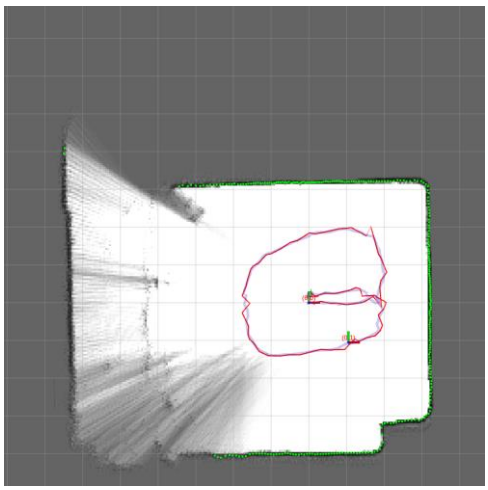

Figure 63: Map visualization of Cartographer (faster linear movement)

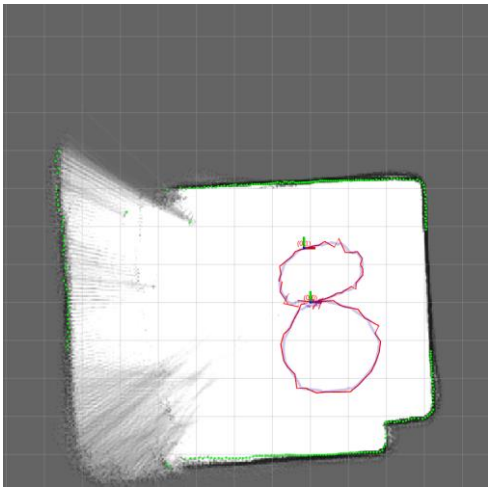

Figure 64: Map visualization of Cartographer (faster circular movement)

When traveling at higher speeds, Gmapping faced a serious issue for map matching algorithms, this results in regenerating a new map which overlapped the previous map (shown in Figure 61 and Figure 62). In the bottom left of Figure 63 and Figure 64, a lot of unoccupied cells were experienced by the Cartographer algorithm. On the bottom left of Figure 59, Hector SLAM had a large unoccupied area, which is now very noticeable. Additionally, there is some minor overlapping issue in Figure 60.

6.3 Overall review on SLAM algorithms

After conducted offline experiments on three different SLAM techniques, further analysis was conducted to provide more insights into the accuracy of pose estimation. Instead of comparing in each axis, displacement in planar dimension was calculated with eq.2, then followed by eq.3.

$$Displacement = \sqrt{(x_{grd} - x_{est})^2 + (y_{grd} - y_{est})^2} \quad (2)$$

where the subscript $_{grd}$ denotes the ground truth value and subscript $_{est}$ denotes the estimated value from SLAM algorithm.

$$RMSD = \sqrt{\frac{\sum_{i=1}^{n}(d_{truth} - d_{est})^2}{n}} \quad (3)$$

where d_{truth} is the true displacement and d_{est} is the estimated displacement. Henceforth, some conclusions on each technique is shown in Table 12.

Table 12: Comparison of different SLAM technique

SLAM Technique	**Accuracy in planar displacement (m)**	**Mapping Quality**	**Comments**
Hector SLAM	Linear Slow: 0.0939 Circular Slow: 0.1483 Linear Fast: 0.1147 Circular Fast: 0.2469	Stable in both cases	-Best performance among all -Decent computational time
Gmapping	Linear Slow: 0.4041 Circular Slow: 0.4260 Linear Fast: 1.3395 Circular Fast: 1.9631	Unreliable during faster speed	-Cannot perform properly without odometry
Cartographer	Linear Slow: 0.1670 Circular Slow: 0.1494 Linear Fast: 0.1595 Circular Fast: 0.2456	Slightly overlap during circular path	-Stable -Fastest computation time -Slightly map overlapping

According to our experiments, some ideas can be retained. Firstly, Hector SLAM algorithm relies only in the laser scan matching without the use of odometry, which could be an advantage for our robot platform. Apart from that, Hector SLAM also provides accurate pose estimation, with an average error of **15.095cm** in translation after taking average RMS of all scenarios. On the other hand, Gmapping showed its robustness only in slowly moving situation, but the time delay accumulated over

time. Most importantly, Gmapping is totally malfunction in a faster movement Lastly, Cartographer achieved the fastest computation time and had a decent average accuracy of 18.04cm in translation, but in some scenario obtained a better result than Hector SLAM. Worth to mention, Cartographer is designed for multiple sensors platform, therefore it is expected to obtain much accurate results in a desired system.

In summary, we have decided to implement the Hector SLAM algorithm onto our onboard UAV platform which only carries one LiDAR sensor and an onboard IMU. This can be justified by its **smooth** trajectory, **reliable** accuracy in different speeds, and **faster** computation time.

7. ON-BOARD EXPERIMENT

From the previous off-board experiment, we decided to use Hector SLAM as our final optimum SLAM algorithm, using a LIDAR sensor with the aid of onboard IMU. To process the real-time experiment, a complete UAV platform was built from scratch, refer to APPENDIX B. The following lists showing the essential components for UAV platform:

Table 13: Basic components of a drone and its function

Names	Function
Pixhawk	Flight Control Unit
RPLidar	LIDAR Sensor for 2D SLAM algorithm
TFmini	Range sensor for altitude detection
4S LiPo battery	Power supply
Intel NUC (installed Ubuntu 16.04)	Onboard processing unit
Motor, Propeller, electronic speed controller	Essential components to provide the momentum of drone
Receiver, Controller	Remote manual control
5V Regulator	Step down the battery voltage

Additional distance sensor is needed to provide information in altitude that can be used for position hovering (e.g. for photography), terrain surface following, collision avoidance, and warning of regulatory height limits, etc. Referring to the Pixhawk documentary, we narrowed down some distance sensors supported by PX4. TFmini Lidar was selected because of its tiny size, low cost, and low power consumption with detecting range from 0.3m – 12m. It can be purchased in Singapore off the shelf and directly connected to Pixhawk via serial communication, shown in Figure 65.

TFmini was configured within QGroundControl (a software to calibrate and configure flight control unit like Pixhawk). Unlike 2D pose information which is determined by the SLAM algorithm with RPLidar, TFmini serves as a range finder that directly provides the altitude with respect to ground. Figure 66 shows the QGroundControl Analyse Tool which can plot distance values in a graph.

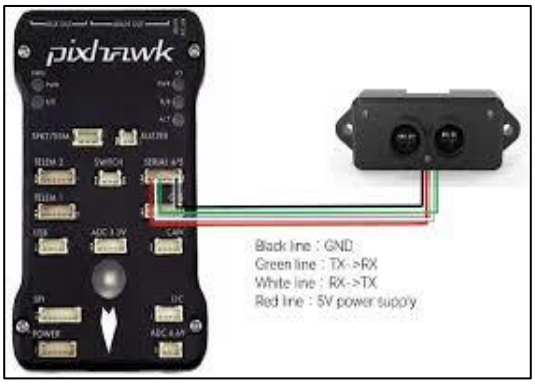

Figure 65: Connection between Pixhawk and TFmini range finder

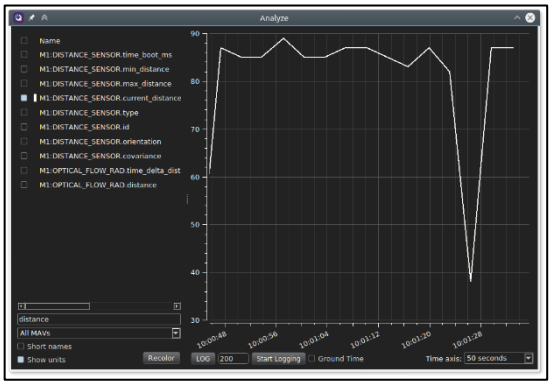

Figure 66: QGroundControl Analyze Tool

The completed UAV system is shown in Figure 67 and Figure 68. Intel NUC serves as the on-board processing unit which receives laser scan data and performs Hector SLAM algorithm meanwhile fusing the altitude (from TFmini) to generate real-time 6 DoF position information.

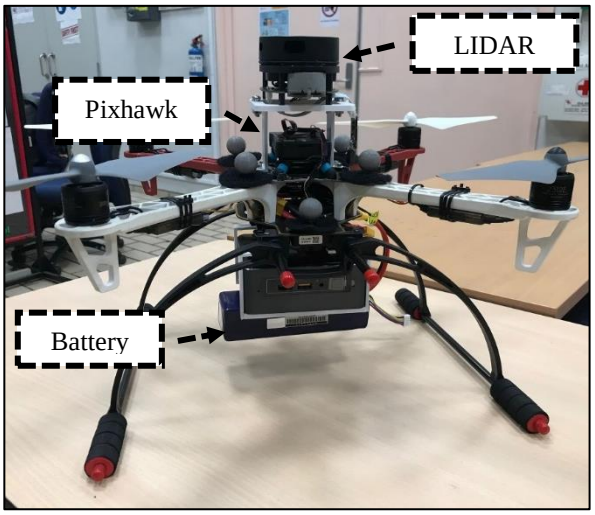

Figure 67: Figure of UAV platform

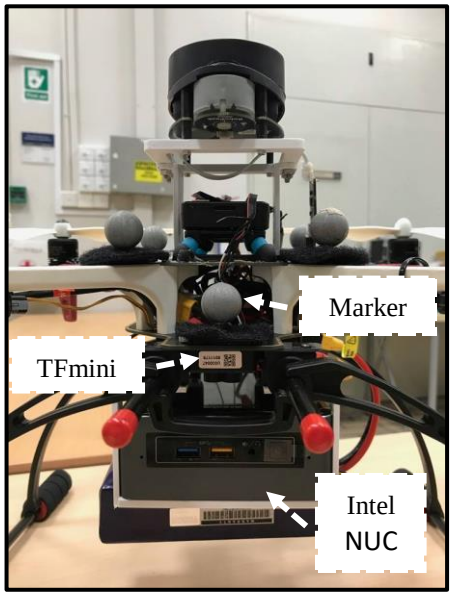

Figure 68: Front view of UAV

Due to the safety reason, the drone was controlled to reach a predetermined set point (0 0 1) without flying around. All the necessary packages were first transferred onto on-board Intel NUC, associated with an external laptop served as a remote control station via Remmina VNC (Virtual Network Computing). Table 14 shows the steps to launch the packages and their functions respectively. Notably, all nodes were running real-time on Intel NUC while all data was recorded for further analysis.

Table 14: Overall launch files and its function

Steps	Launch Files	Functions
1	tutorial_imu.launch	Run RPLidar, Pixhawk, and Hector SLAM stack (contains necessary static_stage_publihser)
2	mocap_bridge.launch	Activate connection between Mocap system and Intel NUC
3	lidar_fusion.launch	Receive altitude, roll, pitch data from Pixhawk, then fuse with 2D pose information from Hector SLAM to become 3D pose
4	tf_broadcaster.launch	Convert *geometry_msgs/PoseStamped* message to tf form
5	state_RCout.launch	Publish the current flight command received by Pixhawk
6	px4_pos_control.launch	Publish a constant setpoint to navigate the drone

After all packages were actively running, the UAV was manually disarmed. It was gradually flying towards the setpoint with the pose feedback from the ground truth. Figure 69 shows the overview of tf frame tree within the current system. Firstly, both LIDAR sensor and TFmini were attached to base_link. Then the vision_link was the 6 DoF transformation of our platform with respect to the map frame.

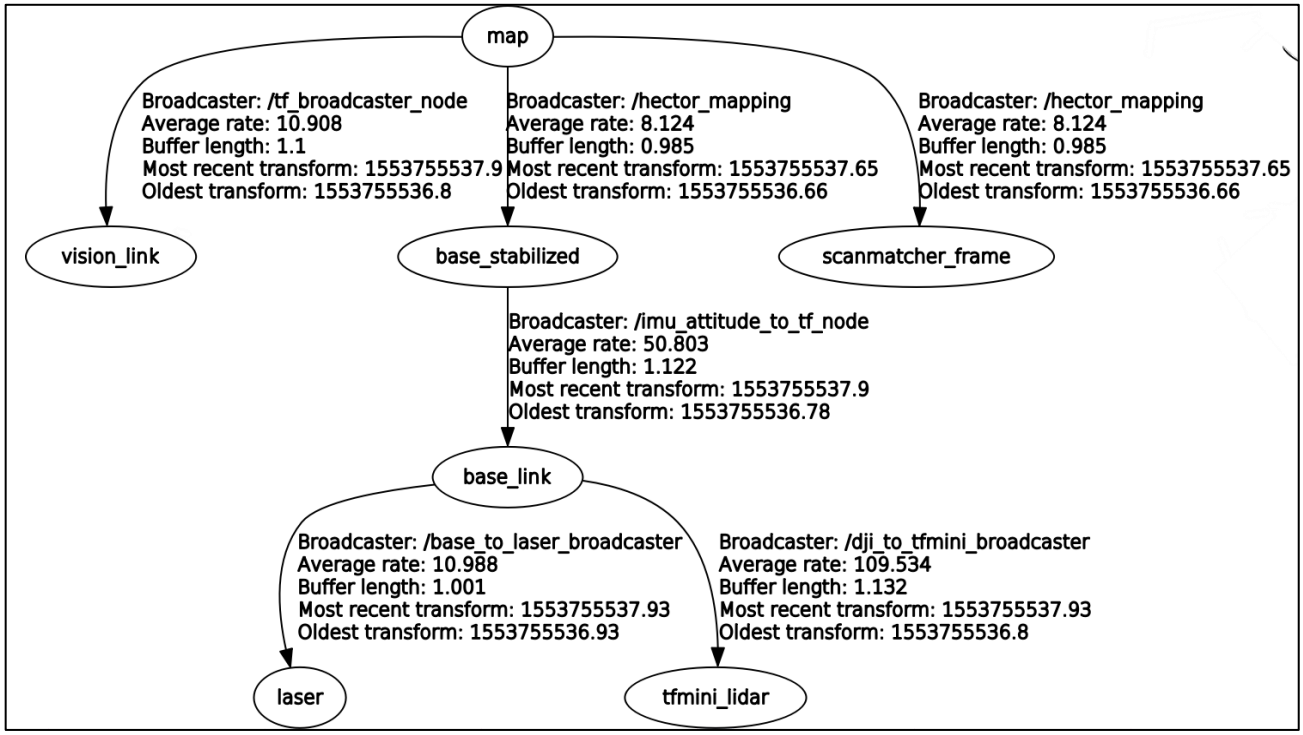

Figure 69: Overview of tf frame tree

In Figure 70, we can see the relationship of all active nodes. Mainly, the mavros node managed all the topics publish and subscribe within Pixhawk. Particularly, /mavros/vision_pose/pose was the 6 DoF pose information after the fusion of sensos.

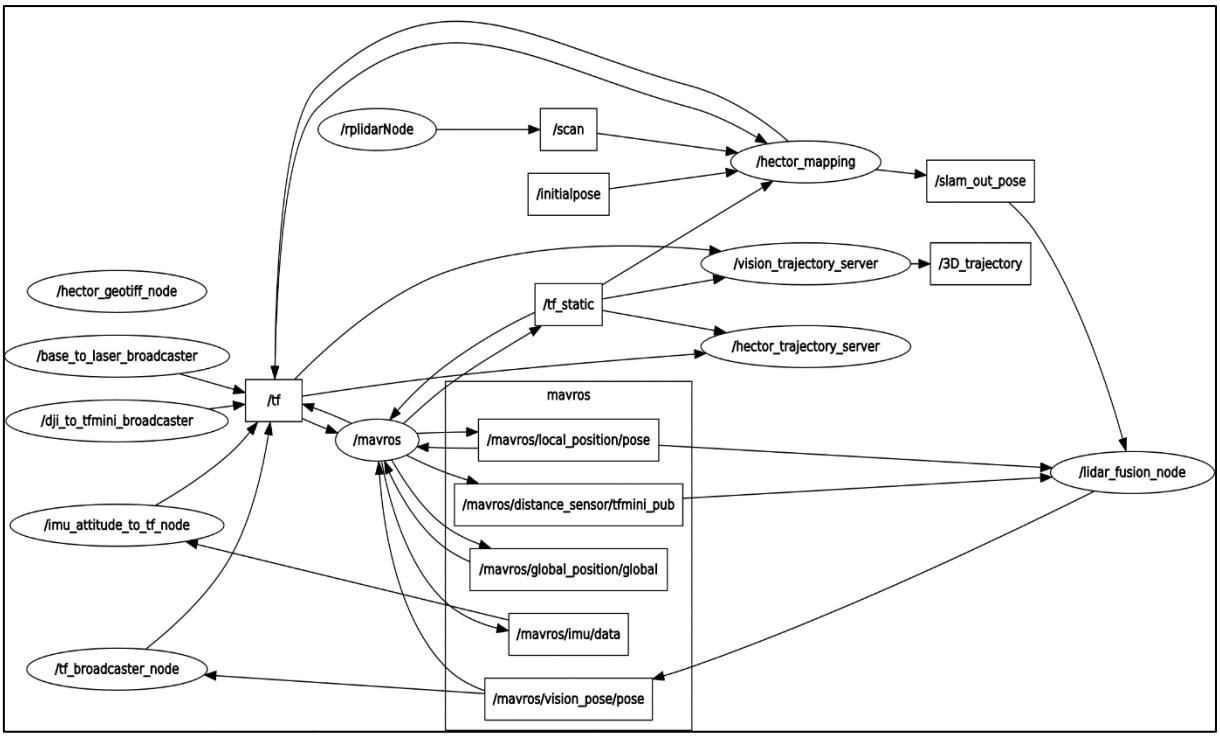

Figure 70: Active nodes and topics

The recorded data was extracted using MATLAB. Instead of calculating the RMS separately for each axis (in Section 6.1), we first used eq.4 to calculate the displacement difference in 3D space, followed by RMS formula mentioned in eq.3.

$$d = \sqrt{(x_{grd} - x_{est})^2 + (y_{grd} - y_{est})^2 + (z_{grd} - z_{est})^2} \quad (4)$$

where x, y, z_{grd} denote the ground truth value and x, y, z_{est} denote the estimated pose.

In this onboard experiment, the localization performance in 3D space obtained an accuracy of **13cm** in translation. Figure 71 shows the visualization of trajectory in 3D plot. Notably, the detecting range of TFmini is only from 0.3 to 12m, therefore any distance below 30cm will be considered as a minimum value of 0.3m. Since the placement of TFmini was 0.08m offset below the CG of the drone, the effective detecting range of altitude is 0.38m to 12.08m.

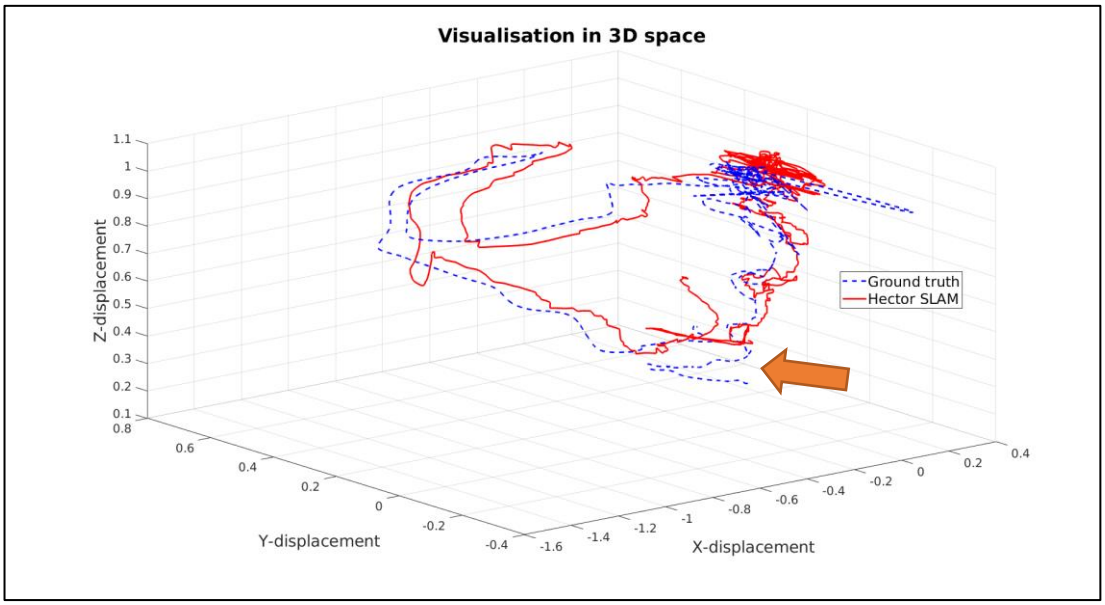

Figure 71: XYZ 3D pose estimation with the ground truth in blue dotted line and Hector SLAM in red line

The orange arrow in Figure 71 shows the gap between the ground truth value and estimated pose due to the drawback of TFmini range sensor.

In nut shell, we successfully obtained 3D pose information with the fusion of Hector SLAM and TFmini range sensor. The comparison with the ground truth value also showed us a satisfactory accuracy within 15cm.

8. CONCLUSION AND FUTURE WORK

In this project, LIDAR-based SLAM algorithm was proposed to solve the localization problem for a UAV platform in a tunnel-like environment that without any GPS signal. Overall, three potential algorithms were implemented and evaluated according to their tracking and mapping performances. From the offline experimental results, the Hector SLAM algorithm obtains an accuracy of 15cm in 2D translation within an indoor environment. Apart from that, Hector SLAM shows a great robustness throughout all scenarios, especially during circular movement in faster pace. This clearly shows the capability of Hector SLAM to perform localization and mapping excellently for a UAV platform in a tunnel-like environment.

As we move forward into the future, autonomous mobile robots will be trend in various manufacturing and service sectors. Self-localization and mapping capability become the critical parts in a robotic system, ensuring a highly reliable and robust navigation stack within an unknown environment. Therefore, in the future research, based on the position information from SLAM algorithm, autonomous path planning and navigation in an unknown working area can be developed with PID feedback control.

For further improvement of current stage, the accuracy and robustness of Hector SLAM algorithm can be further improved by fusing the result from other algorithm such as visual-based SLAM algorithm that using stereo camera to detect the features within an environment. As a drawback, this could lead to an increase of the weight of UAV platform. In fact, the current UAV system can barely take off due to the payload. Therefore, to reduce the weight, the casing of Intel NUC can be replaced by lightweight material such as 3D printed, or an alternative onboard processor unit, Nvidia Jetson TX2, is suggested.

On the other hand, a real experiment needs be conducted in tunnel-like environment to obtain the actual localization result. Some difficulties and obstacles can be expected, for example, the rough surface of the tunnel might not be optimum for laser sensor. As an alternative method, a Gazebo simulation can be designed to test

the UAV platform, which not only avoids the possibility to damage the system, but also reduces the expensive cost of building a real system.

Lastly, intensive-based SLAM algorithm mentioned in [28] can be studied and implemented to overcome the localization problem in a featureless long corridor scenario. The laser scan data which provides the range distances can also be utilized for obstacles avoidance.

In conclusion, by implementing the LIDAR-based Hector SLAM algorithm with the aid of onboard IMU device, this can obtain a highly reliable localization performance within a tunnel-like environment. This would potentially achieve the goal of autonomous UAV inspection in deep sewer without human intervention.

9. REFERENCE LIST

[1] F. Nex and F. Remondino, "UAV for 3D mapping applications: A review," *Appl. Geomatics*, vol. 6, no. 1, pp. 1–15, 2014.

[2] Land Transport Authority, "LTA to Tap on Technology to Enhance Tunnel Inspections | Press Room | Land Transport Authority," 2017. [Online]. Available: https://www.lta.gov.sg/apps/news/page.aspx?c=2&id=8c205baa-6ee5-4cfd-a152-66ba6943bc11. [Accessed: 20-Sep-2018].

[3] C. Siagian, C. K. Chang, and L. Itti, "Autonomous Mobile Robot Localization and Navigation Using a Hierarchical Map Representation Primarily Guided by Vision," *J. F. Robot.*, vol. 31, no. 3, pp. 408–440, May 2014.

[4] I. Skog, "Sensing and Perception: Localization and positioning," *WASP Sweden Graduate School - Autonomous System*, 2016. [Online]. Available: http://wasp-sweden.org/custom/uploads/2016/10/michael-felsberg-overview-20161005.pdf. [Accessed: 12-Mar-2019].

[5] J. M. Santos, D. Portugal, and R. P. Rocha, "An evaluation of 2D SLAM techniques available in Robot Operating System," *2013 IEEE Int. Symp. Safety, Secur. Rescue Robot. SSRR 2013*, 2013.

[6] R. Li, J. Liu, L. Zhang, and Y. Hang, "LIDAR/MEMS IMU integrated navigation (SLAM) method for a small UAV in indoor environments," in *2014 DGON Inertial Sensors and Systems (ISS)*, 2014, pp. 1–15.

[7] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," in *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, 2011, pp. 155–160.

[8] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual SLAM algorithms: a survey from 2010 to 2016," *IPSJ Trans. Comput. Vis. Appl.*, vol. 9, no. 1, p. 16, Dec. 2017.

[9] D. Tardioli and J. L. Villarroel, "Odometry-less localization in tunnel-like environments," *2014 IEEE Int. Conf. Auton. Robot Syst. Compet. ICARSC 2014*, vol. 32100, pp. 65–72, 2014.

[10] H. Wang, W. Mou, G. Seet, M.-H. Li, M. W. S. Lau, and D.-W. Wang, "Real-time Visual Odometry Estimation Based on Principal Direction Detection on Ceiling Vision," *Int. J. Autom. Comput.*, vol. 10, no. 5, pp. 397–404, Oct. 2013.

[11] I. Nikolov and C. Madsen, "LiDAR-based 2D Localization and Mapping System using Elliptical Distance Correction Models for UAV Wind Turbine Blade Inspection," *Proc. 12th Int. Jt. Conf. Comput. Vision, Imaging Comput. Graph. Theory Appl.*, no. Visigrapp, pp. 418–425, 2017.

[12] H. Peel, S. Luo, A. G. Cohn, and R. Fuentes, “Localisation of a mobile robot for bridge bearing inspection,” *Autom. Constr.*, vol. 94, no. April 2017, pp. 244–256, 2018.

[13] B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner, “On Measuring the Accuracy of SLAM Algorithms.”

[14] G. Dissanayake, S. Sukkarieh, E. Nebot, and H. Durrant-Whyte, “The aiding of a low-cost strapdown inertial measurement unit using vehicle model constraints for land vehicle applications,” *IEEE Trans. Robot. Autom.*, vol. 17, no. 5, pp. 731–747, 2001.

[15] Pixhawk, “Introducing Pixhawk® 4 - the flight controller for developers - Pixhawk.” [Online]. Available: http://pixhawk.org/pixhawk4/. [Accessed: 13-Mar-2019].

[16] Wim Meeussen, “REP 105 -- Coordinate Frames for Mobile Platforms (ROS.org),” 2010. [Online]. Available: http://www.ros.org/reps/rep-0105.html. [Accessed: 20-Jan-2019].

[17] W. Hess, D. Kohler, H. Rapp, and D. Andor, “Real-time loop closure in 2D LIDAR SLAM,” *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2016–June, pp. 1271–1278, 2016.

[18] SLAMTEC, “RPLIDAR-A1 360°Laser Range Scanner _ Domestic Laser Range Scanner|SLAMTEC.” [Online]. Available: http://www.slamtec.com/en/lidar/a1. [Accessed: 21-Mar-2019].

[19] SLAMTEC, “RPLIDAR-A2 Laser Range Scanner_ Solid Laser Range Scanner|SLAMTEC.” [Online]. Available: http://www.slamtec.com/en/Lidar/A2. [Accessed: 21-Mar-2019].

[20] IDMiner, “LDS-01 光達 | 採智科技股份有限公司.” [Online]. Available: http://idminer.com.tw/product/lds-01-光達/. [Accessed: 21-Mar-2019].

[21] WONG PEI TING, “TODAYonline | Deepest tunnels in S’pore to start carrying electricity from end-2018.”

[22] ROS.org, “rplidar - ROS Wiki.” [Online]. Available: http://wiki.ros.org/rplidar. [Accessed: 21-Mar-2019].

[23] R. Yagfarov, M. Ivanou, and I. Afanasyev, “Map Comparison of Lidar-based 2D SLAM Algorithms Using Precise Ground Truth,” *2018 15th Int. Conf. Control. Autom. Robot. Vision, ICARCV 2018*, pp. 1979–1983, 2018.

[24] ROS.org, “hector_slam - ROS Wiki.” [Online]. Available: http://wiki.ros.org/hector_slam. [Accessed: 18-Mar-2019].

[25] ROS.org, “gmapping - ROS Wiki.” [Online]. Available: http://wiki.ros.org/gmapping. [Accessed: 19-Mar-2019].

[26] ROS.org, “cartographer - ROS Wiki.” [Online]. Available: http://wiki.ros.org/cartographer. [Accessed: 20-Mar-2019].

[27] The Cartographer, “Cartographer — Cartographer 1.0.0 documentation.” [Online]. Available: https://google-cartographer.readthedocs.io/en/stable/index.html. [Accessed: 19-Mar-2019].

[28] S. Khan, D. Wollherr, and M. Buss, “Modeling Laser Intensities For Simultaneous Localization and Mapping,” *IEEE Robot. Autom. Lett.*, vol. 1, no. 2, pp. 692–699, 2016.

APPENDIX A

Source Codes are on GitHub repository: https://github.com/chowjf/fyp

Simulation Videos:
https://drive.google.com/drive/folders/1Y_NUmGVM9gOJYcFDILGDgqbl_MIg18ov?usp=sharing

MATLAB scripts:
https://drive.google.com/drive/folders/1Y_NUmGVM9gOJYcFDILGDgqbl_MIg18ov?usp=sharing

Rosbag files:
https://drive.google.com/drive/folders/1Y_NUmGVM9gOJYcFDILGDgqbl_MIg18ov?usp=sharing

APPENDIX B

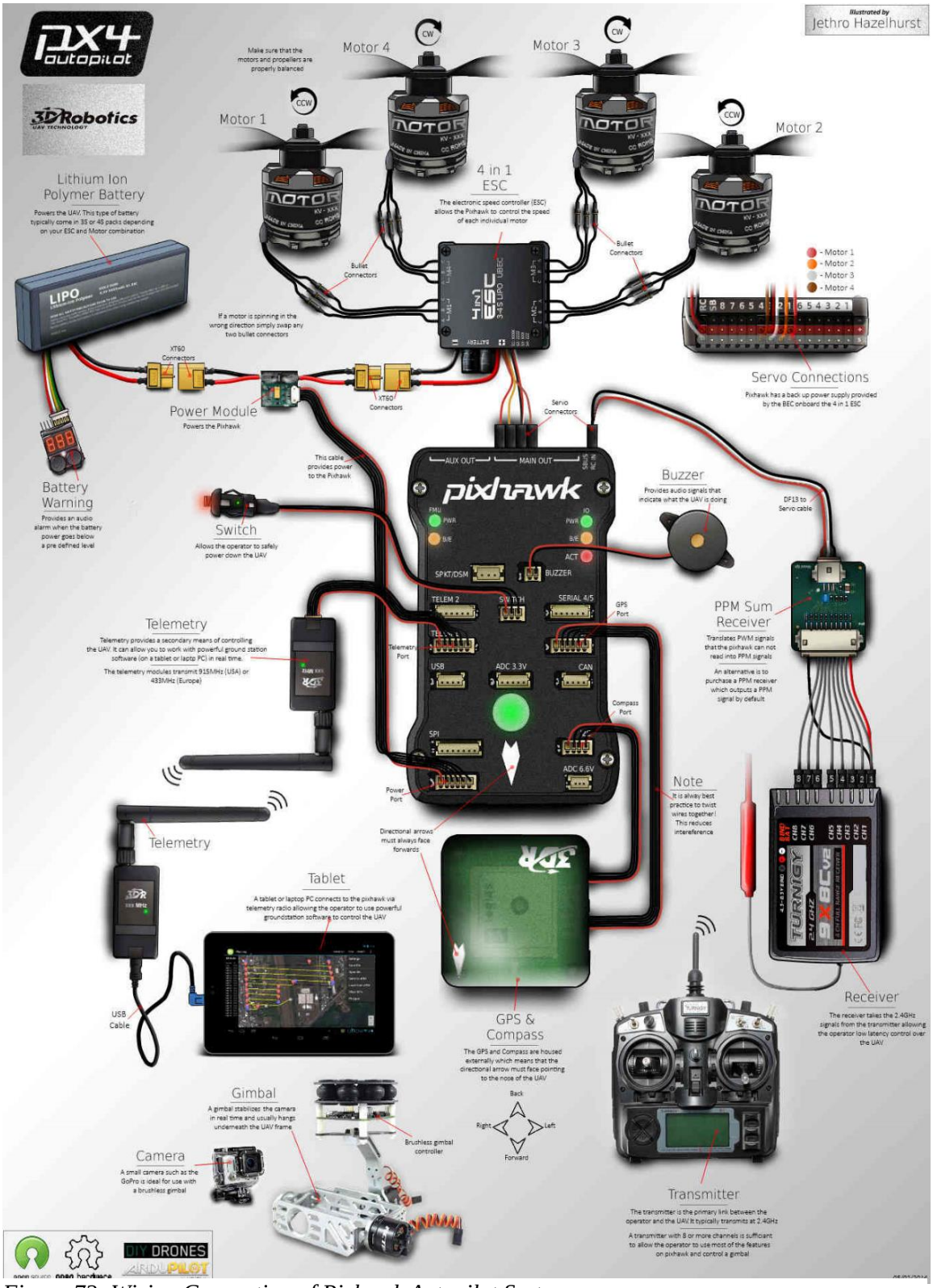

Figure 72: Wiring Connection of Pixhawk Autopilot System