Enhancing Body-Mounted LiDAR SLAM using an IMU-based Pedestrian Dead Reckoning (PDR) Model

by

Hamza Sadruddin

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Master of Applied Science

in

Electrical and Computer Engineering

Carleton University
Ottawa, Ontario

Abstract

With the significant reduction in motion sensors' cost and power, Simultaneous Localization and Mapping (SLAM) has emerged as a core technology that powers up a wide range of applications such as virtual/augmented reality, search-and-rescue, first-responders, mining, and defence. Existing SLAM methods have been designed mainly for robotic platforms that use wheel odometry as a system motion model. However, wheel odometry is not available for body-mounted platforms, which limits the accuracy of SLAM algorithms when implemented on such platforms.

This thesis addresses the challenge of body-mounted SLAM by proposing an integrated sensor fusion scheme. A Pedestrian Dead Reckoning (PDR) model based on the Inertial Measurement Unit (IMU) is used to enhance LiDAR-based SLAM. This proposed fusion uses the PDR model as a replacement for wheel odometry in vehicular platforms. A system prototype has been developed and used for data collection and experiments. Plus, a PDR model was implemented and integrated into the Google Cartographer SLAM engine and tested against different positioning systems such as stand-alone IMU-PDR, Hector SLAM and Cartographer. Experiments demonstrated that the integration of PDR has significantly enhanced head-mounted SLAM accuracy leading to accurate positioning under different motion scenarios.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Mohamed Atia, and acknowledge his contributions during my MASc study and research. His continuous academic guidance and moral support throughout my master's journey have been the most important contributor to my progress. I thank him for his patience, motivation, technical knowledge, and being available whenever I needed any help with the projects, and for taking out time for reviewing the papers and my thesis writing. I especially thank him for always having trust in my abilities.

Besides my advisor, I would like to thank my friends at Embedded Multi-Sensor Systems Lab (EMS-Lab) in making my academic journey enjoyable. My special thanks go to Ahmed Mahmoud for his guidance and help, both as a mentor and as a friend. I am also thankful to Alan Zhang for taking out time to review my thesis, and all the thoughtful discussions we had from technical subjects to the cultural made my stay more pleasant. My sincere appreciation goes to Soroush Sheikhpour for his valuable guidance during my master's program and being there to provide helpful advice both as a mentor and as a friend. It was a great honor to have Prof. Hussein Mouftah, Assoc. Prof. Wei Shi and Asst. Prof. Leila Mostaço-Guidolin as my thesis committee members. Their valuable comments and positive criticism added a great value to my thesis.

Most importantly, I deeply express my praise and gratitude to Allah Almighty for blessing me with the strength to complete this research and his bountiful blessings for everything. I would like to express my profound gratitude to my beloved family for their prayers and unwavering support. I thank them for always believing in me. Last but not the least, I would like to thank my uncle and aunt in Toronto for their constant support throughout my stay in Canada. Without them, this accomplishment would not have been possible.

Dedication

This is dedicated to my beloved family.

Table of Contents

List of Tables

List of Figures

List of Acronyms

Acronym	Definition
AHRS	Attitude Heading-Reference System
EKF	Extended Kalman Filter
GNSS	Global Navigation Satellite Systems
GUI	Graphical User Interface
IF	Information Filter
IMMU	Inertial-Magnetic Measurement Unit
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
KF	Kalman Filter
LiDAR	Light Detection and Ranging
MEMS	Micro Electromechanical Systems
MT	Motion-Tracker
NDT	Normal Distribution Transform
OSM	Open Street Map
PDR	Pedestrian Dead Reckoning
POSE	Position and Orientation
Qt	Pronounced “Cute”, a widget toolkit for creating graphical user interfaces and cross-platform applications
RCS	Remote Control Software
RMSE	Root Mean Square Error
ROS	Robot Operating System
RSDL	Realtime Synchronization and Data Logging

SDI	Strap-Down Integration
SLAM	Simultaneous Localization and Mapping
VPU	Vision Processing Unit
V-SLAM	Visual Simultaneous Localization and Mapping

Chapter 1: Introduction

1.1 Background

Pedestrian tracking can be approached using either infrastructure-based or infrastructure-free methods. Infrastructure-based methods require pre-installation of a sensor network such as WiFi, Ultra-wideband (UWB) or Bluetooth networks. This requirement imposes significant constraints on the environment, which may not be feasible in scenarios where no prior-installation or pre-existing infrastructure can be assumed, such as search-and-rescue and defence applications. On the other side, infrastructure-free methods do not require pre-installed setups. Instead, these methods use self-mounted sensors such as inertial, visual, and range sensors. Therefore, infrastructure-free pedestrian tracking is more suitable for those scenarios where the environment is unknown. An example of such methods is Simultaneous Localization and Mapping (SLAM). SLAM is a framework that estimates position/orientation (i.e. pose) of a platform, and a map of the environment. Therefore, it is an ideal tracking method where there is no prior knowledge of the environment or no pre-installed infrastructure in the environment.

In the past couple of decades, various algorithms have been proposed to implement SLAM [1] by combining different sensor such as Light-Detection-and-Ranging (LiDAR) sensors, cameras, inertial sensors, wheel encoders, and ultrasound sensors. SLAM can also benefit from any external tracking sources such as Global Navigation Satellite Systems (GNSS) [2]. SLAM algorithms mainly utilize exteroceptive sensors such as LiDAR, radar and camera to identify unique features in the environment using measurement models. Measurement model is the mathematical model that maps exteroceptive sensors

observations such as laser range to system states such as pose. Tracking these unique features through consecutive scans (i.e. scan-matching) provides odometry information, which is an estimate of position/orientation change of the platform. Moreover, SLAM platforms may also use proprioceptive sensors such as wheel encoders and inertial sensors. Such proprioceptive sensors provide additional odometry information about platform motion using a system model. System model is the mathematical model that accepts proprioceptive sensors data as input and predicts platform pose as output.

Most existing SLAM algorithms have been originally designed for robotic vehicle platforms that use wheel encoders. Wheel encoders provide an accurate system model that can boost SLAM scan-matching, bridge feature-less environments, and enhance the overall tracking accuracy to avoid wrong loop closures. However, in recent years, several applications that require light-weight accurate SLAM platforms that can be body-mounted have emerged. Body-mounted SLAM is crucial for the safe operation of mission-critical applications such as search-and-rescue, mining industry, and defence applications. However, implementing SLAM on body-mounted platforms has received less attention in the navigation research community. With the development of low-cost light-weight sensors, low-power/compact embedded computing platforms, and advanced communication networks, robust and accurate body-mounted SLAM solutions became possible.

Visual and LiDAR SLAM work well in features-rich environments where dense unique features enable accurate scan-matching. Under constrained motion, such as in land-based vehicles, wheel encoders are used to enhance SLAM accuracy by enhancing the system model. Inertial Measurement Units (IMUs) and inertial navigation systems (INS)

are also commonly used in SLAM algorithms as a compact low-cost source of motion prediction. Using IMU data to support SLAM algorithms works well for ground vehicles where ground motion constraints (i.e. nonholonomic constraints) are applied to the INS system model to reduce IMU-induced random drifts. Therefore, IMU can be used as a replacement for or an augmenting sensor to wheel encoders for wheeled platforms. Other types of odometry sensors, such as air-flow speedometers [3], can be used with aerial vehicles.

LiDAR SLAM is often superior to visual SLAM in indoor environments where the ambient light condition is not optimum. Some popular SLAM engines are known to be dependent on an additional odometry source for robust performance, and these systems are usually designed to assign more weight or trust more on improved system models. OpenSlam's GMapping [4] is an example of one of these engines which depend on odometry information to decrease the uncertainty of the platform's pose.

Body-mounted SLAM implementation is challenging because of the absence of wheel encoders and wheel-based odometry. Furthermore, INS system models cannot be directly used as a replacement for wheel odometry in body-mounted platforms. The highly varying pattern of human motion leads to rapid and large drifts due to IMU random noise accumulation. This challenge requires special processing of IMU data to provide an accurate integrated SLAM framework. Researchers have recently developed reliable human odometry algorithms using IMU known as Pedestrian Dead-reckoning (PDR) [5]. PDR algorithms utilize human motion pattern recognition (e.g. for walking) along with IMU-based Attitude and Heading Reference Systems (AHRS) [6] to provide reliable human motion tracking. In this thesis, we investigate the feasibility of integrating the IMU-

based PDR model within the SLAM framework to support human body-mounted SLAM platforms. Although SLAM provides an estimation of both poses and maps, the primary focus in this thesis is about pose estimation. Plus, the focus in this work was on investigating the calculated 2D pose estimate from the SLAM engine and not on the generated map result or the utilization of the generated map to help in positioning. Thus, to accomplish our goal, the two 2D SLAM engines (Hector SLAM and Cartographer SLAM) and a 2D LiDAR sensor were used to achieve this.

1.2 Thesis Objectives

The ultimate objective of this thesis is to develop a light-weight low-cost helmet-mounted infrastructure-free positioning platform. To achieve this objective, a SLAM platform consisting of a low-cost/low-power/lightweight 2D LiDAR sensor and a compact-size low-powered IMU sensor has been used. To improve SLAM localization accuracy on this head-mounted platform, a PDR model was implemented. Moreover, this work utilized popular open-source LiDAR SLAM engines commonly used for wheeled vehicular platforms, such as Cartographer and Hector SLAM engines. We modified the Cartographer SLAM engine by integrating the PDR model within the odometry tracking modules of the SLAM framework to enable reliable and accurate body-mounted SLAM. The developed PDR can be seen as a replacement for the wheel odometry and INS system model in wheeled vehicular platforms. Two approaches, namely Hector SLAM [7] and Google's Cartographer SLAM [8], were implemented and configured to be tested. The localization results from all approaches are evaluated and compared with a ground truth data trajectory.

1.3 Contributions

The main contributions of this thesis can be summarized as follows:

1) Implementation of an IMU-based PDR Model and integrating it into the Google Cartographer SLAM engine.
2) The development of a physical body-mounted SLAM prototype system consisting of a Helmet-Mounted low-cost/lightweight 2D LiDAR sensor and a low-cost/compact IMU sensor. To build this prototype, a multi-sensor Real-time Synchronized Data Logger (RSDL) firmware has been developed for the sensor integration and the data collection. Also, a Remote-Controlled System (RCS) software with a sophisticated GUI for online monitoring and control of the RSDL was developed.
3) The investigation of the validity and accuracy of the proposed method on real indoor collected datasets under different scenarios and comparisons against aground truth solution.
4) A conference paper about the thesis work has been presented in the IEEE MWSCAS2020 conference. The paper information is given as follows:

   H. Sadruddin, A. Mahmoud and M. M. Atia, "Enhancing Body-mounted LiDAR SLAM using IMU-based Pedestrian Dead Reckoning (PDR) Model," Accepted in *the 63rd IEEE International Midwest Symposium on Circuits and Systems*, Springfield, MA, USA, 2020.

1.4 Thesis Organization

The thesis is organized as follows:

- **Chapter 2:** explains IMU and PDR. It starts with the background of the IMU principles and errors. Then, the IMU system model and AHRS are explained. Subsequently, the PDR concept is elaborated, and our implemented PDR model technique is presented.
- **Chapter 3:** explains the mathematical foundations of SLAM and their application to 2D LiDAR SLAM. Next, two popular SLAM techniques, namely, Hector SLAM and Cartographer SLAM, are elaborated. Afterward, the integration of the PDR model with Google's Cartographer is explained.
- **Chapter 4:** is dedicated to the hardware and software description of the prototype, and explains the design and implementation of RSDL firmware and RCS software. The chapter concludes with a discussion about the testbed and the usage of the Robot Operating System (ROS) software in this work.
- **Chapter 5:** presents the experimental work and results that demonstrate the accuracy of the developed prototype and the proposed system under different conditions and different datasets. The results from different approaches are compared with a ground truth solution providing error metrics.
- **Chapter 6:** provides a summary of the work and main contributions. It concludes with a discussion about future work and potential improvement directions.

Chapter 2: Inertial Measurement Unit and Pedestrian Dead Reckoning

This chapter discusses IMU and the Pedestrian Dead Reckoning (PDR) concepts. The chapter begins with an introduction of IMUs, their working principle, main components, and measurement errors. Subsequently, the chapter explains PDR concepts and common existing PDR methods. The chapter ends with an explanation of the PDR method implemented in this thesis.

2.1 IMU

An IMU is a sensor device that consists of accelerometers that measure specific force, gyroscopes that measure angular velocity, and a processing unit that performs various functions such as noise filtering and temperature calibration. IMU can be used to estimate an object's relative position, velocity, and orientation. IMU is a self-reliant device that belongs to a class of proprioceptive sensors which do not need information from an external source. An Inertial Navigation System (INS) is a navigation method that provides positioning information by processing raw IMU data. INS is a common component of aircrafts, ships, automobiles, and recently, cellphones [9]. A typical IMU comprises three mutually orthogonal gyroscopes and three orthogonal accelerometers. This type of IMU is called a six degree-of-freedom (DoF) IMU. A typical 6DoF IMU can be used to estimate 3D pose. Modern IMUs are commonly 9DoF that come with 3D magnetometer which measures earth's magnetic field. Some IMUs can by 10DoF by including a barometer to measures the atmospheric pressure, which can be used to estimate the altitude (height).

Accelerometers measure the body's acceleration in a body-fixed or sensor frame. However, to provide meaningful navigation data, acceleration measurements are transformed into a local reference frame, commonly in the North-East-Down (NED) frame. To perform this transformation, gyroscope (and possibly magnetometer) measurements are used to calculate the object's orientation, which is used to transform the accelerometer readings to a local reference frame (e.g. NED).

2.2 INS Working Principle

The operating principle of an INS (inertial navigation system) is based upon Newton's first law of motion, which states that an object at rest will remain at rest, and an object in uniform motion will continue to exhibit that motion unless an external force acts upon it. The external force is directly proportional to the body's inertia (linear acceleration and rotation rate), which can be sensed by an IMU.

The gyroscope data is integrated to estimate the orientation. The acceleration is then converted to the local navigation frame by making use of the estimated orientation of the moving platform. The gravity vector is subtracted from the transformed accelerometer data to extract the pure acceleration of the object. The resulting acceleration is then integrated to obtain the velocity. Then it is integrated again to retrieve the position, provided that both the initial velocity and initial position are priori known. This concept is called Dead Reckoning. A common practice to overcome the need to know the initial velocity is to start the integration process at rest. (i.e., zero velocity). The working principle of INS is shown in Figure 2.1.

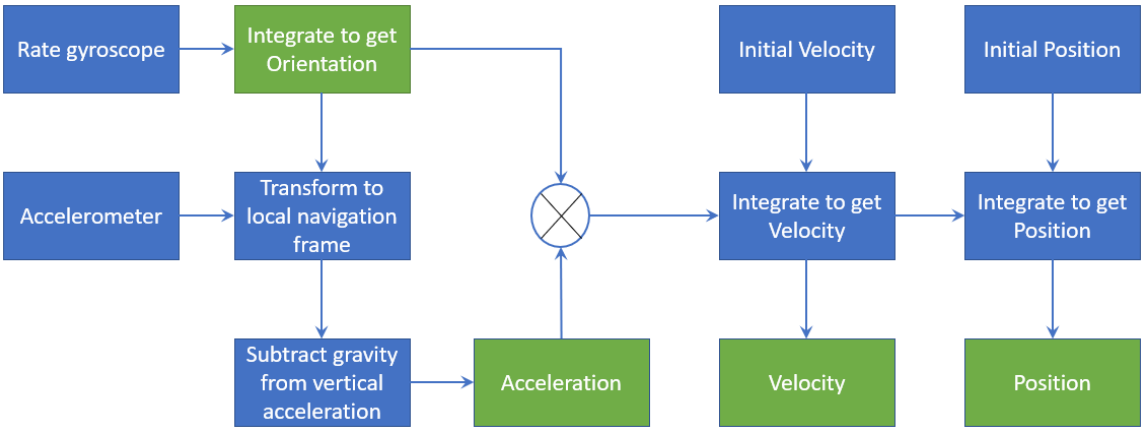

Figure 2.1: IMU Working Principle - Block Diagram

2.3 IMU Components

A typical 6DoF IMU diagram is shown in Figure 2.2. IMU and INS work based on the dead reckoning concept where the change in orientation, position or velocity is estimated and integrated. The result of change derived from the integration of orientation, position or velocity is added to the previous state to acquire the current pose. The output of an IMU device can be potential difference, current or pulses, and these are often converted into specific force and angular rate. These are then integrated over sampling interval τ_i as follows:

$$v_{ib}^{b}(t) = \int_{t-\tau_i}^{t} f_{ib}^{b}(t') \tag{2.1}$$

$$\alpha_{ib}^{b}(t) = \int_{t-\tau_i}^{t} \omega_{ib}^{b}(t')dt' \tag{2.2}$$

Where f_{ib}^{b} is the specific force, ω_{ib}^{b} is the angular rate, v_{ib}^{b} are the velocity increments and α_{ib}^{b} are the attitude increments. Superscripts *"b"* refers to body frame to express that

the measurements vectors are represented in body-frame. Subscript *"ib"* refers to the fact that acceleration and angular rate are of the body *"b"* with respect to an inertial frame "*i*".

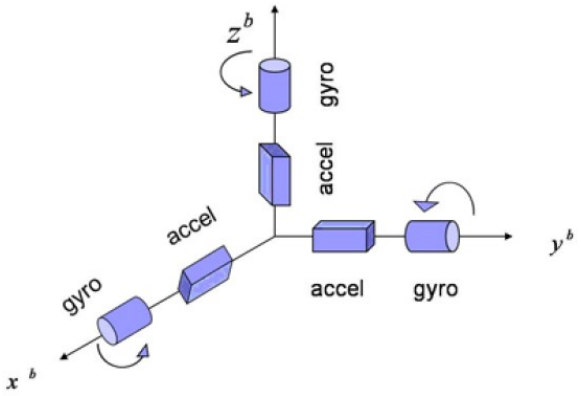

Figure 2.2: Typical 6DoF IMU Components

2.3.1 Accelerometers

An accelerometer is normally made up of three components: a proof mass, a suspension or spring and a pickoff, which together are used in calculating the acceleration force of a body, as the proof of mass moves along the appropriate direction depending on the force being exerted on the accelerometer. An illustrative diagram of an accelerometer is shown in Figure 2.3. The specific force f, which is the output of the accelerometer, is given as follows:

$$f = a - g \tag{2.3}$$

Here, a is the acceleration with respect to the inertial frame and g is the gravitational acceleration. Accelerometers are generally unable to separate the total acceleration of the moving platform or vehicle and the acceleration with respect to inertial space (specific force f) from the force caused by the presence of a gravitational field (g). Hence, gravity acceleration g, must be explicitly removed.

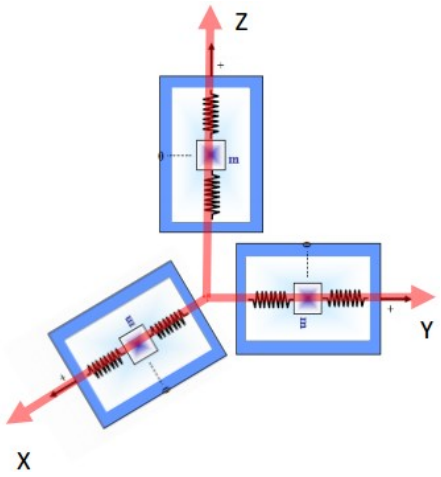

Figure 2.3: Accelerometer Illustrative Diagram

2.3.2 Gyroscopes

Gyroscopes measure the change in angle or rotational motion (angular rate) with respect to an initially known orientation within an inertial frame of reference. Vibrating proof masses are used in almost all micromachine manufactured gyroscopes to monitor rotational motion by the principle of energy transfer between two vibration modes of a structure caused by the Coriolis force [2]. The Coriolis force is proportional to the body velocity. The angle between the direction of motion of the body and the axis of rotation at a given rate of rotation of the observer is given as:

$$A_c = -2\omega \times v \tag{2.4}$$

Where v is the velocity in the rotating frame, and ω is the angular velocity of the rotating platform. For measuring the change in angles, all vibrating gyroscopes implement the Coriolis effect. Figure 2.4 shows a typical diagram of a gyroscope when the angular velocity is applied.

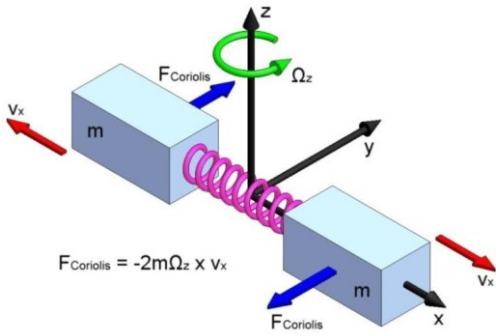

Figure 2.4: Principle of operation of MEMS Gyroscope

Source: Adapted from [10]

2.3.3 Magnetometers

Some modern IMUs also have an additional magnetometer which provides the 3-axis heading direction of an object by referencing the strength and direction of the earth's magnetic field [2]. These IMUs are usually referred to as a Inertial-Magnetic Measurement Unit (IMMU). The sensing element of the magnetometer's sensitivity tends to deteriorate due to the changing environment it operates in and adds error to the measurement data, corrupting the actual readings. The common environmental conditions to which the MEMS are usually exposed to are banking angle and elevation changes. One method usually adopted to remove the errors is to couple the magnetometer with an accelerometer to calculate a heading angle.

2.4 IMU Measurement Errors

IMU sensors are prone to errors, which can accumulate over time, giving less accurate measurements. The main errors that contribute to the output of the accelerometer and gyroscope are biases, scale factor, cross-coupling errors, and random noises. Common systematic errors are shown in Figure 2.5.

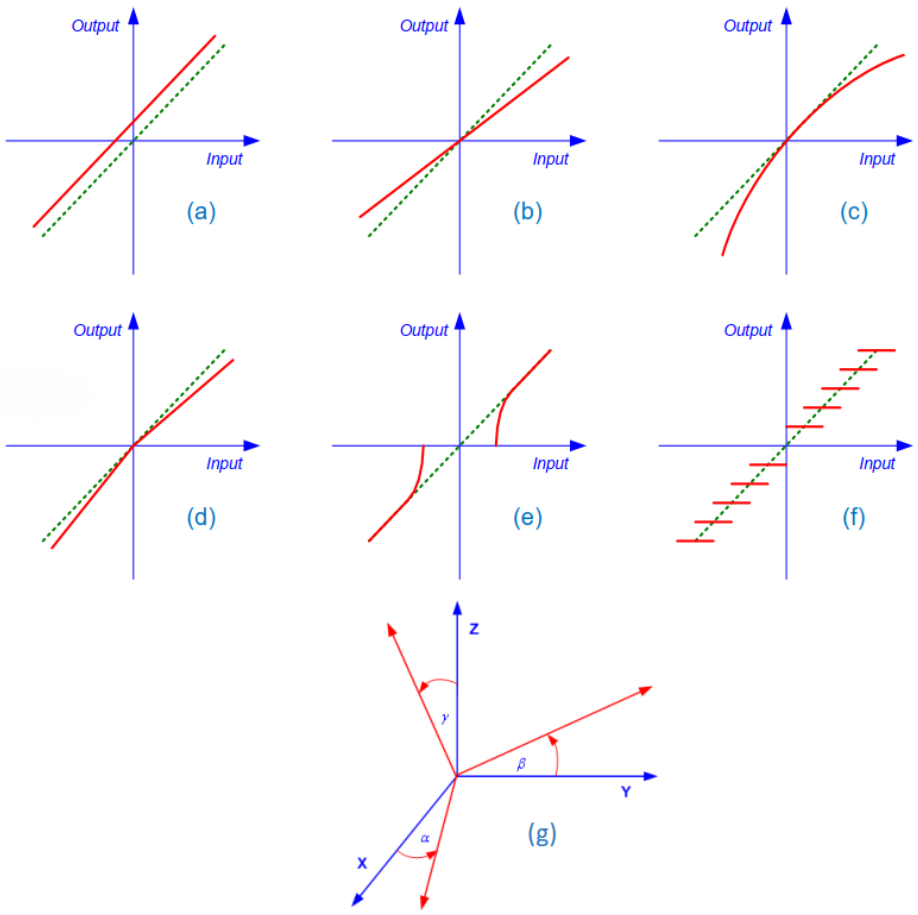

Figure 2.5: IMU Systematic Errors (a) Bias (b) Scale factor (c) Nonlinearity (d) Scale factor sign asymmetry (e) Dead zone (f) Quantization (g) Cross-Coupling

2.5 Classical INS System Model

To describe the inertial motion of an object, four coordinate frames are involved: the Body frame (B), the Local Level Navigation frame (L), the Earth-Centered Earth-Fixed frame (ECEF), and the Earth-Centered Inertial frame (I). The system of frames is shown in Figure 2.6 and described in Table 2.1. In Figure 2.6, the symbol φ represents the geocentric latitude. It is the angle of intersection of the line starting from the center to a point on the surface of the ellipsoid with the equatorial plane. The symbol λ, corresponds

to the longitude. It is the angle between the meridian plane containing the point of interest and the prime meridian [2].

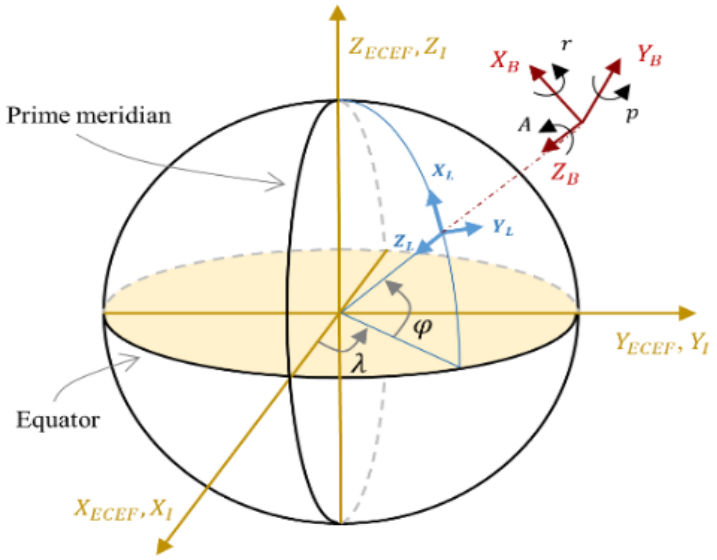

Figure 2.6: The System of Coordinate Frames

Newton's law of motion applies in the *I* frame; therefore, IMU measurements are taken with respect to *I* frame but represented in the *B* frame. The overall navigation solution is computed and represented in a common local frame (*L*), which is the North-East-Down "NED" frame. The Earth-Centered-Earth-Fixed (ECEF) is included to make the system compatible with GNSS if available. An IMU is used to predict the orientation of the body frame *B* with respect to frame *L* and then projects accelerations onto the *L* frame to calculate full 3D positions and velocities by double integration after compensating for gravity.

Table 2.1: Reference Coordinate Frames

Frame	Definition
B: Body Frame	Origin: Body center of mass X: Longitudinal (forward) direction Y: Transversal (lateral) direction Z: Down (vertical) direction
L: Local Level navigation frame	Always levelled with respect to Earth ellipsoid surface Origin: Vehicle location X: True north direction Y: East direction Z: Down vertical direction
E: Earth-Centered Earth-Fixed (ECEF)	Rotates with Earth Origin: Center of the Earth Z: extends through the North Pole. X: passes through the intersection of the equatorial plane and the prime meridian. Y: completes the right-hand coordinate system in the equatorial plane.
Earth-Centered Inertial Frame (I)	Coinciding with ECEF at zero longitude and it does not rotate with Earth

In this work, the quaternion vector $q_B^L = [a\ b\ c\ d]^T$ representation for orientation is used [11], [12]. The orientation kinematic differential equation is represented as follows:

$$\dot{q}_B^L = \frac{1}{2}\begin{bmatrix} a & -b & -c & -d \\ b & a & -d & c \\ c & d & a & -b \\ d & -c & b & a \end{bmatrix}\begin{bmatrix} 0 \\ \omega_{LB_x}^B - b_{g_x} \\ \omega_{LB_y}^B - b_{g_y} \\ \omega_{LB_z}^B - b_{g_z} \end{bmatrix} \tag{2.5}$$

Where ω_{LB}^{B} is the gyroscope measurements which measures the rotation rate of the B frame with respect to the L frame and gyroscope biases b_{g_x}. The velocity differential equation can be approximated by [11], [12]:

$$\dot{v}^{L} \approx R_{B}^{L}(a_{IB}^{B} - b_{a}) + g^{L} \tag{2.6}$$

Where a_{IB}^{B} is the measured acceleration in the B frame, b_a is an accelerometer bias vector, g^L is the gravity vector in the L frame, and R_B^L is the rotation matrix from B to L. The position differential equation in the L frame can be approximated by [11], [12]:

$$\dot{P}^{L} \approx v^{L} \tag{2.7}$$

The direct application of this classical INS system dynamic model described above results in unbounded drifts due to the integration of IMU errors. Those drifts can be reduced under vehicular motion constraints using wheel odometry. However, in the absence of wheel odometry, classical INS system model in pedestrian tracking generates large and rapid drifts. To address this challenge, PDR is used as a more accurate INS model for pedestrian motion prediction. In the following section, we explain the PDR and some of its techniques, and subsequently, we will explain our implemented PDR technique.

2.6 Pedestrian Dead Reckoning (PDR)

One solution to improve the pedestrian navigation is to use an IMU for step counting as a measure of forward motion instead of direct integration of acceleration data like in the conventional INS system model. A step is the movement of one foot while the other foot remains stationary, whereas a stride is the successive movement of both feet [2]. The person's step length can be projected in the direction of motion to estimate the person's

new position. This approach or technique is called Pedestrian Dead Reckoning. To be feasible for pedestrian use, sensors must be small, lightweight, low power, and low-cost. Thus, Micro Electromechanical Systems (MEMS) IMU sensors are used. In the literature, some authors use shoe-mounted IMU PDR to benefit from Zero Velocity Updates (ZUPT) that happen when the person's foot touches the ground during a walking step. However, in our thesis, the IMU is used to support LiDAR SLAM, which cannot be shoe mounted. Therefore, in this thesis, step detection using IMU detected acceleration spikes (instead of ZUPTs) is used. A pedestrian dead-reckoning algorithm generally comprises four phases: 1) heading angle estimation, 2) step detection, 3) step length estimation, and 4) position update. A general block diagram of PDR is shown in Figure 2.7. The following subsections will shed more light on each block of Figure 2.7.

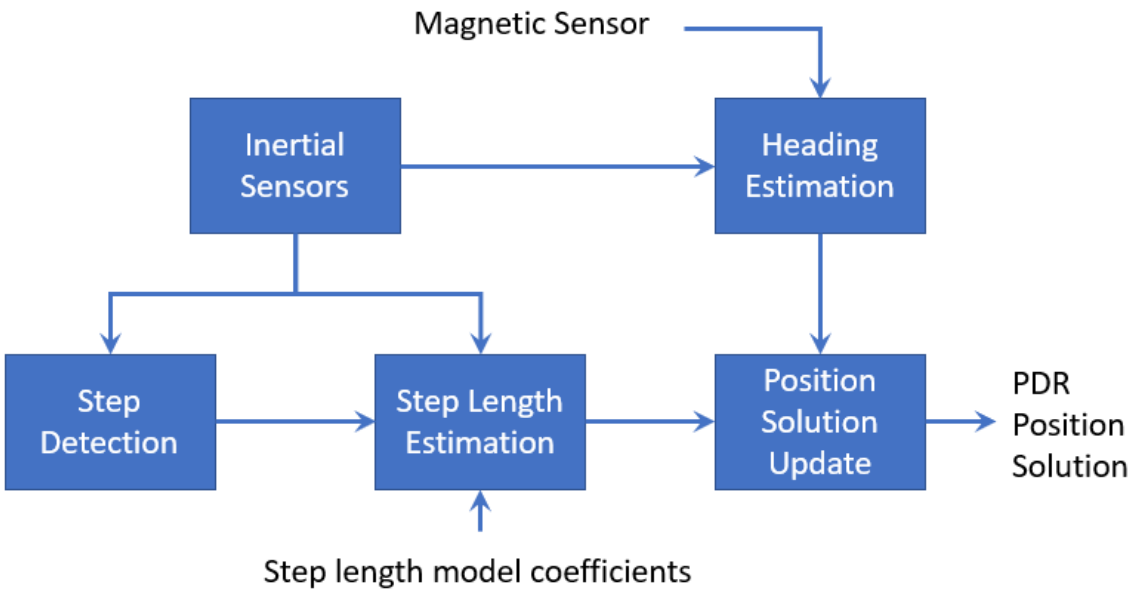

Figure 2.7: PDR- General Block Diagram

2.6.1 Heading Estimation using Attitude Heading-Reference System (AHRS)

This subsection demonstrates the sensor fusion filter that processes the raw IMU measurements to calculate 3D orientation. This is also known as the Attitude Heading-Reference System (AHRS).

2.6.1.1 AHRS EKF and System Model

An AHRS is commonly implemented using the Kalman Filter (KF) [9]. The KF generally performs well for real-time implementations, as the set of equations, which applies Bayesian recursive estimation, are simple. The KF works under the assumptions of linear state transitions and linear measurements with added Gaussian noise, which are rarely applicable in practice. Hence, a variation of the KF for non-linear systems called the Extended Kalman Filter (EKF) is utilized. The EKF can be applied to optimally estimate a set of states modelled by a nonlinear dynamic system. A state vector $x(t)$ that evolves non-linearly over time can be modelled as follows:

$$\dot{x}(t) = f\big(x(t), u(t), w(t)\big) \tag{2.8}$$

$$y(t) = h\big(x(t)\big) + v(t) \tag{2.9}$$

where $f(.)$ is a nonlinear system model, $u(t) \in \mathbb{R}^d$ is a control signal vector of d sensors that trigger the evolvement of n-element state vector $x(t) \in \mathbb{R}^n$ and $w(t) \in \mathbb{R}^n$ is the stochastic noise of covariance Q. A measurement vector $y(t) \in \mathbb{R}^m$ represents m observable states of the systems; these are the states that can be directly measured by an external observer. The measurement model is commonly a nonlinear function *h(.)* that maps the system's state $x(t)$ into the system's observables $y(t)$. As measurement instruments are not perfect, $v(t) \in \mathbb{R}^m$ is used to model measurement noise with

covariance R. Using first-order Taylor series approximation, (2.8) and (2.9) can be linearized as follows:

$$\begin{gathered}\delta\dot{x}(t) = F(t)\delta x(t) + w(t) \\ \delta\dot{y}(t) = H(t)\delta x + v(t) \\ F(t) = \frac{\partial f(x(t), u(t))}{\partial x} \\ H(t) = \frac{\partial h(x(t))}{\partial x}\end{gathered} \tag{2.10}$$

EKF optimal state estimation can be performed through a prediction step, and an update step described as follows:

Prediction

$$\begin{gathered}x_{k+1} = f(x_k, u_k) \\ P_{k+1} = (I + F_{k+1}T)P_k(I + F_{k+1}T)^T + Q_{k+1}\end{gathered} \tag{2.11}$$

Update

$$\begin{gathered}K_{k+1} = P_{k+1}\,H_{k+1}^T(\,H_{k+1}\,P_{k+1}\,H_{k+1}^T + R_{k+1})^{-1} \\ x_{k+1} = x_{k+1} + K_{k+1}[y_{k+1} - h(x_{k+1})] \\ P_{k+1} = (I - K_{k+1}H_{k+1})P_{k+1}\end{gathered} \tag{2.12}$$

In an AHRS, the system model (also known as 3D Orientation Kinematic Model) is simply the orientation prediction part of the INS system model described in (2.5).

2.6.1.2 AHRS Measurement Models

To perform the update step in the AHRS EKF, the 3D orientation information independently calculated by accelerometers and magnetometers sensors is used. At the stationary state or very low-speed applications, the tilt angles (roll and pitch) can be estimated from accelerometers [2], [13], which will be mainly measuring the gravity, and heading can be estimated from the magnetometers [13]. Tilt (roll and pitch) angle

measurements (r_m and p_m) can be calculated from accelerometer measurement vector a_{SF}^B as follows [2], [14]:

$$r_m = atan2\left(-a_{SF_Y}^B, -a_{SF_Z}^B\right) \tag{2.13}$$

$$p_m = atan2\left(a_{SF_X}^B, \sqrt{{a_{SF_Y}^B}^2 + {a_{SF_Z}^B}^2}\right) \tag{2.14}$$

Heading information is calculated using magnetic field sensors. A well-calibrated magnetometer [2], [13], which when pointed to magnetic north, will ideally measure reference magnetic field m^L. Similar to gravity, m^L can also be determined from standard tables as the one publicly accessible on Natural Resources Canada, the magnetic field calculator [15]. In a scenario where the magnetometer has a non-zero heading, m^L will be projected to body axis accordingly. To calculate heading from magnetometer measurements, the following formula can be used [2], [13], [16]:

$$\begin{aligned} A_m = atan2(&-m_y \cos(r_m) + m_z \sin(r_m), m_x \cos(p_m) + \\ &m_y \sin(r_m) \sin(p_m) + m_z \cos(r_m) \sin(p_m)) + D(l, L) \end{aligned} \tag{2.15}$$

where, $[{m^B}_X \; {m^B}_Y \; {m^B}_Z]^T$ are 3D-magnetometer readings, and $D(l, L)$ is the declination angle (i.e. deviation between magnetic north and true north). If we have full orientation estimation from accelerometers and magnetometer, we can convert the updates (r_m, p_m, A_m) to quaternion vector $q_v^m$and feed them to the EKF as measurement updates. The overall AHRS EKF module is represented in Figure 2.8.

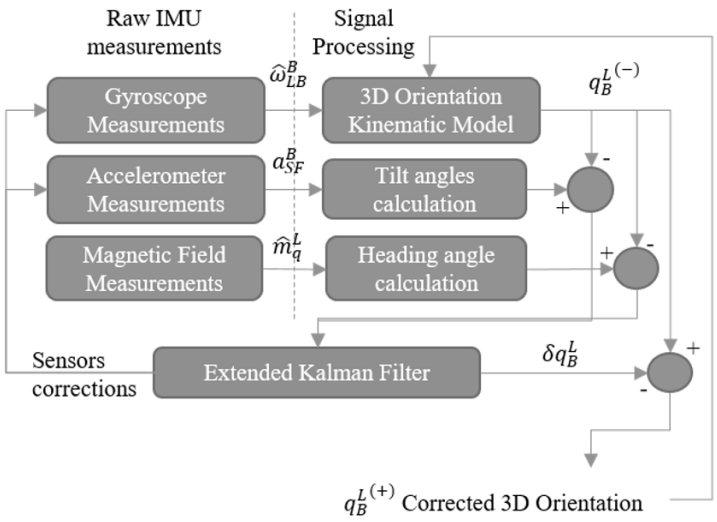

Figure 2.8: Loosely-coupled AHRS EKF

Source: Adapted from [6]

2.6.2 Step-Detection

Pedestrian walking patterns generate peaks in the acceleration, as shown in Figure 2.9. Therefore, step detection can be implemented using a peak detector applied to accelerometer data. To avoid the effects of sensor tilt and body swing, instead of using the acceleration component, the magnitude of the acceleration is preferred. In this thesis, the peak detection algorithm was applied to the total acceleration vector given by:

$$a_r = \sqrt{a_x^2 + a_y^2 + a_z^2} \tag{2.16}$$

Where, a_r is the resultant acceleration and a_x, a_y and a_z represent the x, y and z-axis accelerometer readings respectively. In the figure, a red triangle on a peak corresponds to a step detection.

Figure 2.9: Peak-Step Detection using Resultant Acceleration Data

2.6.3 Step Length Estimation

Different methods exist for estimating the stride length of a pedestrian. For example, one of the methods uses an empirical non-linear model to calculate the stride length [17]**.** Some use an algorithm based on a backpropagation artificial neural network to estimate the stride length [18]. Investigation of step length estimation techniques is out of the scope of this thesis. Here, the following empirical approach was adopted:

1. A walking portion of each dataset (discussed in Chapter 5:) was selected.
2. The number of steps during the selected walking portion was counted.
3. Travelled distance during the selected portion was measured using a RealSense solution [19]. RealSense is the ground truth solution that has been

used in this thesis to compare the developed algorithms and provide error metrics.

4. Stride length was then estimated via dividing the travel distance measured from step 3 by the total number of steps calculated in step 2.
5. Finally, the average stride length from all the datasets was used.

2.6.4 Position Calculation

Once a peak is detected, a step forward is applied to update the pedestrian's position. The pedestrian's step length and the heading calculated from the AHRS are used to calculate the 2D pedestrian's position (x, y) using the following model:

$$x_k = x_{k-1} + (l \times \sin(\theta_k)) \tag{2.17}$$

$$y_k = y_{k-1} + (l \times \cos(\theta_k)) \tag{2.18}$$

Where:

- x and $y -$ 2D pose estimate
- $\theta -$ heading angle
- $l -$ Step length

Chapter 3: Simultaneous Localization and Mapping (SLAM)

This chapter describes the concepts and algorithmic details of SLAM. Subsequently, Hector and Google's Cartographer SLAM engines are elaborated. The chapter ends with a section describing how the developed PDR model was integrated within the Cartographer SLAM engine.

3.1 Mathematical Representation of SLAM

3.1.1 SLAM Problem Parameters Definition

To understand SLAM concepts, it is important to define the main variables that represent the SLAM problem. Suppose there is a robot moving through an environment taking relevant observations using a range sensor, as shown in Figure 3.1. At the time instant k, following quantities can be defined:

- $\mathbf{x}_k$: the state vector which describes the robot's location and orientation
- $\mathbf{u}_k$: the control vector that moves the robot to a state $\mathbf{x}_k$ at time k
- $\mathbf{m}_i$: a vector describing the location of the ith landmark
- $\mathbf{z}_{ik}$: observation of the location of the ith landmark at time k

Plus, the following sets are also defined:

- $\mathbf{X}_{0:k} = \{\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_k\} = \{\mathbf{X}_{0:k-1}, \mathbf{x}_k\}$: the history of robot locations
- $\mathbf{U}_{0:k} = \{\mathbf{u}_0, \mathbf{u}_1, \cdots, \mathbf{u}_k\} = \{\mathbf{U}_{0:k-1}, \mathbf{u}_k\}$: the history of control inputs
- $\mathbf{m} = \{\mathbf{m}_0, \mathbf{m}_1, \cdots, \mathbf{m}_n\}$ the set of all landmarks
- $\mathbf{Z}_{0:k} = \{\mathbf{z}_0, \mathbf{z}_1, \cdots, \mathbf{z}_k\} = \{\mathbf{Z}_{0:k-1}, \mathbf{z}_k\}$: the set of all landmark observations.

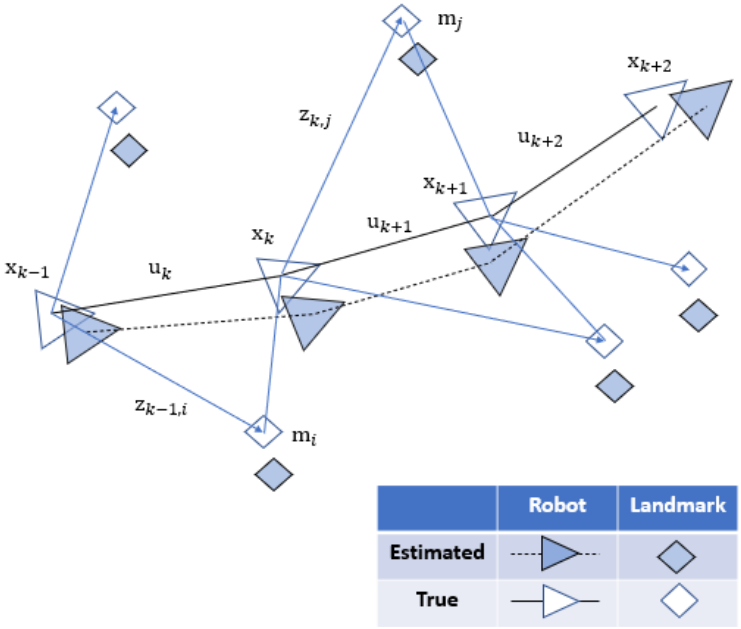

Figure 3.1: Classical SLAM problem

3.1.2 Probabilistic SLAM Formulation

Since all sensors include random noise, the SLAM problem is commonly formulated as a probability distribution. The fundamental probabilistic form of SLAM is given as follows:

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) \tag{3.1}$$

This probability distribution defines the joint posterior density of the landmark locations and robot state (at time k) given the logged observations and control inputs up to and including time k, with the initial state of the robot. A recursive solution of the SLAM problem is used using Bayes rule. By using Bayes theorem, starting with an estimate for the distribution $P(\mathbf{x}_{k-1}, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1})$ at time $k-1$, the joint posterior, following a control $\mathbf{u}_k$ and observation $\mathbf{z}_k$, is computed. To compute this probability distribution, an observation (measurement) model and a state transition (system) model are defined to

describe the effects of the observation and control input, respectively. The system model of the robot can be described in the form:

$$P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) \tag{3.2}$$

The state transition model is based on a Markov process, which means that the next state $\mathbf{x}_k$ is depending only on the immediate previous state $\mathbf{x}_{k-1}$ and the applied control input $\mathbf{u}_k$, and does not depend on the observations nor the map. The observation model describes the probability of seeing $\mathbf{z}_k$ when the robot location and landmark locations are known. A general equation of it is as follows:

$$P(\mathbf{z}_k \mid \mathbf{x}_k, \mathbf{m}) \tag{3.3}$$

Once the robot location and map are defined, observations are assumed to be conditionally independent provided the map and the current robot state [20]. The implementation of the SLAM algorithm in a standard two-step recursive (sequential) prediction (time-update) correction (measurement-update) form is implemented as follows:

Time-Update

$$P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0) = \int P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) \times P(\mathbf{x}_{k-1}, \mathbf{m} \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1}, \mathbf{x}_0) \mathrm{d}\mathbf{x}_{k-1} \tag{3.4}$$

Measurement Update

$$P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) = \frac{P(\mathbf{z}_k \mid \mathbf{x}_k, \mathbf{m}) P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0)}{P(\mathbf{z}_k \mid \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k})} \tag{3.5}$$

Equation (**3**.**4**) and (3.5) provide a recursive method for computing the joint posterior $P(\mathbf{x}_k, \mathbf{m} \mid \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0)$ for the robot state $\mathbf{x}_k$ and map $\mathbf{m}$ at a time k, based on all

observations $\mathbf{Z}_{0:k}$ and all control inputs $\mathbf{U}_{0:k}$ up to and including time k. The recursion is a function of a system model $P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k)$ and an observation model $P(\mathbf{z}_k \mid \mathbf{x}_k, \mathbf{m})$.

All existing SLAM algorithms provide solution implementation to the probabilistic formulation described in (**3.4**) and (3.5) in different ways using different forms of system models and measurement models. The most common SLAM implementations are 1) EKF SLAM [1], 2) Fast SLAM [21], and 3) Graph SLAM [22]. Furthermore, there are two distinguish SLAM approaches; 1) *Online SLAM* and 2) *Full SLAM*. Online SLAM tries to recover only the current robot pose $\mathbf{x}_k$, whereas full SLAM tries recover the entire robot path $\mathbf{X}_{0:k}$ [9]. In this thesis, two popular open-source SLAM engines that are widely known in the literature are used, Hector SLAM [7] which is an online SLAM engine and Google's Cartographer SLAM which is a full SLAM engine. In the subsequent sections, the background of Hector SLAM and Google's Cartographer SLAM are explained.

3.2 LiDAR SLAM

In this section, the SLAM concepts are demonstrated on the 2D LiDAR sensor as an exteroceptive sensor and IMU as a proprioceptive sensor. A simple block diagram of the LiDAR SLAM concept is shown in Figure 3.2. In the following subsection, the main steps of LiDAR SLAM are described.

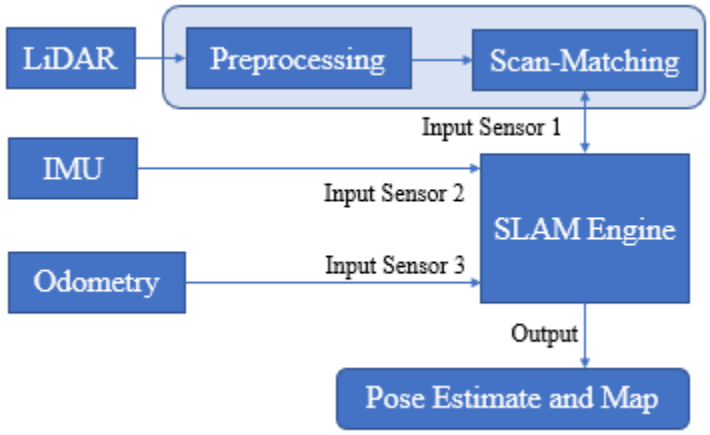

Figure 3.2: SLAM - General Block Diagram

3.2.1 Preprocessing of laser scans

2D LiDAR provides depth (range to objects) information in multiple directions. However, some of the readings are irrelevant for the SLAM due to undesired reflection, sensor noise, etc. This is usually dealt with by preprocessing of the laser scans via downsampling the number of points or outliers removal. Hence, preprocessing of the point clouds must be done to filter the noisy or irrelevant data. For example, in Hector SLAM, the endpoints of the beams created from the laser scans which fall within the threshold of the scan plane are used for the scan-matching. The Cartographer SLAM engine uses the voxel filter [23] for handling the computational weight of points handling. The voxel filter downsamples the raw scan points into cubes of a constant size and then keeps the centroid of each cube for computation.

3.2.2 Scan-Matching

A LiDAR scan can be represented as a point cloud where each beam is represented by a cartesian coordinate of a point in 2D space. Scan-matching is the process of finding the rigid body transformation (combination of translation and rotation) that optimally aligns two scans (i.e. point clouds). The scan-matching process is visualized in Figure 3.3.

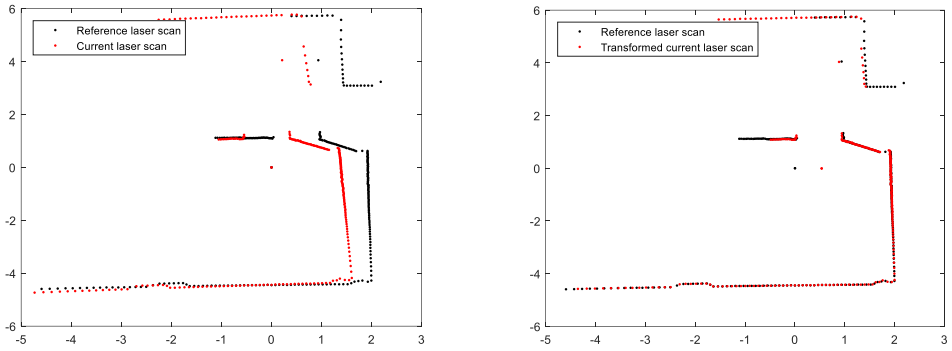

Figure 3.3: Scan-matching concept. (Left) raw scans, (Right) transformed scans.

Source: Adapted from [24]

One of the popular algorithms that perform scan-matching is the Iterative Closest Point (ICP) approach [25]. ICP algorithm iteratively revises the transformation needed to minimize an error metric, usually a distance from the source to the reference point cloud, such as the sum of squared differences between the coordinates of the matched pairs. Another popular scan-matching technique called Normal Distribution Transform (NDT) is also commonly used. NDT transforms the set of the 2D points, which are extracted and reconstructed from the LiDAR scan data into a piecewise continuous and differentiable probability density defined on the 2D plane. The calculated probability density consists of a collection of normal distributions. Matching the second scan to the reference scan is then achieved by maximizing the sum, which the aligned points from the second scan achieved

on this density [26]. The position estimate and map generation are described below under two popular SLAM engines, Hector SLAM and Google Cartographer SLAM.

3.2.3 SLAM Engines

SLAM engine is the actual implementation of the pose and map estimation given the basic sources of information (e.g. system model, measurement model, and scan-matching) as described above. In the following subsections, we describe two popular engines that implement the above described concepts to perform 2D LiDAR SLAM: Hector SLAM and Google Cartographer SLAM.

3.3 Hector SLAM

3.3.1 System Overview

Hector SLAM [7] is a simple SLAM engine that consists of two main blocks, a Navigation Filter, and a SLAM subsystem. The Navigation Filter fuses information from IMU and other available sensors (GPS, compass, etc.) using an EKF to form a 3D solution, and the 2D SLAM subsystem provides the position and heading estimations within the ground plane using LiDAR scan-matching. Both blocks operate individually and are loosely coupled for time synchronization. An overview of Hector SLAM's main structure, in the form of a block diagram, is shown in Figure 3.4. The dashed lines in the figure depict optional information.

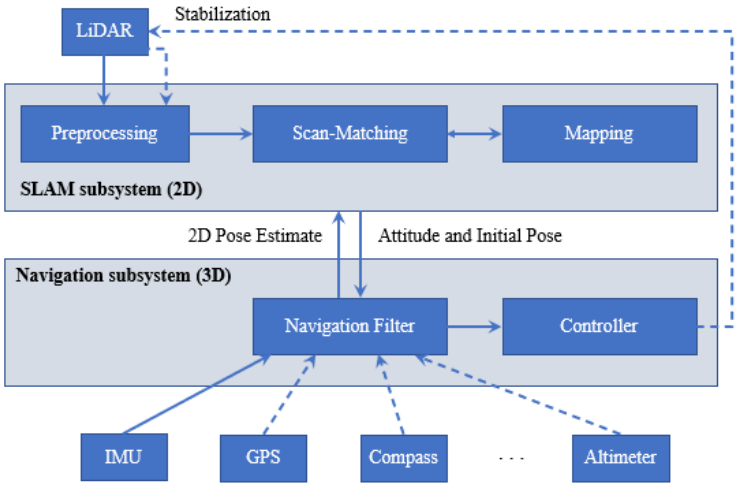

Figure 3.4: Overview of the Mapping and Navigation System

3.3.2 System Model

The full 3D state is represented as follows:

$$\mathbf{x} = (\boldsymbol{\Omega}^T \ \mathbf{p}^T \ \mathbf{v}^T)^T \tag{3.6}$$

Where $\boldsymbol{\Omega} = (\phi, \theta, \psi)^T$ are the roll, pitch, and yaw Euler angles, and $\mathbf{p} = \left(p_x, p_y, p_z\right)^T$ and $\mathbf{v} = \left(v_x, v_y, v_z\right)^T$ represent the position and velocity of the platform stated in the navigation frame. The inertial measurements provide the body fixed angular rates $\boldsymbol{\omega} = \left(\omega_x, \omega_y, \omega_z\right)^T$ and accelerations $\mathbf{a} = \left(a_x, a_y, a_z\right)^T$, and thus form an input vector $\mathbf{u} = (\boldsymbol{\omega}^T \ \mathbf{a}^T)^T$. The motion of an arbitrary rigid body is described by the non-linear differential equation system models described in (2.5-2.7). Due to the existence of IMU sensor noises, the integrated velocity and position show significant drift. To mitigate this, additional sensor information is needed. In Hector SLAM, the scan-matcher is used to retrieve this information, which is suitable for indoor scenarios and applications.

3.3.3 2D SLAM Model

To represent the arbitrary environments, an occupancy grid map [1] is used, which is a well-known approach for LiDAR-based localization in a real-world environment. Using the estimated orientation of the LiDAR sensor (using the AHRS), the scan is transformed into a local stabilized horizontal coordinate frame. This scan is then converted into a point cloud of endpoints using the estimated platform orientation and joint values. For the scan-matching purpose, only the endpoints within a threshold of the intended scan plane are used.

3.3.3.1 Map Access

Due to the discrete nature of the occupancy grid maps, they hamper the good precision and hinder the direct computation of the interpolated values or derivatives. To mitigate this, an interpolation scheme through bilinear filtering is utilized for both calculating occupancy probabilities and derivatives. Grid map cell values can also be viewed as samples containing continuous probability distribution.

3.3.3.2 Scan-matching

As aforementioned, scan-matching is the process of aligning the laser scans among each other or with an existing map. Hector SLAM's approach uses the optimization of the alignment of lidar beam endpoints with the map learnt so far. The optimization is implemented using the Gauss-Newton approach. The benefit of this approach is that it does not require data association between the beam endpoints or an exhaustive and computational pose search. As scans get aligned with the map, the matching is also performed with all the recorded scans. The objective here is to find the rigid transformation

$\xi = (p_x, p_y, \psi)^T$ (p_x and p_y represent the position, and ψ represent heading) that minimizes:

$$\boldsymbol{\xi}^* = \underset{\xi}{\operatorname{argmin}} \sum_{i=1}^{n} \left[1 - M(\mathbf{S}_i(\boldsymbol{\xi}))\right]^2 \tag{3.7}$$

So, the transformation returns the best alignment of the laser scan with the map. Here, $\mathbf{S_i(\xi)}$ are the world coordinates of the scan-endpoints $\mathbf{s_i} = (s_{i,x}, s_{i,y})^T$. They are a function of ξ, the pose of the robot in world coordinates:

$$\mathbf{S}_i(\boldsymbol{\xi}) = \begin{pmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{pmatrix} \begin{pmatrix} s_{i,x} \\ s_{i,y} \end{pmatrix} + \begin{pmatrix} p_x \\ p_y \end{pmatrix} \tag{3.8}$$

The function $M(\boldsymbol{S_i(\xi)})$ returns the map value at the coordinates fed by $\mathbf{S}_i(\boldsymbol{\xi})$. Having some starting estimate of $\boldsymbol{\xi}$, to compute $\boldsymbol{\Delta\xi}$ which optimizes the error measure according to

$$\sum_{i=1}^{n} \left[1 - M(\mathbf{S}_i(\boldsymbol{\xi} + \boldsymbol{\Delta\xi}))\right]^2 \to 0 \tag{3.9}$$

After applying first-order Taylor expansion to $M(\boldsymbol{S}_i(\boldsymbol{\xi} + \boldsymbol{\Delta\xi}))$, and minimizing it by setting the partial derivative with respect to $\boldsymbol{\Delta\xi}$ to zero, and finally, solving for $\boldsymbol{\Delta\xi}$ produces the Gauss-Newton equation for the minimization problem:

$$\Delta\boldsymbol{\xi} = \mathbf{H^{-1}} \sum_{i=1}^{n} \left[\nabla M(\mathbf{S}_i(\boldsymbol{\xi})) \frac{\partial \mathbf{S}_i(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\right]^T \left[1 - M(\mathbf{S}_i(\boldsymbol{\xi}))\right] \tag{3.10}$$

With

$$\mathbf{H} = \left[\nabla M(\mathbf{S}_i(\boldsymbol{\xi})) \frac{\partial \mathbf{S}_i(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\right]^T \left[\nabla M(\mathbf{S}_i(\boldsymbol{\xi})) \frac{\partial \mathbf{S}_i(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}}\right] \tag{3.11}$$

Using the equation (3.8) we get:

$$\frac{\partial \mathbf{S}_i(\boldsymbol{\xi})}{\partial \boldsymbol{\xi}} = \begin{pmatrix} 1 & 0 & -\sin(\psi)_{s_{i,x}} & -\cos(\psi)_{s_{i,x}} \\ 0 & 1 & \cos(\psi)_{s_{i,x}} & -\sin(\psi)_{s_{i,x}} \end{pmatrix} \tag{3.12}$$

Using an approximation for the map gradient $\nabla M\left(\mathbf{S}_i(\boldsymbol{\xi})\right)$ [7] and $\frac{\partial \mathbf{S}_i(\xi)}{\partial \xi}$, the Gauss-Newton equation (3.10) can be evaluated, which yields a step $\boldsymbol{\Delta\xi}$ towards the minimum. To achieve a Gaussian approximation of the match uncertainty, an approximate of Hessian matrix is used to derive the covariance estimate. Here, the covariance matrix is represented as follows:

$$\mathbf{R} = Var\{\boldsymbol{\xi}\} = \sigma^2 \cdot \mathbf{H}^{-1} \tag{3.13}$$

Here, σ is the scaling factor, which is dependent on the properties of the laser scanner device used.

3.3.3.3 Multi-Resolution Map Representation

Since the present approach is based on gradient ascent, it is possible to get stuck in local minima. To mitigate this, a multi-resolution map representation similar to the one from the image pyramid [27] approaches in computer vision is used. In Hector SLAM's approach, they optionally use multiple occupancy grid maps in which each coarser map has half the resolution of preceding one. Here, different maps are kept in memory and are simultaneously updated by the pose estimates, which are generated by the alignment process. This approach ensures that the maps are consistent across scales and avoids the costly downsampling operations. One benefit of this approach is the immediate availability of raw-grained maps, which can be used for path planning.

3.3.4 3D State Estimation

This section will cover the explanation of integrating the estimation of the 3D state vector from the navigation filter and the 2D SLAM solution. The navigation filter runs in real-time at a constant rate of 100Hz and is updated asynchronously with the pose generated from the scan-matching.

3.3.4.1 Navigation Filter

For estimating the full 6D pose of the platform, Hector SLAM uses EKF, where the system model is the general INS system model defined in section 2.5. Additionally, the state vector is augmented with the biases from the gyroscope and accelerometer readings. The velocity and position updates are computed from the integration of the accelerometer readings, and to stabilize the system, additional feedback from measurement updates are used. In Hector SLAM, the scan-matcher updates the 2D position and orientation in the horizontal plane. In contrast, for 3D estimation, additional data from a barometer or other height sensors would be necessary.

3.3.4.2 SLAM Integration

To achieve better performance, information between 2D SLAM and 3D EKF estimates are exchanged in both directions. Pose estimates from the EKF solution are projected onto the horizontal plane and used as a starting estimate for the optimization process to enhance the scan-matching process. Conversely, covariance intersection (CI) is employed to integrate the SLAM pose with the full belief state [28]. If a basic Kalman measurement update is used, it would lead to over-confident estimates, because it would then assume statistically independent measurement errors. Kalman estimates at the time of

the scan are denoted as $\hat{\mathbf{x}}$ with covariance $\mathbf{P}$, and the SLAM pose $(\boldsymbol{\xi}^*, \mathbf{R})$ as defined in (3.7) and (3.13). The fusion estimate is obtained as follows:

$$(\mathbf{P}^+)^{-\mathbf{1}} = (1-\omega).\mathbf{P}^{-1} + \omega.\mathbf{C}^T\mathbf{R}^{-1}\mathbf{C} \tag{3.14}$$

$$\hat{\mathbf{x}}^+ = \mathbf{P}^+\left((1-\omega).\mathbf{P}^{-1}\hat{\mathbf{x}} + \omega.\mathbf{C}^T\mathbf{R}^{-1}\boldsymbol{\xi}^*\right)^{-\mathbf{1}} \tag{3.15}$$

Here, $\mathbf{C}$ is the observer matrix used for projecting the full state space into the 3d subspace of the SLAM system. The purpose of the parameter $\omega \in [0,1]$ is to tune the effect of the SLAM update. Due to the similarities present between Kalman Filter and the Information Filter, the covariance form of the covariance intersection can be written as,

$$\mathbf{P}^+ = \mathbf{P} - (\mathbf{1}-\omega)^{-\mathbf{1}}.\mathbf{KCP} \tag{3.16}$$

$$\hat{\mathbf{x}}^+ = \hat{\mathbf{x}} + \mathbf{K}(\boldsymbol{\xi}^* - \mathbf{C}\hat{\mathbf{x}}) \tag{3.17}$$

where,

$$\mathbf{K} = \mathbf{PC}^T\left(\frac{\mathbf{1}-\omega}{\omega}\cdot\mathbf{R} + \mathbf{C}^T\mathbf{PC}\right)^{-1} \tag{3.18}$$

This is the approach employed by the Hector SLAM's algorithm here because the inversion of the full state covariance according to (3.14) and (3.15) is computationally expensive.

3.4 Google's Cartographer SLAM

3.4.1 System Overview

Google's Cartographer is a Graph SLAM engine. It provides a real-time solution for map building that generates 2D grid maps. Laser scans are inserted into submaps at the best estimated pose. Scan-matching happens against recent-submaps, and the error of pose estimates accumulates in the world frame. To cope with the accumulation of errors, the SLAM engine regularly runs pose optimization using Graph Optimization [22]. All the finished submaps along with the new scans take part in the loop-closure. Based on the information from the pose estimate, if the scans are found to be closed enough, scan-matching tries to find the best scan in the submap. If a good match has been found in the search window near the currently calculated pose, it is included as a loop-closing constraint to the optimization problem. Scan-matching for the loop-closure must happen quicker than the arrival of new scans; otherwise, it will fall behind noticeably. For this, a branch-and-bound approach and several pre-computing grids per finished submap are employed.

3.4.2 Local 2D SLAM

Cartographer combines separate local and global approaches to optimize the pose $\xi = (\xi_x, \xi_y, \xi_\theta)$, consisting of a translation (x, y) and rotation ξ_θ. On an unleveled platform (e.g. 2D motion is not guaranteed like head-mounted platforms), it can use an IMU to estimate the orientation using gravity and project the scans from a 2D LiDAR on to the 2D horizontal plane. In the local approach, each consecutive scan is compared against a small chunk of the map already learnt called submap M. This process of alignment is also

called scan-matching. Scan-matching is prone to error, which accumulates over time. It is taken care of by the global approach, which is discussed in section 3.4.3.

3.4.2.1 Local Scans

Submap construction involves the iterative process of repeatedly aligning scans and submap coordinate frames. With the origin of the scan at $\mathbf{0} \in \mathbb{R}^2$, the information about the scan points can be written as:

$$H = \{h_k\}_{k=1,\dots,K}, h_k \in \mathbb{R}^2 \tag{3.19}$$

The pose ξ of the scan frame can be represented in the submap frame as the transformation frame T_ξ, which rigidly transforms the laser scans from the scan frame to the submap frame, described as:

$$T_\xi p = \underbrace{\begin{pmatrix} \cos\xi_\theta & -\sin\xi_\theta \\ \sin\xi_\theta & \cos\xi_\theta \end{pmatrix}}_{R_\xi} p + \underbrace{\begin{pmatrix} \xi_x \\ \xi_y \end{pmatrix}}_{t_\xi} \tag{3.20}$$

3.4.2.2 Submaps

A few consecutive scans create Submaps. The submaps take the form of a probability grid $M : r\mathbb{Z} \times r\mathbb{Z} \rightarrow [p_{min}, p_{max}]$, which map from discrete grid points at a given resolution r (e.g. 5 cm), to values. The probability grid keeps updating until a submap has been created. These grid points can be thought of containing probabilistic values. Whenever a scan is to be inserted into the grid points, a set of grid points for *hits* and a disjoint set of *misses* are calculated. For every hit, a set of closest grid points is inserted into the hit set, on the other hand, for every miss, a grid point associated with each pixel is inserted that intersects one of the rays between the scan origin and each scan point. This excludes grid points that are already in the hit set. Figure 3.5 represents the scans and pixels

associated with the hits, represented by crossed and shaded, whereas the misses are represented as only the shaded one. Here, every previously unobserved grid point is assigned a probability of p_{hit} or p_{miss}. If the grid point x has already been observed, the rule for updating the grid cell value odds for hit and misses is represented as follows:

$$M_{new}(cell) = \text{clamp}\left(\text{odds}^{-1}\left(\text{odds}\left(M_{old}(cell)\,.\,\text{odds}(p_{hit})\right)\right)\right) \tag{3.21}$$

Where,

- $M_{old}(.)$ – the old probability of the cell
- p_{hit} – the probability of the cell being hit
- $\text{odds}(p) = \frac{p}{1-p}$

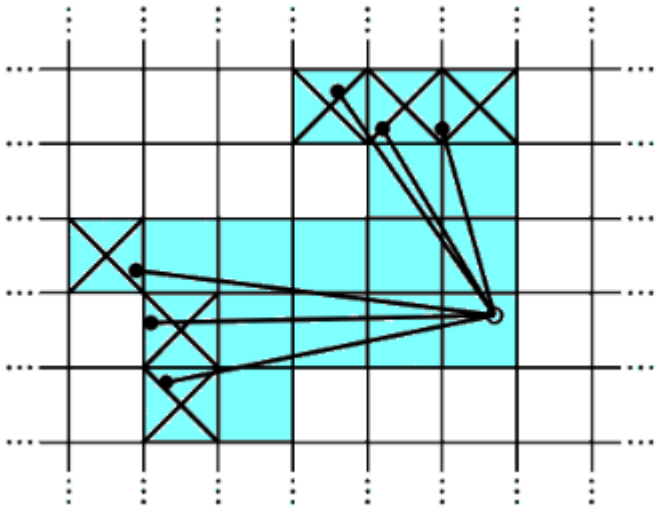

Figure 3.5: A scan and pixels representing the hits and misses

Source: Adapted from [8]

3.4.2.3 Ceres Scan-Matching

Before inserting the scan pose ξ into the submap, it is optimized using a ceres-based [29] scan matcher. Thus, the scan-matching process goes through minimization by casting it as a non-linear least squares problem using the following equation:

$$\underset{\xi}{argmin} \sum_{k=1}^{K} \left(1 - M_{smooth}(T_{\xi} h_k)\right)^2 \tag{3.22}$$

Where,

- $M_{smooth}(x)$ – the value of a cell x smoothed by values in its neighbourhood
- h_k – a cell involving point from laser scan
- T_{ξ} – the transformation matrix shifts a point h_k by ξ

Bicubic interpolation is used here, hence the values outside the interval $[0, 1]$ can occur but are considered innocuous. Since this is a local optimization, hence good initial estimates are usually desired. An IMU, which can give information about the angular velocities, can be used to compute the rotational component θ of the pose between the scans for scan-matching. In the absence of an IMU, a higher frequency of laser-scans from LiDAR can compensate for the absence of angular velocity information available from IMU.

3.4.3 Loop-Closure

The incoming scans coming from a LiDAR are matched against a submap. Containing only a few recent saved scans, this approach can accumulate error. For only a few dozen of scans, the error is minute, but in the long term can grow to a large error. To solve the problem of error accumulation during scan-matching, Sparse Pose Adjustment (SPA) [30] is followed, which optimizes the poses of all scans and submaps. Cartographer

stores the poses of scans and submaps in memory, and during the submap building phase, if no change is observed, all the pairs consisting of scans and submaps are considered for loop-closure. The scan-matcher keeps running in parallel in the background, and if a good match is found, the resultant relative pose is added to the optimization problem.

3.4.3.1 Optimization Problem

Loop-closure optimization is also devised as a non-linear least-squares problem. Ceres is used to compute the solution as:

$$\underset{\Xi^m,\Xi^s}{argmin}\frac{1}{2}\sum_{ij}\rho\left(E^2\left(\xi_i^m,\xi_j^s,\Sigma_{ij},\xi_{ij}\right)\right) \quad (3.23)$$
(SPA)

Where the submap poses $\Xi^m = \{\xi_i^m\}_{i=1,\dots,m}$ and the scan poses $\Xi^s = \{\xi_j^s\}_{j=1,\dots,n}$ are optimized according to the given constraints. These constraints take the form of relative poses ξ_{ij} and associated covariance matrices Σ_{ij}. For a pair of submap i and scan j, the pose ξ_{ij} describes the estimate where in the submap coordinate frame scan was matched. The covariance matrix can be calculated using the covariance estimation feature from Ceres [29] with (3.21). To reduce the effect of outliers, which can appear in (SPA) (3.23) when scan-matching adds inaccurate constraints to the optimization problem, a loss function ρ, called *Huber loss*, is applied. This problem may arise, for example, in symmetric environments such as office cubicles.

3.4.3.2 Branch-and-bound scan-matching

Cartographer uses branch and bound approach to compute accurate pixel matches over large search windows effectively. The literature on the branch and bound approach is extensive; the reader is encouraged to see [31] for a short review. The main idea here is to represent the subset of possibilities as nodes in a tree, with the root node containing all the possible solutions. The children of each node form a partition of their parent, and they also together represent the same set of possibilities. Also, the leaf nodes are singletons, meaning that each leaf node represents a single viable solution. For creating a concrete algorithm, cartographer employs the method *of node selection, branching*, and computation of *upper bounds*. To read in detail about it, the reader is encouraged to study the original paper from the google authors [8].

Cartographer uses the Ceres Solver [29], an open-source C++ library for modelling and solving large, complicated optimization problems to make the processing faster. The optimization is also solved via a brute-force approach. To search for only one value of ξ, it is required to search a set of all suitable transformations between matrices. Since these optimizations require time, thus, these are done in parallel behind the scenes, and then the logged data is optimized for accuracy. Hence the system can be said to be close to real-time

3.5 Integrating PDR Model with Cartographer SLAM

To evaluate the effect of the integration of PDR, it must be integrated as a third-party system model within a SLAM engine. The current Hector SLAM opensource version does not provide access to external third-party system model while Cartographer SLAM engine

does. In the Cartographer, the Local SLAM subsystem [32] inserts a new scan into its current submap by scan-matching by using the initial guess from the pose extrapolator. The purpose of pose extrapolator is to predict the insertion of the scan into submap by making use of data from other sensors beside LiDAR. The Global SLAM subsystem runs in the background to re-arrange submaps to form a coherent global map. It can alter the trajectory and change the alignment of the submaps to with regards to loop closure to reduce the overall error. This global SLAM step is a Graph SLAM optimization that builds constraints between nodes and submaps and then optimizes the resulting constraints graph. This role of global SLAM leads us to draw an analogy with the "Graph-based SLAM" [22], in which to compute the correct data-association, the local SLAM requires an estimate of the conditional prior over robot trajectory as:

$$p(x_{1:T}|z_{1:T}, u_{1:T}) \tag{3.24}$$

Where,

- $x_{1:T} = \{x_1, \ldots, x_T\}$, trajectory nodes
- $z_{1:T} = \{z_1, \ldots, z_T\}$, perception sensor readings
- $u_{1:T} = \{u_1, \ldots, u_T\}$, control input (usually odometry information)

The 2D location information (x and y) calculated in section 2.6.4, along with the orientation information (heading or yaw data) estimated in section 2.6.1.2, is sent as a pose update to the Cartographer SLAM. Since the PDR model was used to determine the pose estimate information, it is sent as odometry information to the Cartographer SLAM. Thus, the PDR model works as a control input ($u_{1:T}$) in equation (3.24). Furthermore, the odometry information helps the state transmission model $p(x_t|x_{t-1}, u_t)$ of the "Dynamic Bayesian Network" in the SLAM process and aids in updating the pedestrian position. For

detailed analysis and working of the “Dynamic Bayesian Network” for the SLAM process, the reader is referred to [22]. Hence, sending IMU-PDR as odometry information improves the Global SLAM accuracy, which helps in making correct loop closures and bridges feature-less portions of the environment.

Chapter 4: System Prototype

This chapter describes the prototype that has been developed during the thesis work to verify the effectiveness of the proposed PDR-supported SLAM system. The prototype consists of a 2D LiDAR sensor and IMU mounted on a helmet and connected to a computing platform. On the computing platform, a real time synchronization and data logging (RSDL) firmware is implemented in C Language. Furthermore, a Remote-Controlled Software (RCS) with a sophisticated GUI was developed using Qt C++ to control and monitor the operation of the RSDL software. The chapter concludes with a brief background about the Robot Operating System (ROS), and how it was used in the experimental work.

4.1 Prototype Hardware

4.1.1 IMU-AHRS Module

The IMU used in this work is from Xsens and belongs to "MTw Awinda" series [33]. This IMU is also called an "Inertial-Magnetic Measurement Unit (IMMU)". It is a Motion-Tracker (MT) that consists of a 3D gyroscope, 3D accelerometer, and 3D magnetometer in a single package, can be combined with complex sensor fusion algorithms. The MTw is a miniature IMMU, which has a package size of $47mm \times 30mm \times 13mm$, and has a weight of $16g$. It is shown in Figure 4.1. A block diagram of the architecture is shown in Figure 4.2. In this thesis, the MTw IMU was integrated into the developed RSDL system (to be discussed in Section 4.2.1) using the micro USB cable connector to retrieve the data. The micro USB can be seen in Figure 4.1.

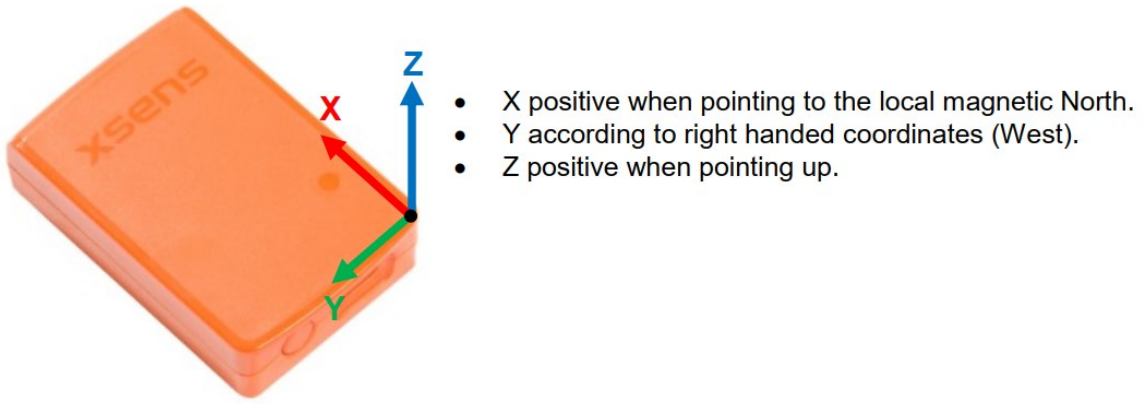

Figure 4.1: IMU - Xsens MTw Awinda

Xsens firmware implements an advanced onboard AHRS using EKF called Strap-Down Integration (SDI) [34] that can generate high accuracy in dynamic conditions independent of the output rate [35]. The output of the SDI includes the EKF-based AHRS (3D orientation) along with the calibrated gyroscope, accelerometer, and magnetometer data. The IMU also includes a thermometer for compensating the temperature dependency of the sensing elements. For more technical details and usage instructions, the reader can see [36], the Xsens MTw user manual.

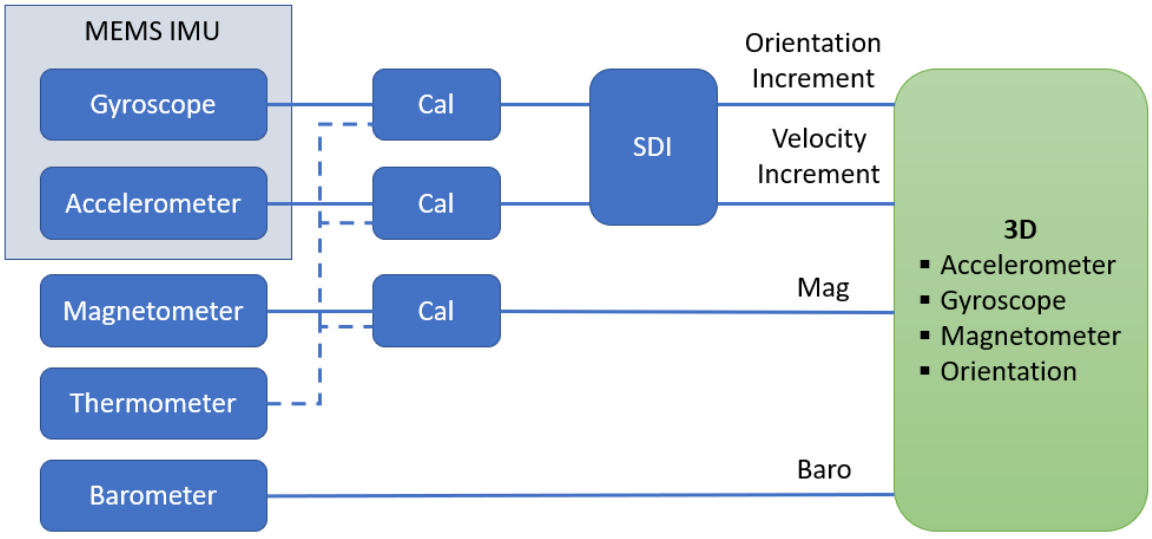

Figure 4.2: MTw Awinda Signal Processing Architecture

4.1.2 2D LiDAR Sensor

The 2D LiDAR used in this thesis work is from SLAMTEC and is called "RPLiDAR A1M8" [37]. It is based on a laser triangulation ranging principle, and it is equipped with high-speed vision acquisition and processing hardware. The system can measure distance data more than 8000 times per second. The rotating component of the RPLIDAR A1 runs clockwise to perform 360-degree omnidirectional laser range scanning of the environment and then generates an outline map of the surrounding environment. The scan rate of the LiDAR is configurable between 2-12 Hz, and it can be changed by alternating the motor PWM (Pulse Width Modulation [38]) signal. The average scan rate of the LiDAR is usually kept at 5.5Hz. RPLiDAR performance metrics are mentioned in Table 4.1. For more technical details, the reader can see [39], the RPLiDAR A1M8 datasheet. The RPLiDAR A1 with system composition is shown in Figure 4.3.

Table 4.1: RPLiDAR A1 Sensor Specifications

Item	Unit	Min	Typical	Max	Comments
Distance Range	Meter (m)	0.15	0.15-12	12	White Objects
Angular Range	Degree	n/a	0-360	n/a	
Distance Resolution	mm	n/a	<1% of the actual distance	n/a	All distance range
Angular Resolution	Degree	n/a	≤1	n/a	
Sample Duration	Millisecond (ms)	n/a	0.125	n/a	

Digital Sampling Rate	Hz	n/a	≥8000	8010	
Scan Rate	Hz	1	5.5	10	Typical value is measured when RPLIDARA1 takes 360 samples per scan

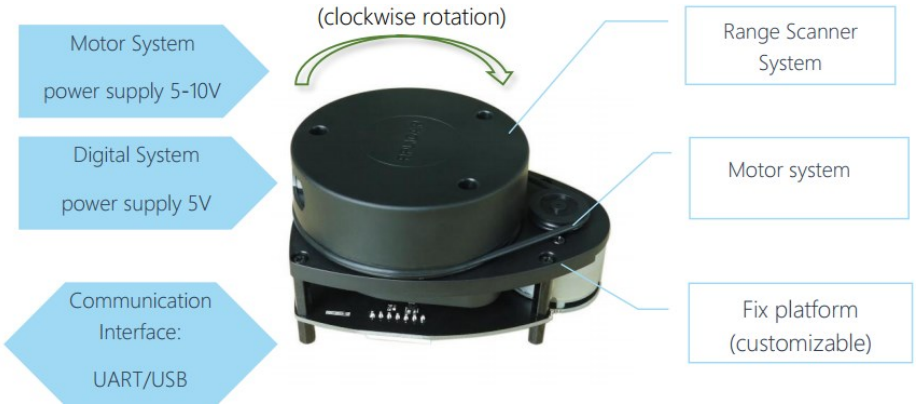

Figure 4.3: RPLiDAR A1 System Composition

4.1.3 Camera Module

To visually record the experiment and to provide a ground truth trajectory, an "Intel RealSense Tracking Camera T265" [40] was used. The T265 camera is an accurate standalone real-time 3D visual SLAM device for use in drones, IoT, Augmented Reality (AR), and Virtual Reality (VR). The T265 includes two overlapped wide-angle-of-view fisheye lens sensors, an internal IMU, and an Intel Movidius Myriad 2 VPU (Vision Processing Unit [41]) [42]. All the V-SLAM (Visual SLAM) algorithms run directly on the VPU, which allows small latency and efficient power consumption. Under the intended usage conditions, the T265 can provide under 1% closed-loop drift. For more technical details, the reader can see [42], the T265 datasheet. The RealSense provides stereo fisheye image pairs with 30 fps, accelerometer, gyroscope, and position estimation. In this work,

RealSense camera was used with the default calibrated settings, and was then configured to be integrated with the data logger (discussed in 4.2.1). Our research team tested the provided position estimation from the RealSense camera, and experiments showed that it provides an accurate trajectory in indoor scenarios with good lighting conditions. The accuracy is typically 1% of the travelled distance [42]. The T265 camera is shown in Figure 4.4.

Figure 4.4: Intel RealSense Tracking Camera T265

4.1.4 Helmet Platform

The prototype of the proposed system was developed using a helmet-mount platform. The front of the mounting system can be seen in Figure 4.5. The platform consists of a RealSense camera mounted on the front, 2D LiDAR mounted on the top of the helmet and IMU rigidly attached to the LiDAR sensor. Figure 4.6 shows the side view of the helmet. This mounting assures that both IMU and LiDAR are aligned together, so their local body-frames are coinciding.

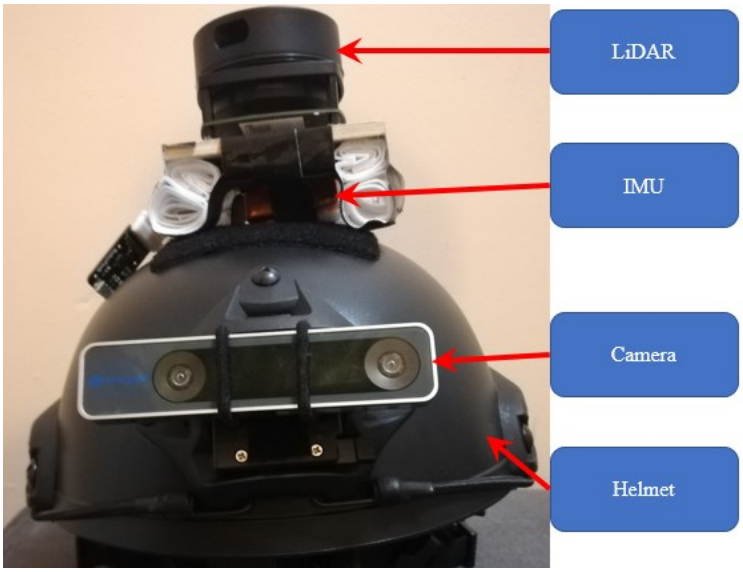

Figure 4.5: Prototype Helmet - Front-View

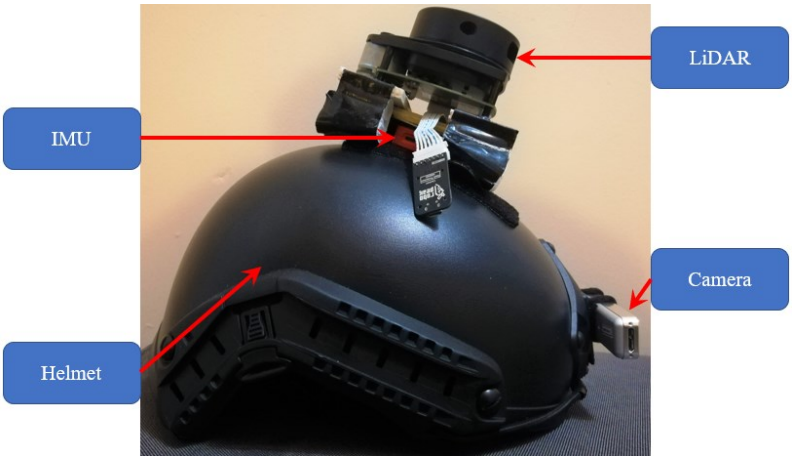

Figure 4.6: Prototype Helmet - Side View

4.1.5 Specifications of the Computing Platform

A 12” Linux Notebook was used to interface with sensors and collect experimental data. The general specifications of this computing platform are given in Table 4.2.

Table 4.2: Specifications of the Logger System

Operating System	Linux Ubuntu 18.04 OS (64-bit)
Processor	Intel i7-6500
RAM	8GB
Hard Disk-Drive (HDD)	128GB HDD

4.2 Prototype Software

A Realtime Synchronization and Data Logging (RSDL) software was developed to run on the computing platform. As the final targeted platform is an embedded system that runs this RSDL as a firmware without Graphical User Interface (GUI), a remote control software (RCS) was developed to remotely connect, communicate, monitor, and visualize the operations that are being implemented by the RSDL.

4.2.1 Realtime Synchronization and Data Logging (RSDL)

The RSDL software performs two main tasks, 1) Time synchronizing and logging the data into a single binary file for offline parsing/post-processing and 2) providing the RCS with low-rate samples of the logged data for realtime visualization, verification and presentation of the logged data. A data structure for each sensor was defined to hold its raw logged data. Each structure has a "sensor_id" as a header field, which used in identifying the sensor data packet to enable the RCS module to perform the proper data parsing and processing step for each sensor type. The second field is the time tag referenced to global start time to unify time reference for all the sensor data, which is crucial for all sensor fusion tasks. The rest of the structure is the actual sensor data. A special data structure was

created for the RSDL-RCS communication containing a field for the availability of each sensor and a full structure of each sensor data.

Five parallel threads build up the RSDL. The first thread is the main thread that synchronizes and coordinates the operation of all other threads. The flowchart of this main thread is shown in Figure 4.7, and it performs the following tasks:

1. Creating the binary log file along with a file access synchronization mutex.
2. Configuring individual sensors interfacing threads with synchronization mutex.
3. Provide the threads of the sensor with a global time and a log file reference
4. Open a communication channel with the RCS.
5. Receive the control commands from the RCS.
6. Send samples from the logged data to the RCS for visualization/monitoring.

The remaining four threads are assigned to each sensor. The flow chart of the sensors thread is shown in Figure 4.8. Each of these threads receives a global start time from the main thread to reference all the logged data to that common timing reference value. Each sensor thread is responsible for:

1. Open a communication channel with the sensor
2. Configure the sensor
3. Create a data buffer 100 times as large as the sensor data structure
4. Start processing the received sensor data
5. Share samples of the logged data along with the sensor status with the RCS every one second

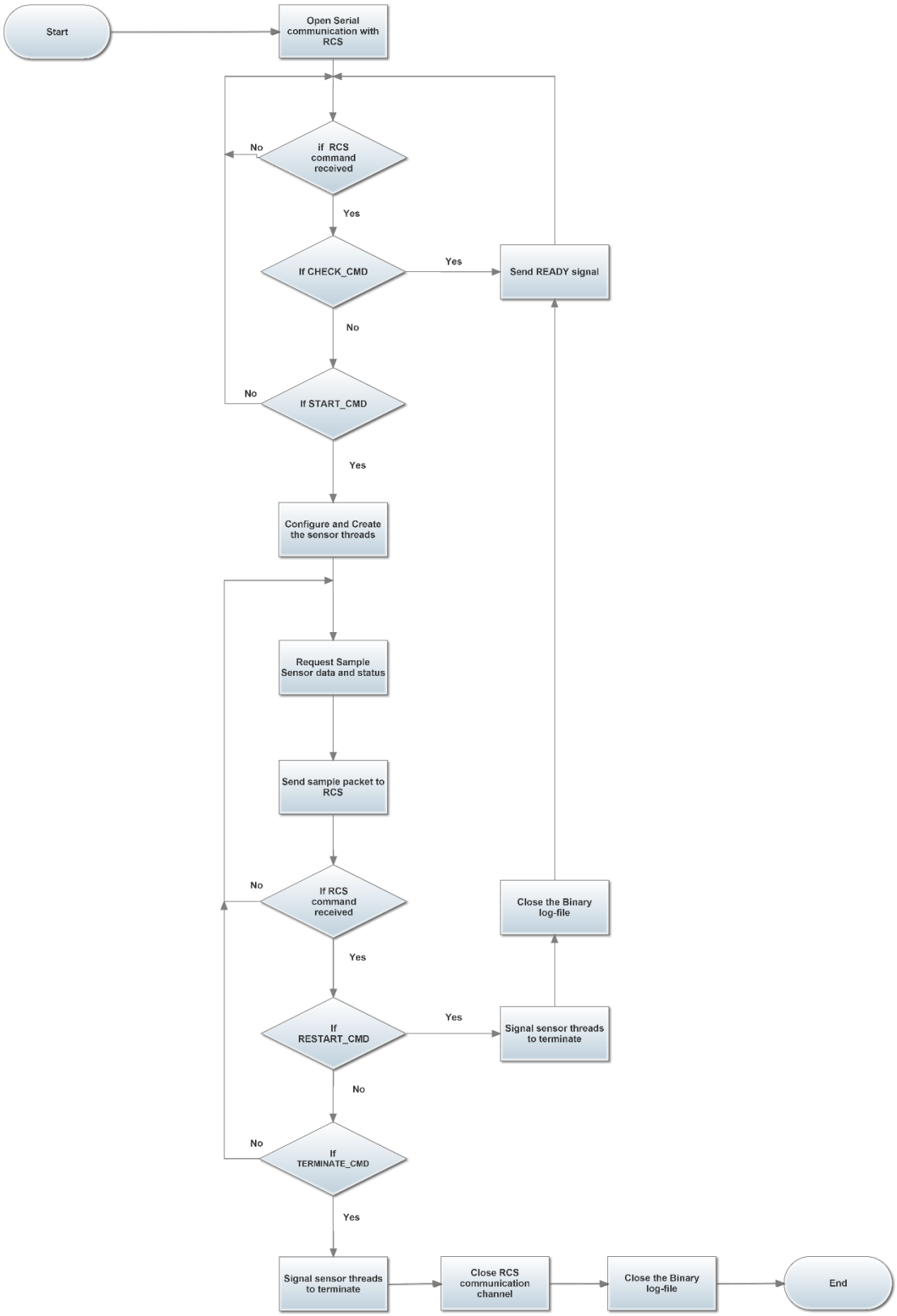

Figure 4.7: Main Thread - Flow Chart

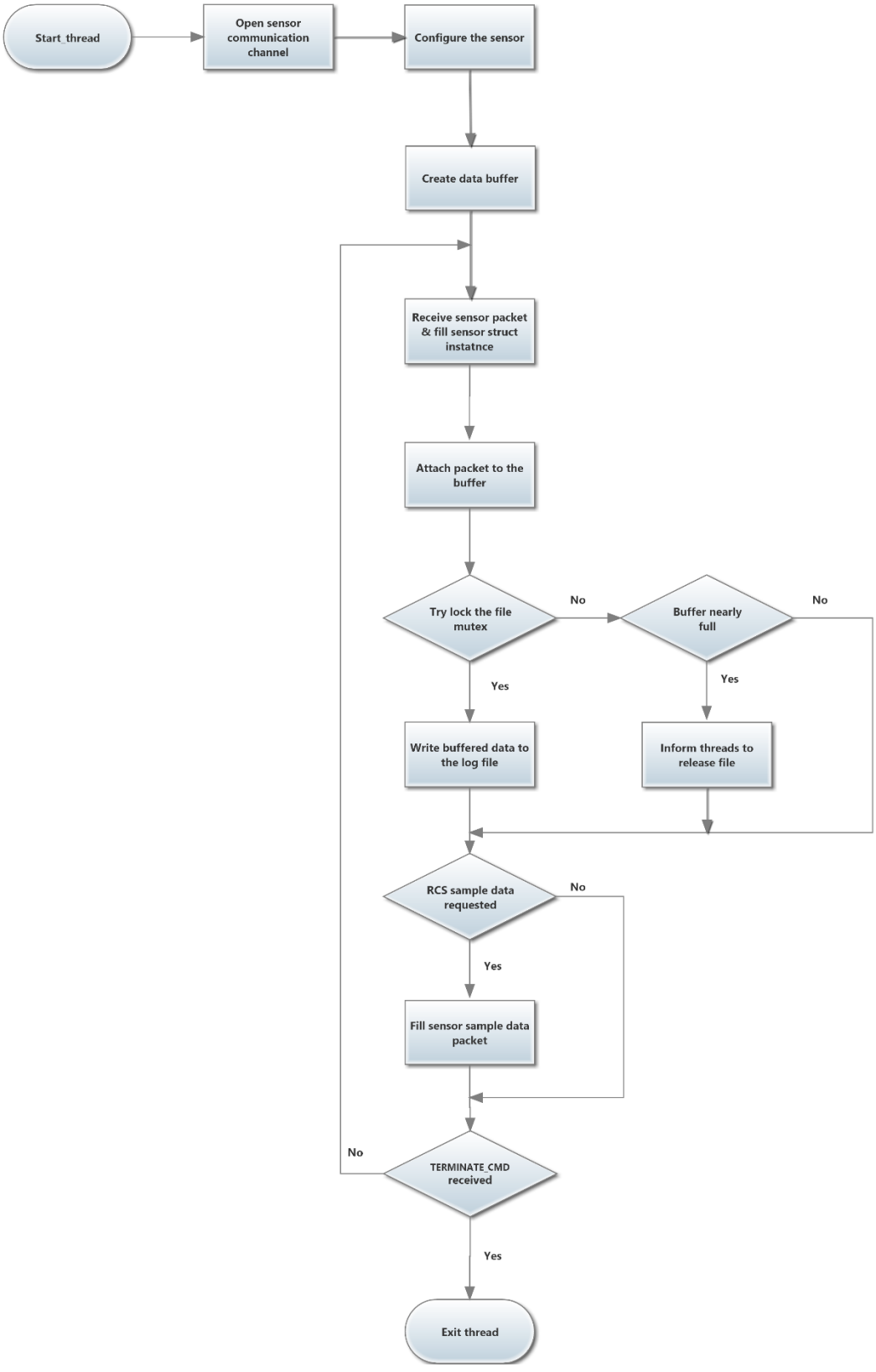

Figure 4.8: Sensor Thread - Flow Chart

In each iteration of any of the sensor-threads function loop, it fills out an instance of the sensor data structure and writes the data to the logging file after synchronizing the access with other threads. The logged data is attached to the data buffer, and in case of having nearly full buffer, the thread informs all other threads and forcibly locks the file in the next cycle and writes all the filled buffer.

4.2.2 Remote-Control Software (RCS)

The developed RSDL software was designed to work on an embedded platform without a GUI. In practice, this embedded platform will be the helmet connected to the LiDAR, IMU, and the Camera sensors. To control and monitor the operations of this embedded platform, a remote-control software (RCS) with sophisticated GUI was developed. As the IMU, LiDAR and Stereo Camera ouput data at high frequency, a large number of packets are being saved by the RSDL module. To monitor and visualize this process, RCS is developed as a fast control, monitoring, and visualization system. To guarantee efficient realtime performance, C++ programming language within Qt development framework was used. Below is a brief description of the Qt-Framework and the developed RCS with its GUI views.

4.2.2.1 Qt-Framework

Qt (“Cute”) [43] is a free and open-source widget toolkit for creating graphical user interfaces and cross-platform applications that run on various software and hardware platforms with little or no change in the underlying codebase. Though Qt can be used with different languages, we worked with C++, due to the reasons already mentioned previously. The version of Qt which we worked with is Qt5 - C++.

4.2.2.2 Functionalities

The RCS module has two main features, which are the offline processing of the data collected by the RSDL module and real-time monitoring of the RSDL module during real-time logging. The RCS module has four tabs (views), called File, Visualize, Map and Terminal.

4.2.2.2.1 Offline Data Processing

The data collected by the RSDL module is in binary format (.bin). The RCS module from the "File" tab opens the file and shows the size of the file, as shown in Figure 4.9. On pressing the "Read File" in the same tab, the RCS module starts parsing and decoding the data for future visualization and post-processing. During this state, the RCS module also generates the comma-separated values (.csv) files of IMU, Magnetometer, Camera and GPS data in the same directory of the RCS module executable.

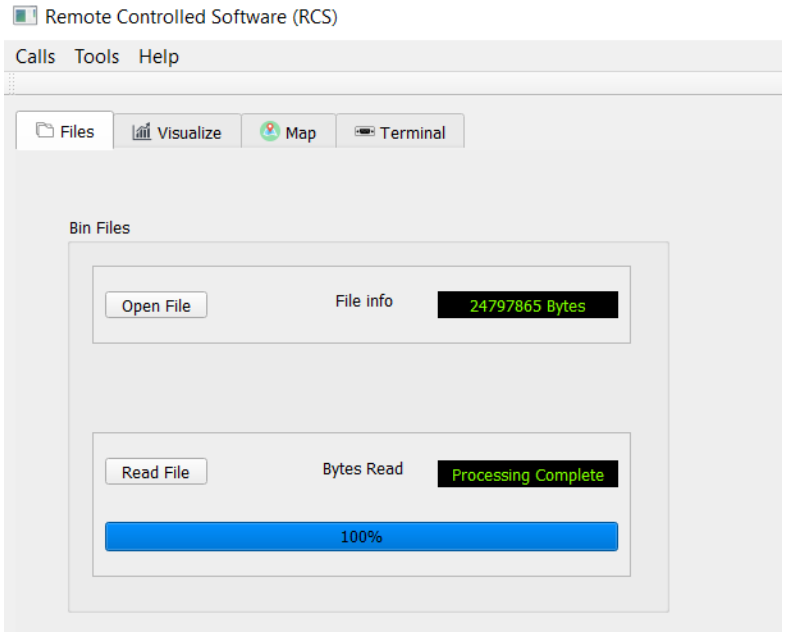

Figure 4.9: RCS - File Tab

Once this stage is complete, the next tab called “Visualize”, can plot the IMU data readings on the graph. The “Visualize” tab is shown in Figure 4.10. When the ‘Run’ button is pressed, the graph will start plotting the data. It plots the 3 Accelerometers (x, y, z) and 3 Gyroscope (x, y, z) readings.

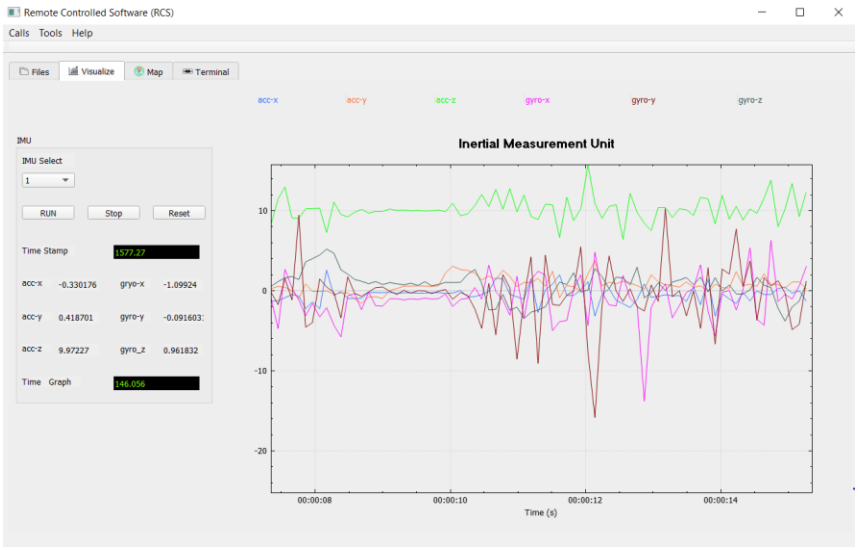

Figure 4.10: RCS - Visualize Tab

Another tab called “Map” is shown in Figure 4.11. It displays a plot of the current pose on OpenStreetMap (OSM) map [44].

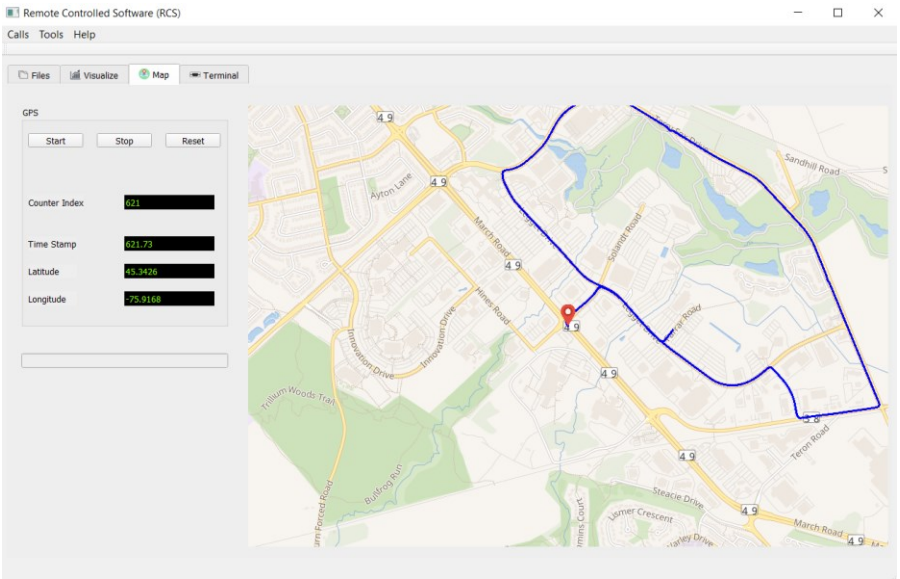

Figure 4.11: RCS - Map Tab

4.2.2.2.2 Realtime Data Logging

During the online data processing phase, the RCS module communicates with the RSDL module via serial communication. The connection setting parameters between the two modules are entered in the “Configure” selection tab, as shown in Figure 4.12. The serial port settings option which can be set from the RCS is shown in Figure 4.13. The visualization of the status of the LS module can be seen in the “Terminal” tab, as in Figure 4.14.

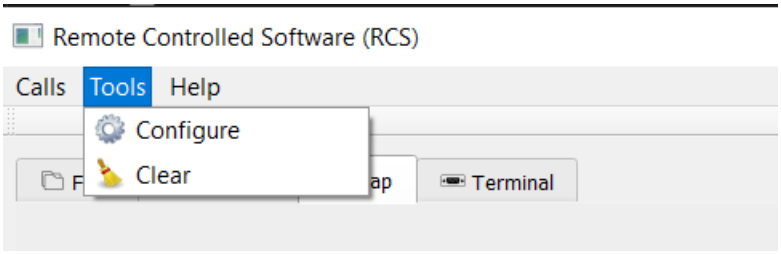

Figure 4.12: RCS - Configure Option

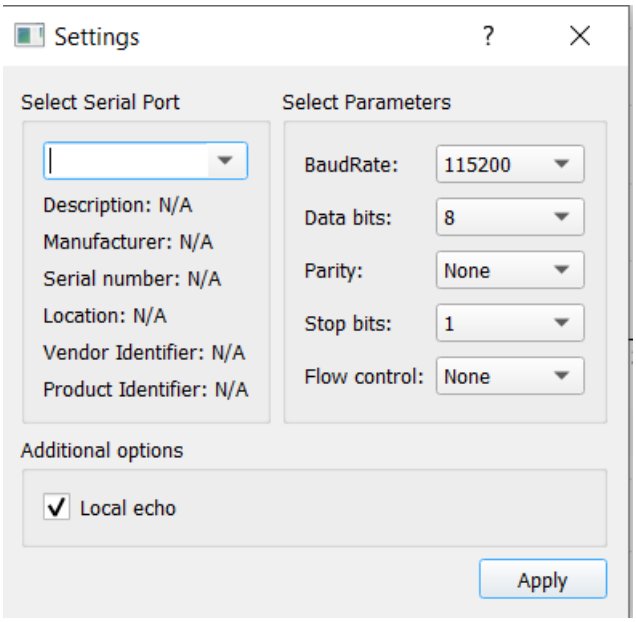

Figure 4.13: RCS - Serial Port Settings Box

The baud rate for the communication is set at 115200, and on selecting the proper "COM Port", the "Connect" selection tab can be selected from the dropdown box of "Calls" tab present at the top of the RCS GUI module. If the connection is successful, at the bottom, a "Connected to COMx" status will be displayed, otherwise an "Error" message will be displayed. The two modules can be disconnected by selecting the "Disconnect" tab.

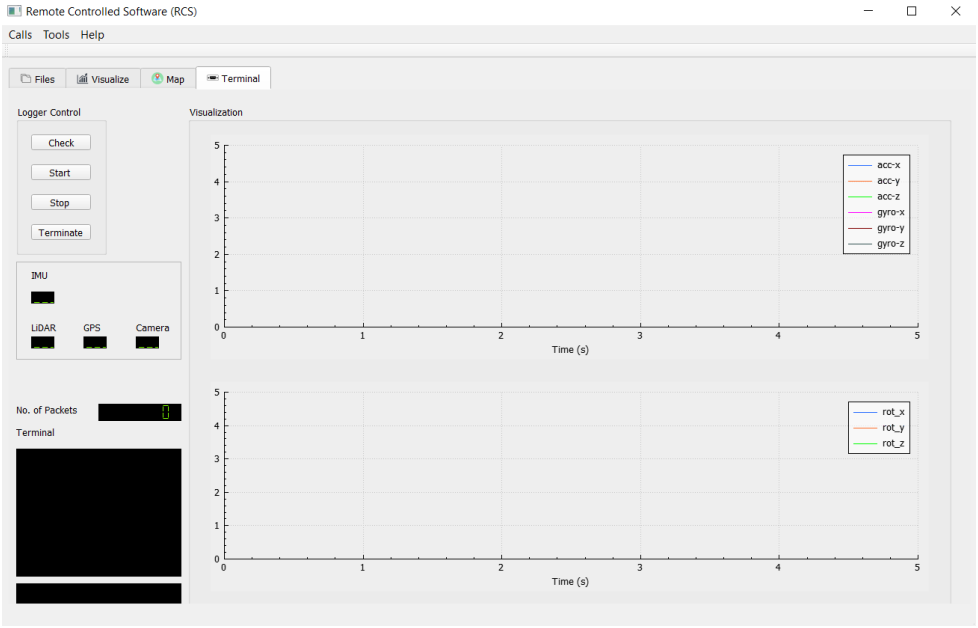

Figure 4.14: RCS - Terminal Tab

Once the connection is successful, from the "Logger Control" region in the "Terminal" tab, different buttons options are provided for the user with their functions mentioned below:

1. *Check:* This button is generally used to check the state of the logger, whether it is in running state or not.
2. *Start:* On pressing this button, the logger is started. It starts by sending a byte "B" serially to the RSDL module. Once the packet is transferred, then the terminal box

present at the bottom displays the “Start” word stating the button was pressed, and the packet was transferred.

3. *Stop:* If this button is pressed, the logger is stopped. It stops the running of program by sending a byte “S” serially to the LS module. Once the packet is transferred, then the terminal box present at the bottom displays the “Stop” word stating the button was pressed, and the packet was transferred.
4. *Terminate:* On pressing this button, the logger is terminated. It terminates by sending a byte “T” serially to the RSDL module. Once the packet is transferred, then the terminal box present at the bottom displays the “Terminate” string, stating the button was pressed, and the packet was transferred.

The frequency with which the RSDL module sends the packet is kept at 1Hz. The RCS module parses the received packet and shows the status of each sensor connected to the RSDL module. On the right side, in the current “Terminal” tab, the two graphs are present for visualization. One plots the IMU data, which includes 3 Accelerometers (x, y, z) and 3 Gyroscope (x, y, z) readings. The second graph plots the solution sent from the camera. Each second a packet arrives, the graphs get updated automatically, giving the user insight on the data being currently logged by the RSDL.

4.3 Robot Operating System (ROS)

Both Hector SLAM and Google Cartographer SLAM uses the Robot Operating System (ROS) software [45] for data processing. This section will provide a brief background of ROS and how it was used in this work.

4.3.1 Robot Operating System (ROS) – Brief Background

ROS is a BSD-licensed based system containing a set of libraries and tools for building robot applications. A ROS system consists of several independent nodes, which communicate with each other using a communication or a messaging model called the "Publisher-Subscribe" messaging model [46] between nodes. For example, a sensor's driver model could be implemented as a node and is also capable of transmitting or publishing sensor data as a stream of messages. These messages then could be subscribed or used by any other nodes, including filters, loggers, and also high-level systems such as navigation, guidance, SLAM engines, etc.

ROS has a "ROS Master" node that controls the operations of other nodes. ROS Master and Node model relationship is shown in Figure 4.15. For example, LiDAR and IMU data can be remotely seen on a laptop or PC and also can be serialized to SLAM engines. The system diagram is shown in Figure 4.16. The system consists of a "ROS Master", and three nodes. The "LiDAR Node" takes care of the communication of the LiDAR, a "SLAM Engine Node" which processed the laser-point clouds and a "Laser Point Cloud Display Node" which simply displays the point clouds. All three nodes are registered with the "ROS Master". The "LiDAR Node" publishes a "laser_scan" message and is

subscribed by the other two nodes. For more information about ROS, the reader is encouraged to see this [47].

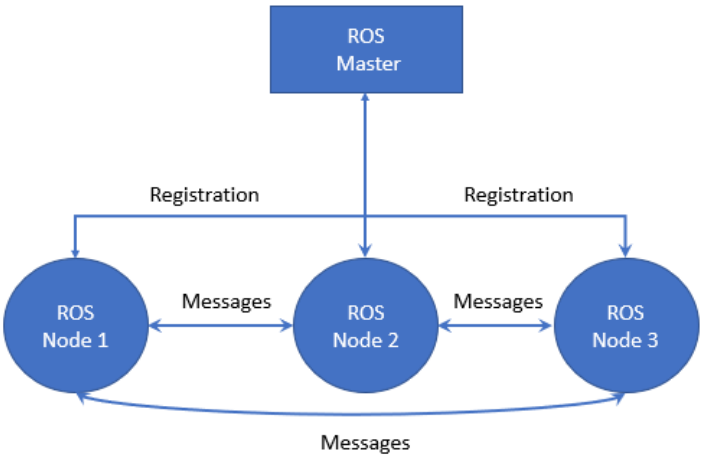

Figure 4.15: ROS Master and Node Model

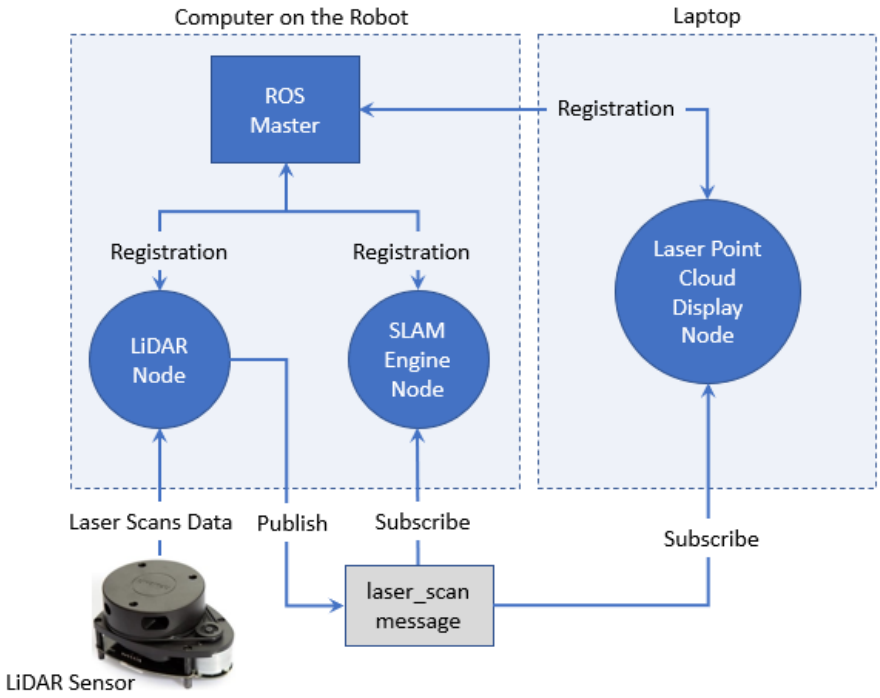

Figure 4.16: ROS - LiDAR example

4.3.1.1 Using ROS with the Proposed System

The two SLAM engines, Hector SLAM and the Cartographer SLAM have open-source ROS nodes available online. To run these engines, they need the data in the ROS custom data format, which is called "ROS message" [48]. Thus, data in any other format must be converted to the ROS message format to be availed by the discussed SLAM engines. As the RSDL software encodes the data in a binary format, the decoded data was converted to the corresponding ROS message format and was then published from a node on a topic to the ROS master to be processed by the Hector and Cartographer SLAM. The LiDAR, IMU and PDR data were converted to the ROS messages format using the format described in [49]. The developed PDR model is used for estimating the pose update and is sent to the Cartographer SLAM engine as a ROS message. Therefore, the pose estimate information is changed to the "Odom" message [50] to be available for interaction with the Navigation Stack of the ROS and is published on a topic. The configuration file of the Cartographer was modified to accept the PDR pose estimates as ROS pose odometry messages.

Chapter 5: Experiments and Results

This chapter demonstrates the testing environment and the collected datasets on which the developed algorithms were applied. Three datasets were collected and four different localization techniques were applied on the collected data: 1) IMU PDR standalone, 2) Hector SLAM, 3) Cartographer SLAM, and 4) Modified Cartographer where PDR model is integrated (the proposed system).

When in operation, the internal map of the environment is stored in memory buffer of the RealSense camera, allowing it to recognize the places which it has visited or seen before to provide a consistent pose for that environment. The camera tries to keep the map for the most recently/seen location and has a fixed size, this feature allows the RealSense solution to limit the drifts. Thus, by keeping the maps of the locations it has seen before, and looking for loop-closures increases the accuracy of the pose estimated result. In our experiments (discussed subsequently), we ended all of our experiments by visiting the location from where we started to have a loop-closure. By having this loop-closure, the visual drifts are compensated and we thus retrieve more accurate position estimation from the RealSense. All experiments were performed under good stable lighting conditions with rich visual features. Under this bright and visually rich environment, the RealSense Camera solution showed accurate and robust performance. Therefore, the RealSense Camera solution was adopted as a ground truth to compare the results of the collected datasets.

5.1 Testing Environment

All the tests were performed at Carleton University. Two datasets collection were performed in the Minto building. One dataset had a travelled path close to a rectangle, and

the other was close to a figure eight shaped rectangle. The third dataset was performed in a multi-floor environment in Mackenzie building involving walking downstairs and coming back to the same spot by walking upstairs. The floor plans of the environment where the datasets were collected is shown in Figure 5.1. Figure 5.2 shows the pictures taken during the experiments and data collection in Minto Building. Table 5.1 shows the total distance travelled for each collected dataset as calculated by the ground truth system.

Figure 5.1: (a) Rectangular-Shaped Dataset (b) Eight-Shaped Dataset (c) Multi-Floor Dataset

Table 5.1: Total Distance travelled (m)

Trajectory	Rectangle Shaped Path	Eight Shaped Path	Multi-Floor Path
Travelled Distance	42m	87m	122m

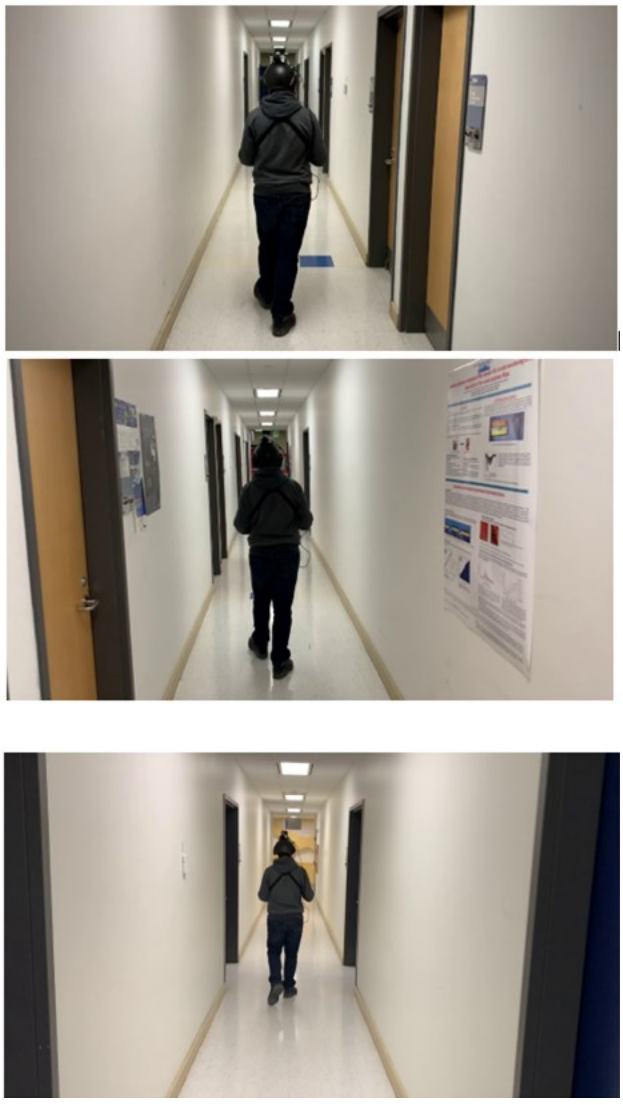

Figure 5.2: Snapshots taken during experiments

5.2 Rectangle Shaped Dataset Results

5.2.1 Ground Truth Solution

The ground truth solution generated by the RealSense camera system is shown in Figure 5.3. The figure is also annotated to let the readers understand the start/stop positions along with the sequence of the trajectory traversal. RealSense solution has showed high accuracy proven by the trajectory shape and the loop error which was sub-meter in all our experiments.

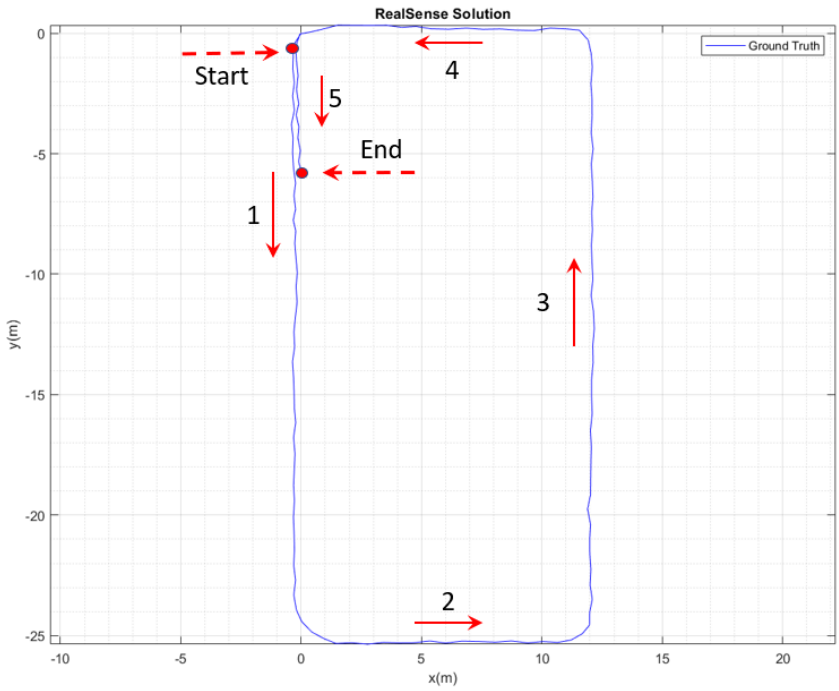

Figure 5.3: Ground Truth Solution for Rectangle Shaped Dataset

5.2.2 IMU and PDR Model Results

The PDR steps calculated using the technique from Section 2.6.2 on the resultant acceleration data of the 3D accelerometers is shown in Figure 5.4. The heading angle and the trajectory generated from the IMU-based PDR system model is shown in Figure 5.5 and Figure 5.6 respectively. As can be seen in Figure 5.6, the drifts in the PDR solution is

significant. The main factors for these drifts are step-length error combined with orientation error.

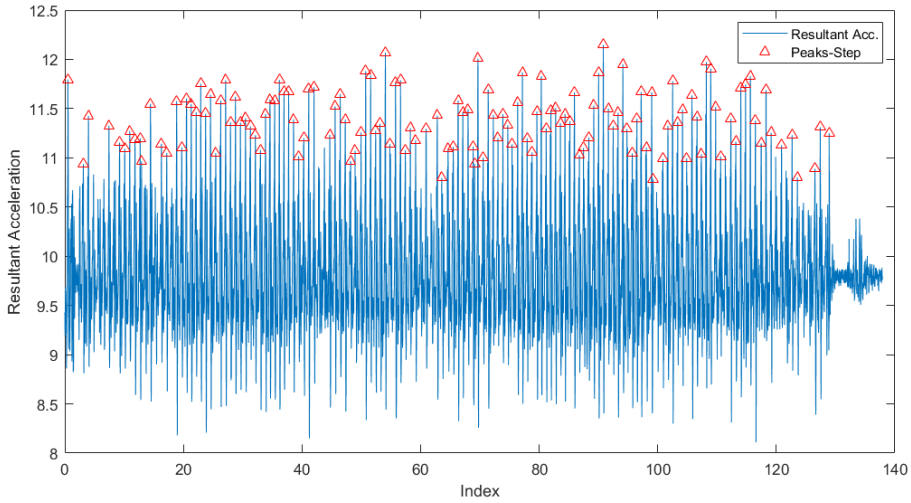

Figure 5.4: Detected Steps for Rectangle Shaped Dataset

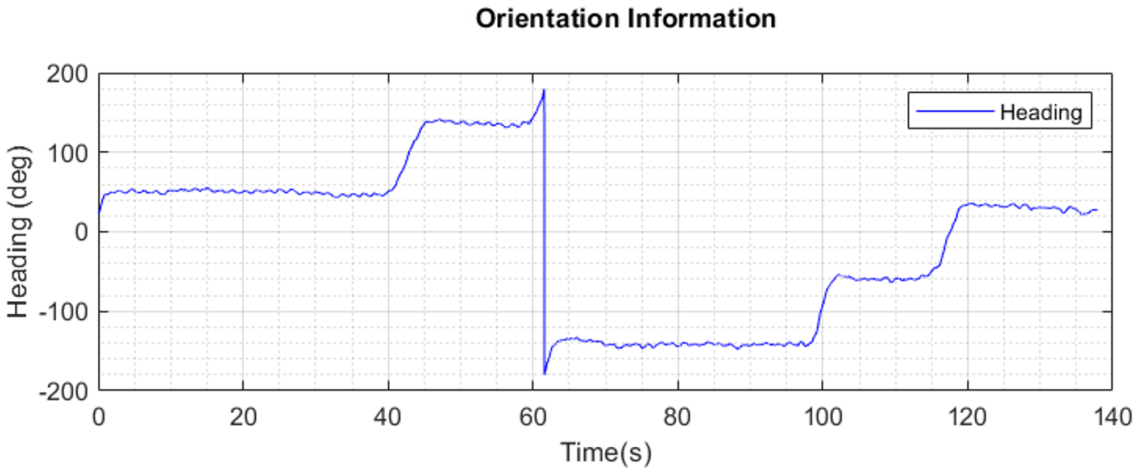

Figure 5.5: Heading Angle for Rectangle Shaped Dataset

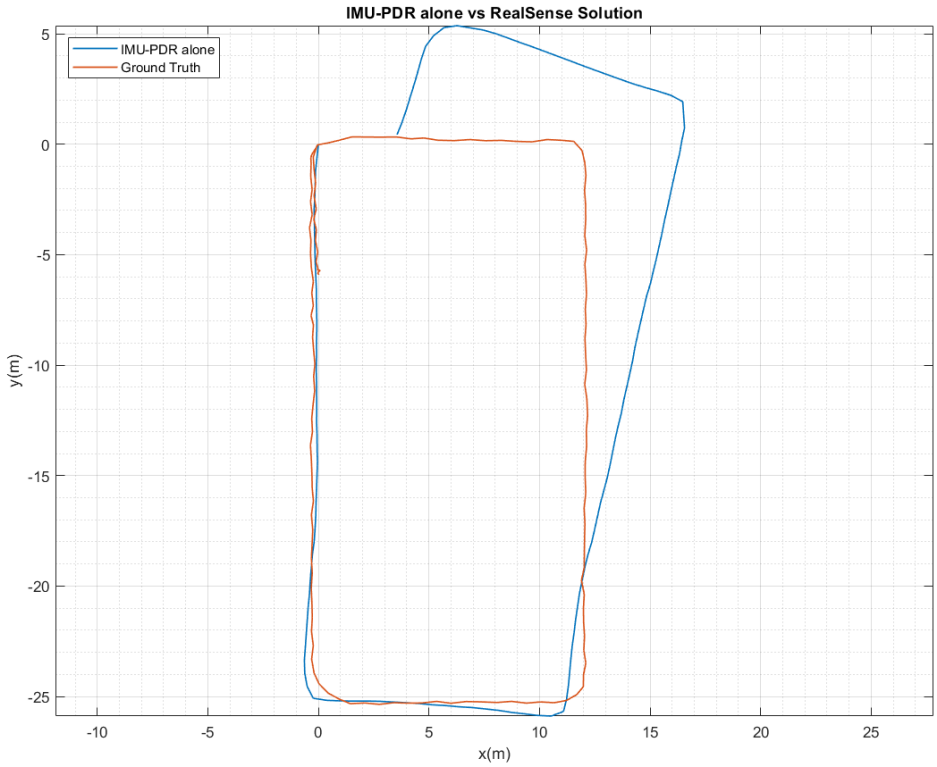

Figure 5.6: PDR vs Ground Truth for Rectangle Shaped Dataset

5.2.3 Hector SLAM Result

The localization result retrieved from the Hector SLAM using only the LiDAR scans is plotted in Figure 5.7. It is evident from the figure that the Hector SLAM struggled with the featureless portions. Due to the absence of accurate odometry or system model, Hector SLAM was unable to update its position and remained stagnant for most period of the time during the path portion 3 (from Figure 5.3). Hector SLAM showed a scale problem in path portion 2. Still, due to the unavailability of good features primarily during path 3 segment, it could not generate a good-shaped trajectory that resembles the ground truth.

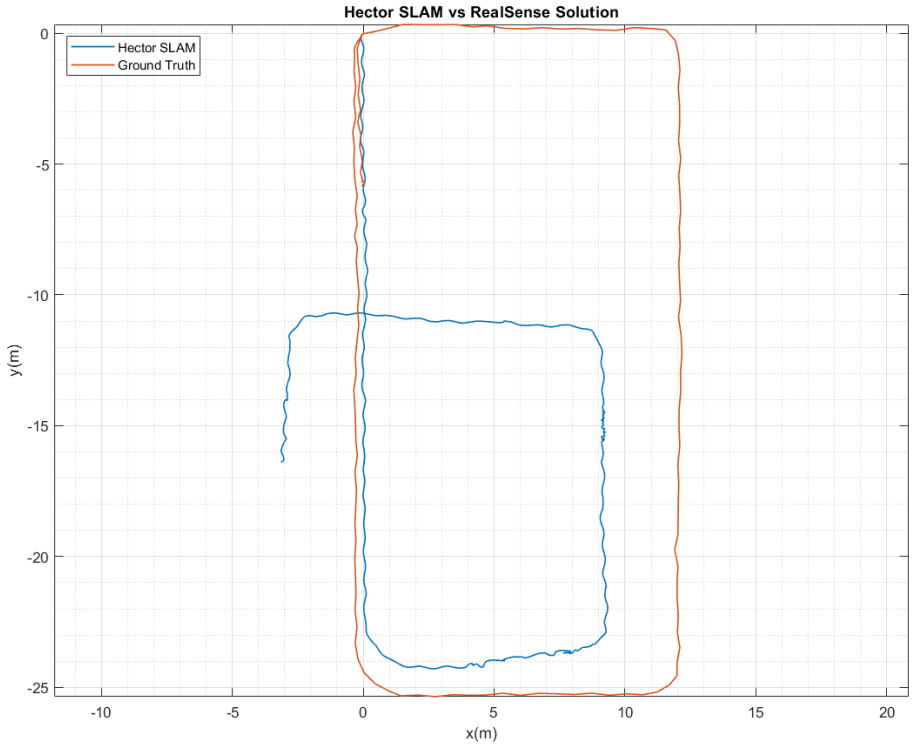

Figure 5.7: Hector SLAM vs Ground Truth for Rectangle Shaped Dataset

5.2.4 Cartographer SLAM with LiDAR only

Figure 5.8 shows the trajectory plot from the Cartographer SLAM using LiDAR scans only. Compared to the Hector SLAM result shown in the preceding subsection, Cartographer SLAM with LiDAR only showed better accuracy against featureless portions. Performance deteriorated during the featureless surrounding during segment 1 (see Figure 5.3) because of which we can see a turn earlier than the ground truth solution. When the shape of the trajectory is compared with the ground truth solution, it is also not close to it. This shows that in the absence of a reliable system model that accurately models person's motion, the accuracy of the Cartographer SLAM degrades.

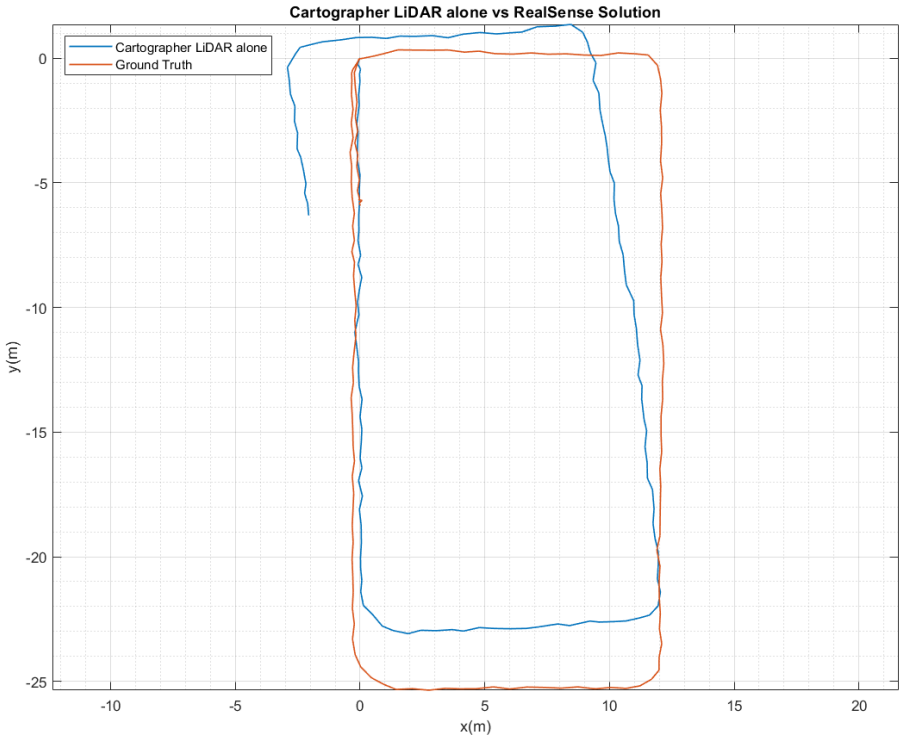

Figure 5.8: Cartographer SLAM vs Ground Truth for Rectangle Shaped Dataset

5.2.5 Proposed Solution Result

The localization result from the Cartographer SLAM using the PDR-aided model along with the LiDAR scans is shown in Figure 5.9. The figure depicts the closeness of the proposed model trajectory to the ground truth solution. Our proposed system generates superior performance by showing robust performance during featureless environments as compared to the Hector SLAM and Cartographer SLAM results used with LiDAR only. The generated map is shown in Figure 5.10. This shows that if proper system model that accurately models person's motion is used, the accuracy of the Cartographer SLAM significantly improves.

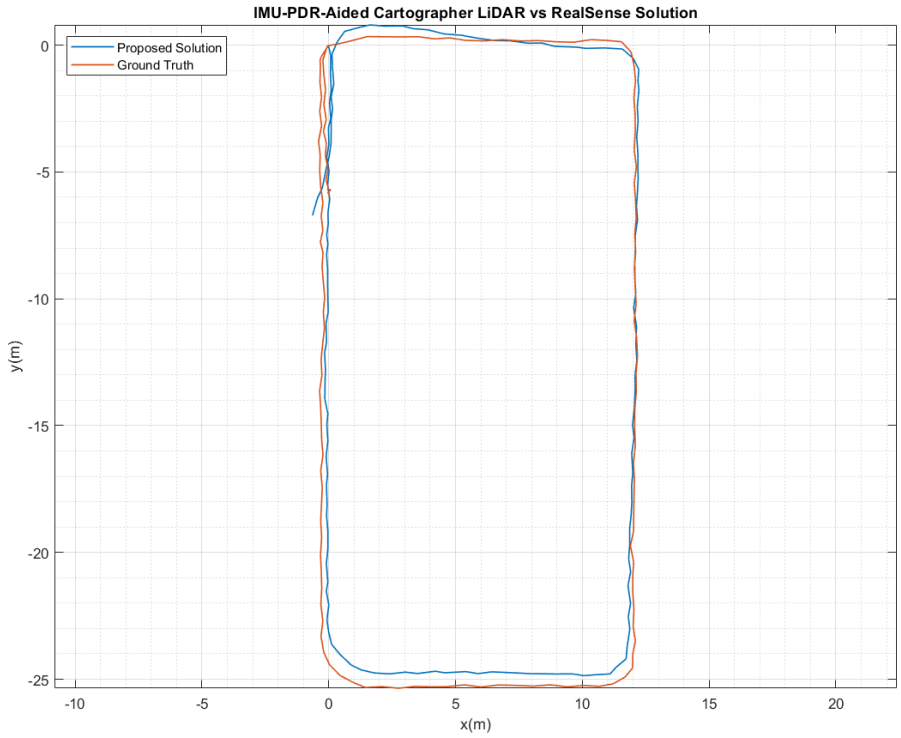

Figure 5.9: Proposed Solution vs Ground Truth for Rectangle Shaped Dataset

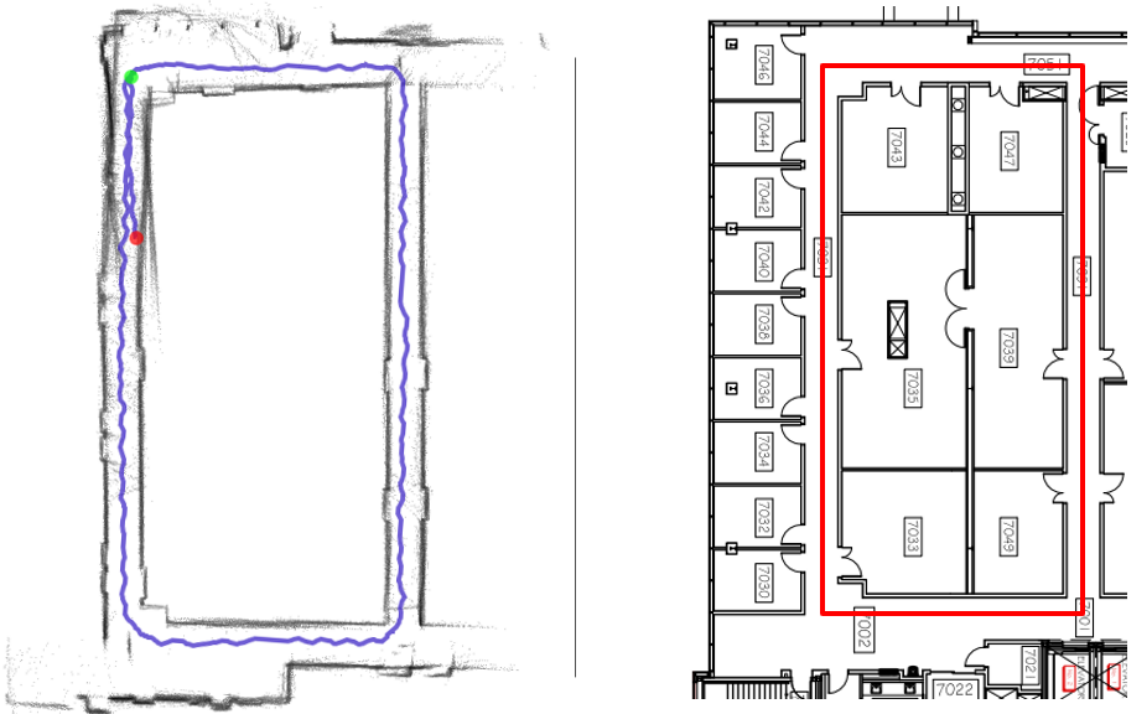

Figure 5.10: Generated Cartographer Map vs Reference Floor Plan

5.3 Eight Shaped Dataset Results

5.3.1 Ground Truth Solution

This dataset is longer with more turns. The ground truth solution is shown in Figure 5.11. The figure is also annotated to let the readers understand the start and stop positions along with the sequence of the trajectory traversal.

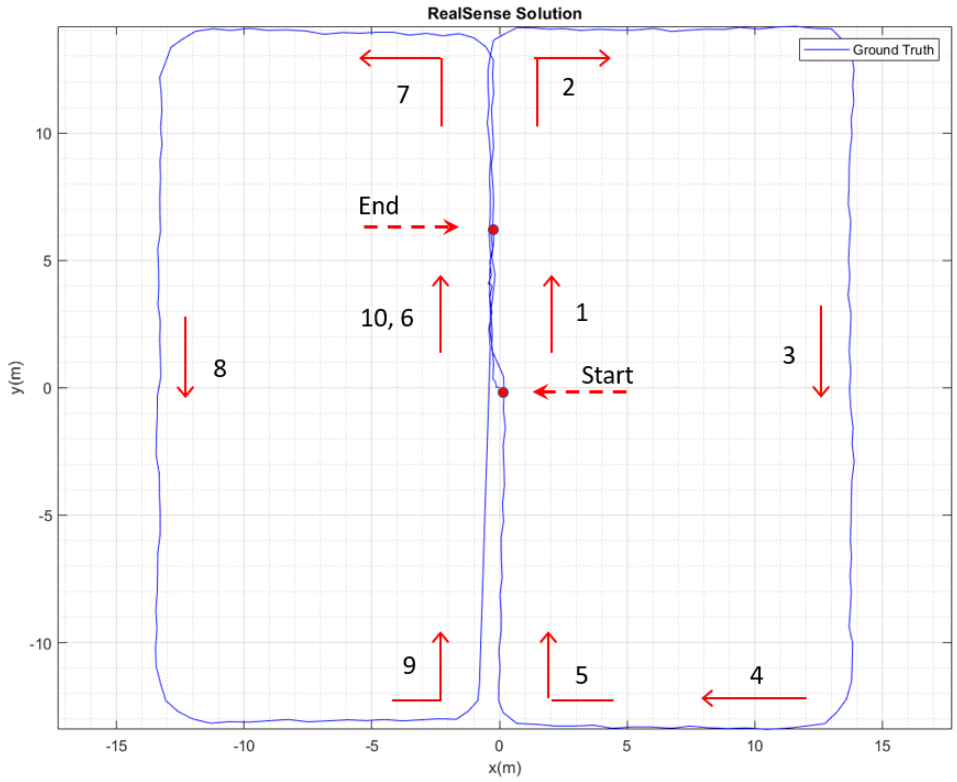

Figure 5.11: Ground Truth Solution for Eight Shaped Dataset

5.3.2 IMU and PDR Model Results

The PDR detected steps are shown in Figure 5.12. Note that during the stationary periods, no steps are detected and hence no step forward motion is applied. The heading angle and the trajectory generated from the IMU-based PDR system model is shown in Figure 5.13 and Figure 5.14 respectively. The apparent spikes in heading angle is not a glitch or error in heading angle. Instead, these apparent spikes are due to angle cyclicity

when the heading angle is close to -180/180 degrees. Despite the apparent heading accuracy shown in Figure 5.14, the PDR solution drifted mainly due to step-length error accumulation.

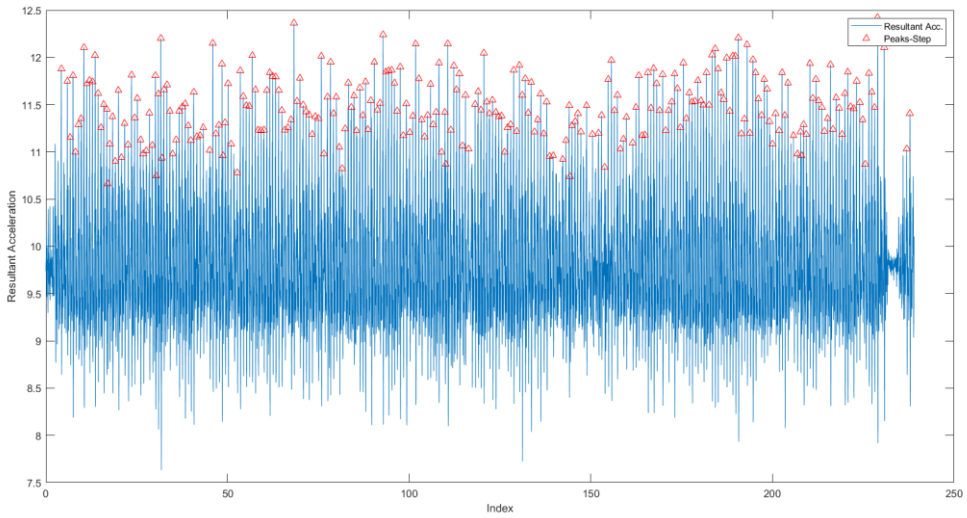

Figure 5.12: Detected Steps for Eight Shaped Dataset

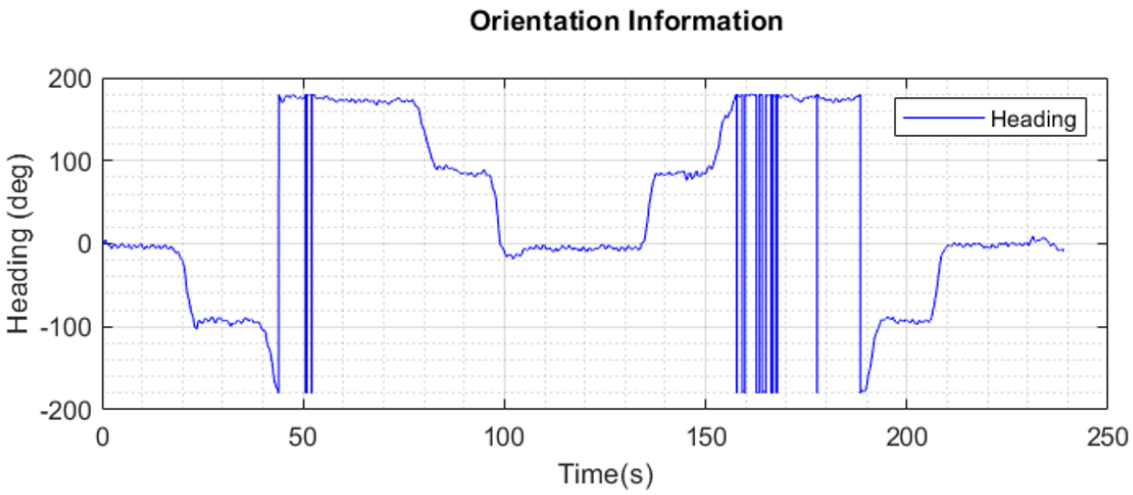

Figure 5.13: Heading Angle for Eight Shaped Dataset

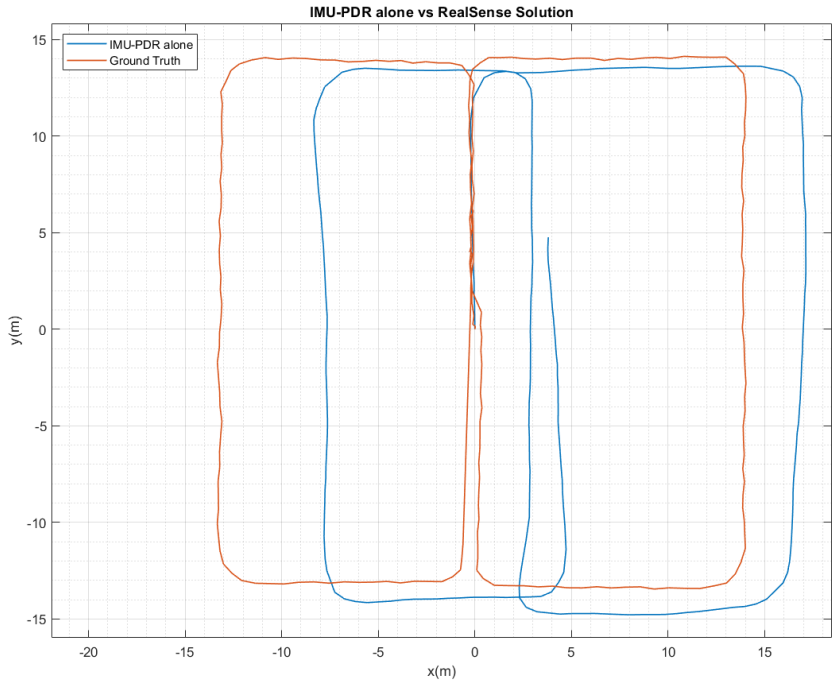

Figure 5.14: PDR vs Ground Truth for Eight Shaped Dataset

5.3.3 Hector SLAM Result

The localization result fetched from the Hector SLAM employing only the LiDAR scans is shown in Figure 5.15. The drifts in Figure 5.15 confirms the limitations of Hector SLAM that have been demonstrated in the rectangle dataset. It is clear that the Hector SLAM conflicted with the featureless portions for almost half of the experiment's duration. After taking the start, during the path segment between 1 and 2 (from Figure 5.11), it failed to perform correct scan-matching for almost half of the trajectory, and thus the location was updated for half the portion only. For the path portion 3, it struggled to observe good features to detect the movement of the platform and hence remained still for most of the time. Similarly, it could not pick features during the path portion from 5 to 7. But Hector SLAM was able to detect the environment features during the motion for path portion from segment 7 to segment 10. Due to the incapability of detecting good features during the

initial path traversal, Hector SLAM could not generate a precise trajectory that is close to the ground truth.

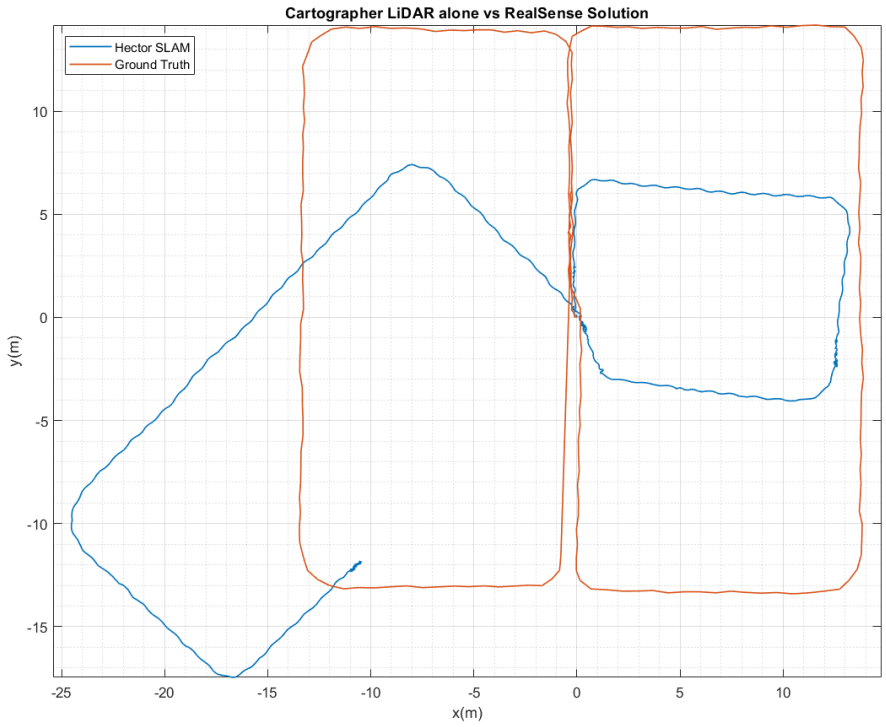

Figure 5.15: Hector SLAM vs Ground Truth for Eight Shaped Dataset

5.3.4 Cartographer SLAM with LiDAR only

Figure 5.16 illustrates the trajectory plot from the Cartographer SLAM using LiDAR scans alone. Compared to the Hector SLAM result presented in the preceding subsection, Cartographer SLAM with LiDAR only displayed better stability against featureless portions. It had trouble doing scan-matching against the featureless surrounding during path segment from segment 1 to segment 2 (from Figure 5.11), because of which we can observe a turn sooner than the ground truth result. Due to this inability to detect good features earlier in the experiment, the overall shape of the trajectory does not mirror the

ground truth solution. The effect of augmenting Cartographer with PDR as a system model is demonstrated in the following subsection.

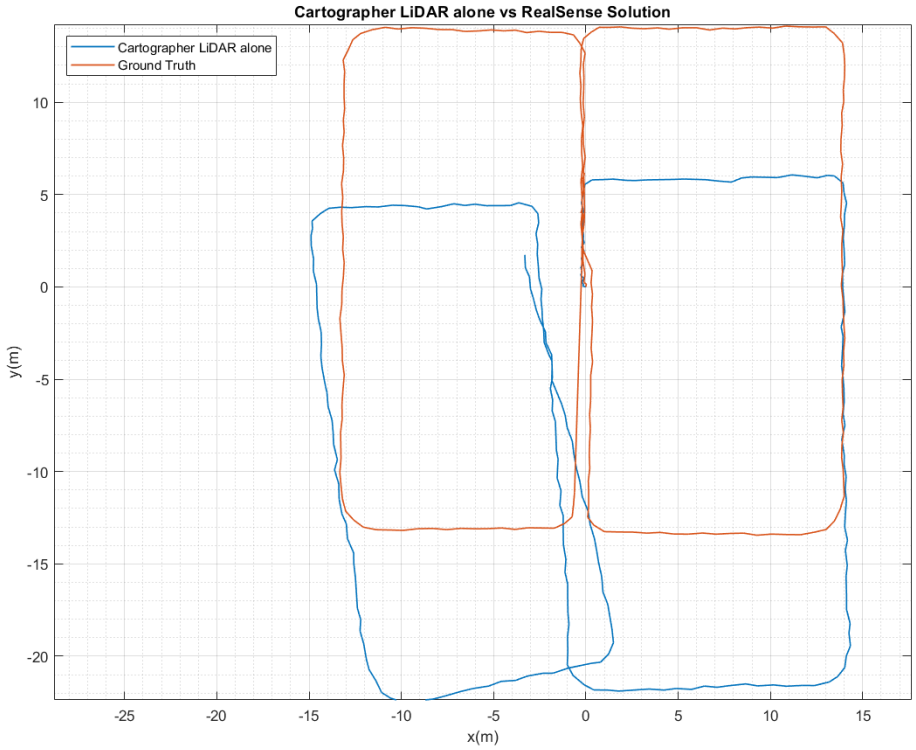

Figure 5.16: Cartographer SLAM vs Ground Truth for Eight Shaped Dataset

5.3.5 Proposed Solution Result

The localization trajectory from the Cartographer SLAM using the PDR-aided model along with the LiDAR scans is displayed in Figure 5.17. The figure depicts the closeness of the proposed model trajectory to the RealSense solution (ground truth). It was able to bridge the gap against the featureless portions during the path section from segment 1 to segment 2 (from Figure 5.11), where both the Hector SLAM and Cartographer SLAM using LiDAR scans only strived. The proposed PDR-aided system generates superior performance by showing robust performance during featureless environments as compared

to the Hector SLAM and Cartographer SLAM results used with LiDAR only presented in the previous two subsections.

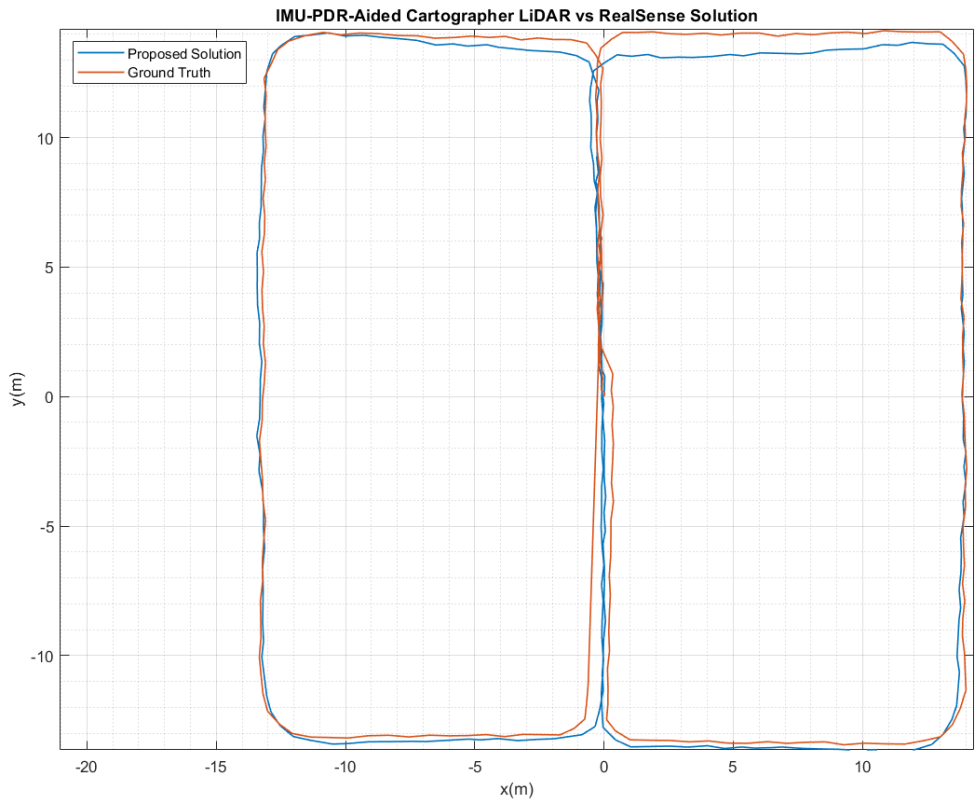

Figure 5.17: Proposed Solution vs Ground Truth for Eight Shaped Dataset

5.4 Multi-Floor Dataset Results

This dataset is more challenging as it contains upstairs and downstairs portions. The purpose of choosing the Multi-Floor environment was to analyze the behavior of the SLAM when traversed through multiple floors. Plus, the aim was also to study the behavior of the scan-matching when the pedestrian is walking down and climbing up the stairs. Since, the features change rapidly during the path section containing stairs, it was desired to

investigate the behavior of the SLAM engine when it receives an odometry information sent from the PDR module to the SLAM engine, and inspect its effect on the Local SLAM. In addition, for future plans (discussed in section 6.2), the data collected from this dataset will be an asset with running simulations and developing algorithms when we move towards 3D SLAM solution from 2D SLAM solution. Although the dataset includes 3D motion, scan-matching and system models were able to capture 2D motion.

5.4.1 Ground Truth Solution

The ground truth solution generated by the RealSense camera system is shown in Figure 5.18. The figure is also annotated to let the readers understand the start and stop positions along with the sequence of the trajectory traversal.

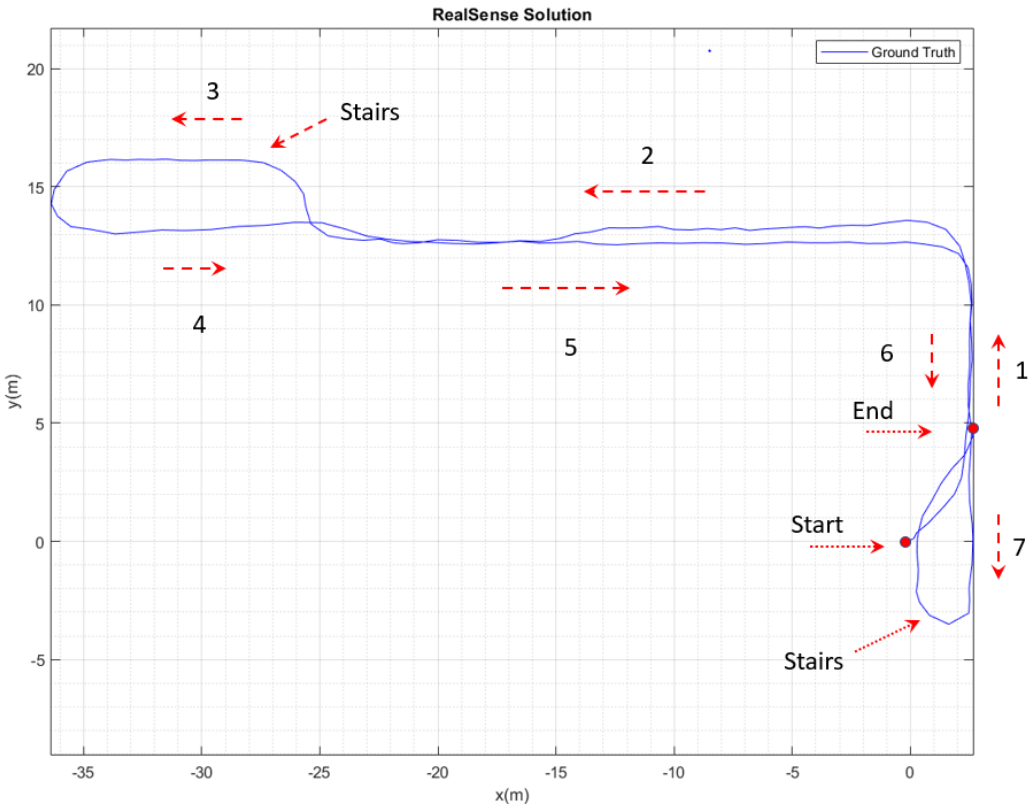

Figure 5.18: Ground Truth Solution for Multi-Floor Dataset

5.4.2 IMU and PDR Model Results

The PDR detected steps is displayed in Figure 5.19. The heading angle and the trajectory generated from the IMU-based PDR system model is shown in Figure 5.20 and Figure 5.21 respectively. It is noticed that the PDR solution drift is larger which is mainly due to the longer travelled distance in this dataset. This is expected for PDR as the longer the integration the largest the drifts in both heading angle and travelled distance.

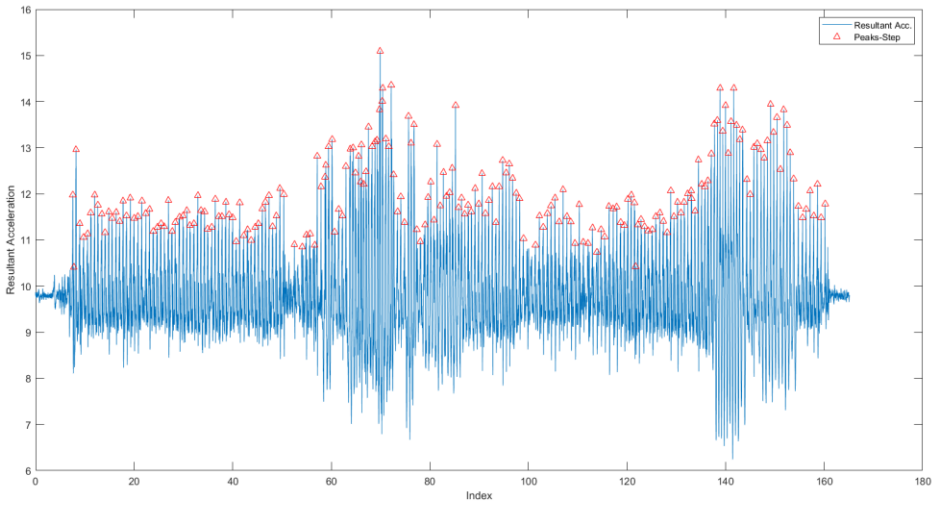

Figure 5.19: Detected Steps for Multi-Floor Dataset

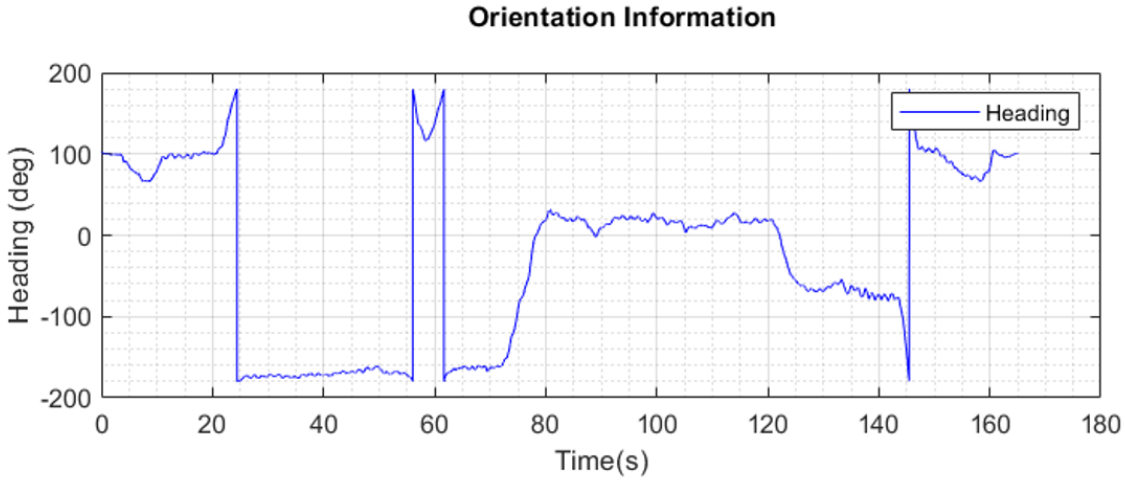

Figure 5.20: Heading Angle for Multi-Floor Dataset

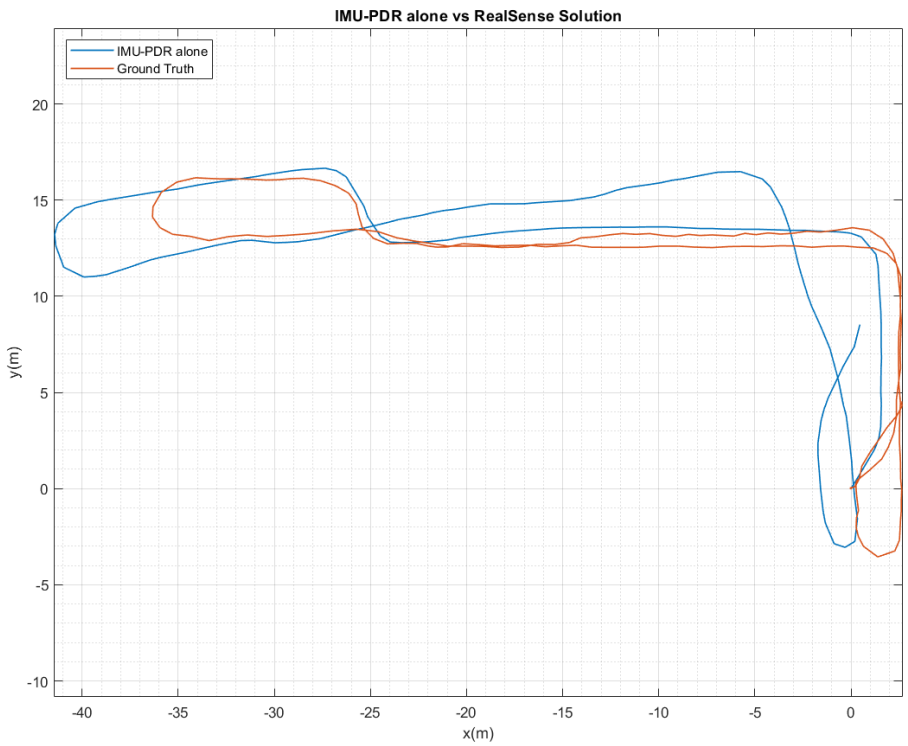

Figure 5.21: PDR vs Ground Truth for Multi-Floor Dataset

5.4.3 Hector SLAM Result

The localization result generated from the Hector SLAM using the LiDAR scans only is plotted in Figure 5.22. In the beginning, Hector SLAM showed poor scan-matching in the corridor during the path segment 1 (see Figure 5.18). It had trouble finding good geometry for almost half of the time, and thus the location was updated for half the portion alone before taking a turn. In the downstairs, for the path segment 3, it drifted a bit, and could not distinguish 180 degrees turn (U-turn), and hence adopted a perpendicular path. After this, Hector SLAM was able to recognize the environment features during the motion for path portion from segment 4 to segment 7. This further confirms the limitations of online SLAM under the absence of reliable system model.

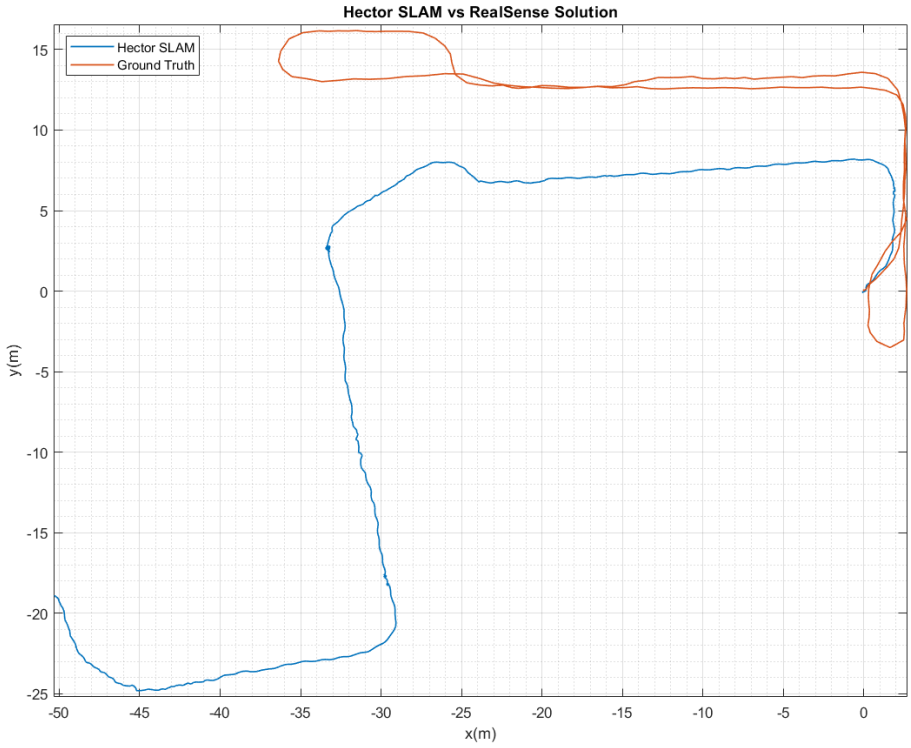

Figure 5.22: Hector SLAM vs Ground Truth for Multi-Floor Dataset

5.4.4 Cartographer SLAM with LiDAR only

Figure 5.23 illustrates the trajectory plot from the Cartographer SLAM using LiDAR scans alone. Compared to the Hector SLAM result presented in the preceding subsection, Cartographer SLAM with LiDAR only showed better accuracy. However, scan-matching shows poor performance in the long hall at the beginning of the experiment for the path segment 1 (from Figure 5.18); that's why the trajectory appears to be a bit shifted as compared to the ground truth. Due to this inability to detect distinguished geometry earlier in the experiment, the overall shape of the trajectory is off by few meters with respect to ground truth solution.

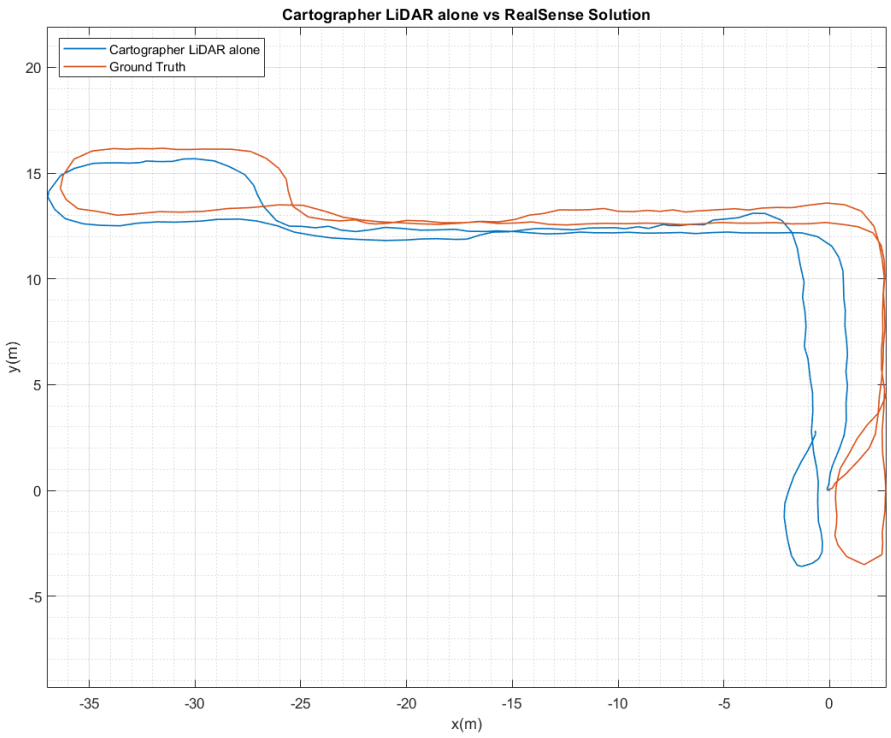

Figure 5.23: Cartographer SLAM vs Ground Truth for Multi-Floor Dataset

5.4.5 Proposed Solution Result

The localization trajectory from the Cartographer SLAM using the PDR-aided solution is displayed in Figure 5.24. The figure depicts the proximity of the PDR-aided system trajectory to the ground truth solution. Using PDR, the Cartographer engine was able to bridge the poor geometry portions in the corridors in path section 1 (see Figure 5.18), where both the Hector SLAM and Cartographer SLAM using LiDAR scans only struggled. The proposed system generated superior performance by showing robust performance during featureless environments as compared to the Hector SLAM and Cartographer SLAM results used with LiDAR only presented in the previous two subsections.

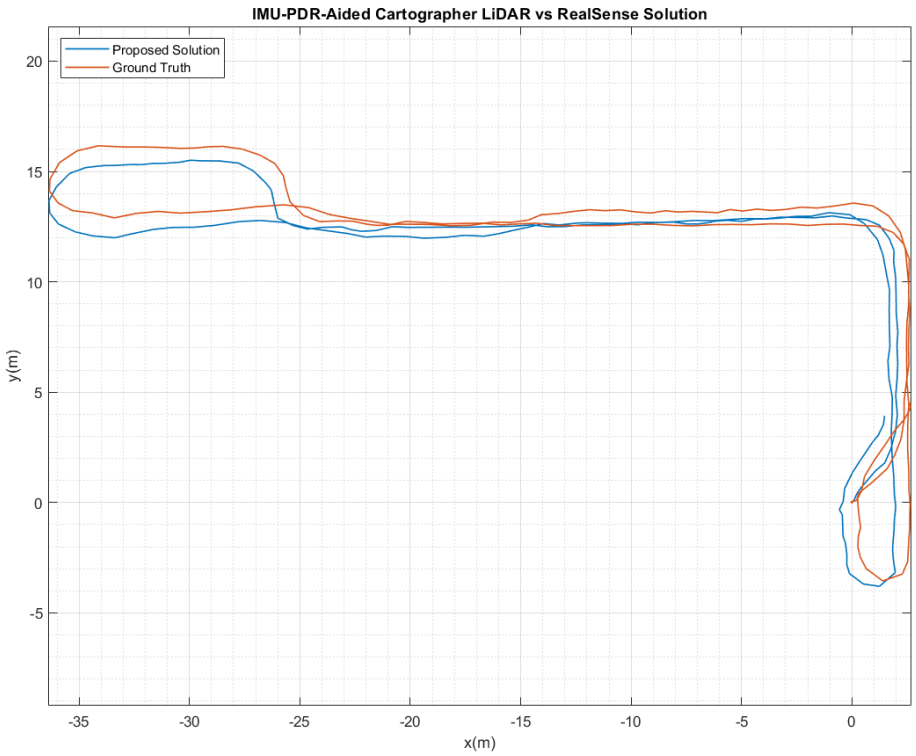

Figure 5.24: Proposed Solution vs Ground Truth for Multi-Floor Dataset

5.5 Quantitative Results Analysis

For some statistical insights, Table 5.2 represents Root Mean Square Error (RMSE) results, whereas the Table 5.3 represents the RMSE error as a percentage of the travelled distance for different implemented approaches.

5.5.1 RMSE Result Calculation

The RMSE is calculated by using the following equation,

$$\text{RMSE} = \sqrt{\sum_{i=1}^{n} \frac{(\hat{x}_i - x_i)^2 + (\hat{y}_i - y_i)^2}{n}} \tag{5.1}$$

Where,

- $(\hat{x}_1,\ \hat{y}_1), (\hat{x}_2,\ \hat{y}_2), \dots, (\hat{x}_n,\ \hat{y}_n)$: Represent the predicted values, which in this case are the 2D Pose position output from the IMU-PDR model, Hector SLAM, Cartographer SLAM using LiDAR only, and IMU-PDR aided Cartographer SLAM results.
- $(x_1,\ y_1), (x_2,\ y_2), \dots, (x_n,\ y_n)$: Represent the observed measurements, which in this case are the ground truth (RealSense 2D Pose position estimates) values.
- n : is the number of observations which is the trajectory pose points.

For example, for an eight shaped dataset, after calculating the sum of the squared difference between the 2D-Pose $(x,\ y)$ position of predicted $(\hat{x}_n,\ \hat{y}_n)$ and observed $(x_n,\ y_n)$ for $n = 244$ values, the rest of the calculation is performed as follows,

$$\text{RMSE\_eight} = \sqrt{\frac{798.9215}{244}} = \sqrt{3.2743} = 1.8095$$

The RMSE values from both the tables clearly show that in all the experiments, the PDR-aided solution yielded better results. The Cartographer engine, with LiDAR only, produced better results when compared against Hector SLAM. This should not be surprising as full SLAM solutions are generally more accurate than online SLAM. For the rectangle-shaped dataset, the PDR-aided system achieved approximately 4.41% improvement compared to Cartographer LiDAR-only result. The PDR-aided solution was able to achieve approximately 78.61% improvement in the RMSE error when compared with Cartographer LiDAR-only solution for the eight-shaped dataset. At the same time,

the improvement for the Multi-floor dataset was approximately 64.33%. Thus, the PDR-aided solution was able to achieve 71.47% of overall average performance for the experiments involving 50m plus distance. In contrast, the overall average performance improvement was 49.12% for all the datasets.

Table 5.2: RMSE Results - Trajectory vs Ground truth Solution

APPROACH	Rectangle Shaped Path RMSE (m)	Eight Shaped Path RMSE (m)	Multi-Floor Path RMSE (m)
IMU-PDR alone	3.1542	3.8682	5.7601
Hector SLAM	13.6360	13.2140	24.6623
Cartographer LiDAR alone	2.2224	8.4589	2.3572
Proposed Solution	**2.1243**	**1.8095**	**0.8407**

Table 5.3: RMSE Error as a Percentage of the Travelled Distance

APPROACH	Rectangle Shaped Path	Eight Shaped Path	Multi-Floor Path
IMU-PDR alone	7.5%	4.4%	4.7%
Hector SLAM	32.5%	15.2%	20.2%
Cartographer LiDAR alone	5.3%	9.7%	1.9%
Proposed Solution	**5.0%**	**2.1%**	**0.7%**

Chapter 6: Conclusions and Future Work

The thesis proposed an IMU-PDR-Aided SLAM solution for body-mounted platforms. The proposed system has been realized on a helmet platform with a 2D LiDAR and IMU mounted on it. Two SLAM engines were tested and compared, Hector SLAM and Google's Cartographer SLAM. A Realtime Synchronization and Data Logging firmware was designed and developed, and sensors were integrated with it. To monitor the operation of the developed firmware and the sensors connected to it, a Remote-Controlled System software with Graphical User Interface was developed. Three datasets were collected in different scenarios. Experimental results demonstrated the superior performance of the proposed PDR-aided solution. Results also showed that the PDR integration helps in bridging featureless environments at which conventional SLAM methods fail if relying only on LiDAR.

6.1 Contributions

The main contributions of this thesis can be summarized as follows:

1) Implementation of a PDR module to be integrated within the SLAM engine.
2) The development of a helmet-mount SLAM prototype using LiDAR/IMU. All firmware and software of the prototype have been developed as well.
3) Using the developed prototype to collect time-synchronized multisensor data supported by a ground truth trajectory from a high-end visual SLAM sensor.
4) Testing the implemented PDR on the data collected by the system prototype using two popular SLAM engines under different testing scenarios.

5) One paper about this thesis has been presented in the IEEE MWSCAS'20 conference:

   - **H. Sadruddin**, A. Mahmoud and M. M. Atia, "Enhancing Body-mounted LiDAR SLAM using IMU-based Pedestrian Dead Reckoning (PDR) Model," in the 63rd IEEE International Midwest Symposium on Circuits and Systems, Springfield, MA, USA, 2020.

6.2 Future Work

The work presented in this thesis opens new frontiers related to the integration of IMU-based PDR with the SLAM engines. A brief list is given below:

1. The thesis work applied the method in post-processing offline mode. An important future work direction is the real-time implementation of the implemented PDR within SLAM engines. To achieve this, a PDR algorithm which can provide a real-time pedestrian odometry information to the SLAM engine needs to be developed. Due to the advancement in the Machine Learning and Deep Learning, a reliable and robust PDR model can be developed. For example, by using the "Back-Propagation" algorithm for training "Feed-Forward Neural Networks" [51] which is widely used in supervised learning can be used to calculate the "Step-length estimate" of a pedestrian. For "Step-detection" phase of PDR, a signal-processing algorithm which could detect the peaks in the resultant accelerometer by employing threshold after passing it through a smoothing filter to remove

the undesired noise could be implemented. The "Navigation-solution update" phase information is already available to us by using the AHRS-EKF in a loosely-couple scheme. Hence, by having the three major steps of PDR in real-time, IMU-PDR information could be made available to the SLAM engine real-time and we could have our proposed solution implemented in an online mode to enhance the localization accuracy of the SLAM engine.

2. The developed PDR-Aided SLAM can be extended to provide a 3D solution using a 3D LiDAR sensor and 10DOF IMU. The 3D LiDARs available in market, have increased scan rates and higher angular accuracy, and are thus capable of capturing more features. Having feature rich readings create high resolution maps, which helps the scan-matching phase of LiDAR SLAM, and can increase the accuracy of the Local SLAM. For example, in our experiments, the 2D LiDAR could read the distance measurements in the 2D plane, horizontally. Hence, it missed the feature information from the floor, ceiling, etc. But, the 3D LiDAR could capture the features from the floor, ceiling, and captures reading in 3D. Thus, a 3D LiDAR can provide more rich information to the SLAM engine to increase the overall accuracy of the SLAM result.
3. The position of IMU mounted on different body parts, can have different accuracy results. Therefore, instead of single IMU, multiple IMUs can be mounted on the body to provide better PDR accuracy. For example, for an "IMU" which when mounted on a foot, has two phases, a "Stance Phase"

and a “Swing Phase”. When a person is walking, one of the feet is stagnant called “Stance-Phase”, whereas the other is in air called “Swing-Phase”. During the “Swing-Phase” the foot is in motion and IMU readings will report rapid changes in the 3D accelerometers and 3D gyroscopes reading. But, the foot on the ground, which is in “Stance-Phase” has very low or near zero readings of accelerometers and gyroscopes. A technique called “Zero-Velocity Update (ZUPT)”, could be employed during the “Stance-Phase” to compensate the drifts of IMU [52]. This technique can detect the steps for the “Step-Detection Phase” of PDR module with good accuracy, and hence can increase the precision of the PDR position result. Adding two IMUs, one on each foot, and using the results from each of the IMUs by sensor fusion can further improve the accuracy of the PDR algorithm. Similarly, when an IMU is attached to the hip of a pedestrian, another technique called “Weinberg-algorithm” [17] can be employed, which better estimates the “Step-Length” of a pedestrian, and thus can help in increasing the PDR model performance.

References

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA, MIT Press, 2005 [Online]. Available: https://mitpress.mit.edu/books/probabilistic-robotics

[2] P. D. Groves, *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems*, 2nd ed. Boston, London: Artech House, 2013 [Online]. Available:https://www.navtechgps.com/principles_of_gnss_inertial_and_multisensor_integrated_navigation_systems_2nd_edition/

[3] S. Zahran, A. Moussa, and N. El-Sheimy, "Enhanced UAV Navigation using Hall-Magnetic and Air-Mass Flow Sensors in Indoor Environment," *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci.*, vol. 4, no. 2/W5, pp. 187–194, 2019, doi: 10.5194/isprs-annals-IV-2-W5-187-2019.

[4] G. Grisetti, C. Stachniss, and W. Burgard, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters," *IEEE Trans. Robot.*, vol. 23, no. 1, pp. 34–46, 2007, doi: 10.1109/TRO.2006.889486.

[5] C. Lu, H. Uchiyama, D. Thomas, A. Shimada, and R. I. Taniguchi, "Indoor positioning system based on chest-mounted IMU," *Sensors (Switzerland)*, vol. 19, no. 2, pp. 1–20, 2019, doi: 10.3390/s19020420.

[6] M. Atia, "Design and simulation of sensor fusion using symbolic engines," *Math. Comput. Model. Dyn. Syst.*, vol. 25, no. 1, pp. 40–62, 2019, doi: 10.1080/13873954.2ii019.1566266.

[7] S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," *9th IEEE Int. Symp. Safety, Secur. Rescue Robot. SSRR 2011*, pp. 155–160, 2011, doi: 10.1109/SSRR.2011.6106777.

[8] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2016, pp. 1–8, doi: 10.1109/ICRA.2016.7487258.

[9] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*, 2nd ed., vol. 49, no. 03. MIT Press, 2011.

[10] A. Cranny, A. Beriain, H. Solar, G. Tartarisco, and G. Tartarisco, "Vital Sign Sensing Technology," in *Systems Design for Remote Healthcare*, Springer, New York, NY, 2014, pp. 55–92.

[11] C. Grubin, "Derivation of the quaternion scheme via the euler axis and angle," *J.*

Spacecr. Rockets, vol. 7, no. 10, pp. 1261–1263, 1970, doi: 10.2514/3.30149.

[12] A. L. Schwab, "Quaternions, Finite Rotation and Euler Parameters," Delft, The Netherlands, 2002 [Online]. Available: http://bicycle.tudelft.nl/schwab/Publications/quaternion.pdf

[13] Jay A. Farrell, *Aided Navigation: GPS with High Rate Sensors*. New York: McGraw Hill, 2008.

[14] T. B. Karamat, M. M. Atia, and A. Noureldin, "An enhanced error model for EKF-based tightly-coupled integration of GPS and land vehicle's motion sensors," *Sensors (Switzerland)*, vol. 15, no. 9, pp. 24269–24296, 2015, doi: 10.3390/s150924269.

[15] "Natural Resources Canada, Magnetic field calculator." [Online]. Available: https://geomag.nrcan.gc.ca/calc/mfcal-en.php. [Accessed: 11-Jul-2020]

[16] M. Atia, "Multi-Sensor Integrated Navigation in Urban and Indoor Environments: GNSS, Inertial, and Range Sensors Integration," in *Positioning and Navigation in Complex Environments*, Hershey, Pennsylvania: IGI Global, 2018 [Online]. Available: https://www.igi-global.com/book/positioning-navigation-complex-environments/181077

[17] H. Weinberg, "Using the ADXL202 in Pedometer and Personal Navigation Applications," Norwood, MA, 2002 [Online]. Available: https://www.analog.com/media/en/technical-documentation/application-notes/513772624AN602.pdf

[18] H. Xing, J. Li, B. Hou, Y. Zhang, and M. Guo, "Pedestrian Stride Length Estimation from IMU Measurements and ANN Based Algorithm," *J. Sensors*, vol. 2017, 2017, doi: 10.1155/2017/6091261.

[19] A. Grunnet-Jepsen, M. Harville, B. Fulkerson, D. Piro, S. Brook, and J. Radford, "Introduction to Intel® RealSense™ Visual SLAM and the T265 Tracking Camera." [Online]. Available: https://dev.intelrealsense.com/docs/intel-realsensetm-visual-slam-and-the-t265-tracking-camera. [Accessed: 11-Jul-2020]

[20] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part I," *IEEE Robot. Autom. Mag.*, vol. 13, no. 2, pp. 99–108, 2006, doi: 10.1109/MRA.2006.1638022.

[21] M. Montemerlo and S. Thrun, *FastSLAM - A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics*, 1st ed., vol. 27, no. 1. Springer-Verlag Berlin Heidelberg, 2007.

[22] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM," *IEEE Intell. Transp. Syst. Mag.*, vol. 2, no. 4, pp. 31–43, 2010,

doi: 10.1109/MITS.2010.939925.

[23] "Algorithm walkthrough for tuning — Cartographer ROS documentation." [Online]. Available: https://google-cartographer-ros.readthedocs.io/en/latest/algo_walkthrough.html. [Accessed: 11-Jul-2020]

[24] MathWorks, "Estimate Robot Pose with Scan Matching." [Online]. Available: https://www.mathworks.com/help/nav/ug/estimate-robot-pose-with-scan-matching.html

[25] Y. He, B. Liang, J. Yang, and J. He, "An Iterative Closest Points Algorithm for Registration of 3D Laser Scanner Point Clouds with Geometric Features," *Sensors (Basel)*, vol. 17, no. 8, p. 1862, 2017, doi: 10.3390/s17081862.

[26] P. Biber, "The Normal Distributions Transform: A New Approach to Laser Scan Matching," *IEEE Int. Conf. Intell. Robot. Syst.*, vol. 3, pp. 2743–2748, 2003.

[27] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer-VerlagBerlin, Heidelberg, 2010.

[28] S. J. Julier and J. K. Uhlmann, "Using covariance intersection for SLAM," *Rob. Auton. Syst.*, vol. 55, no. 1, pp. 3–20, 2007, doi: 10.1016/j.robot.2006.06.011.

[29] S. Agarwal and A. Mierle, "Ceres Solver." [Online]. Available: http://ceres-solver.org/. [Accessed: 11-Jul-2020]

[30] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent, "Efficient sparse pose adjustment for 2D mapping," *IEEE/RSJ 2010 Int. Conf. Intell. Robot. Syst. IROS 2010 - Conf. Proc.*, pp. 22–29, 2010, doi: 10.1109/IROS.2010.5649043.

[31] J. Clausen, "Branch and bound algorithms-principles and examples," Department of Computer Science, University of Copenhagen, Denmark, 1999 [Online]. Available: https://imada.sdu.dk/~jbj/heuristikker/TSPtext.pdf

[32] "Cartographer ROS Integration — Cartographer ROS documentation." [Online]. Available: https://google-cartographer-ros.readthedocs.io/en/latest/. [Accessed: 11-Jul-2020]

[33] "MTw Awinda." [Online]. Available: https://www.xsens.com/products/mtw-awinda. [Accessed: 11-Jul-2020]

[34] H. J. Luinge, F. Dijkstra, and G. Bellusci, "Reduction of Link Requirements of an Inertial Measurement Unit (IMU/AP) for a Strapdown Inertial System (SDI)," EU Patent Application: EP2645061A22013.

[35] M. Paulich, M. Schepers, N. Rudigkeit, and G. Bellusci, "Xsens MTw : Miniature Wireless Inertial Motion Tracker for Highly Accurate 3D Kinematic

Applications," Xsens Technologies B.V, Enschede, Netherlands, Rep. MW0404P.A,May,2018[Online].Available:https://www.xsens.com/hubfs/3446270/Downloads/Manuals/MTwAwinda_WhitePaper.pdf

[36] Xsens Technologies B.V, "MTw Awinda User Manual," Xsens Technologies B.V, Enschede, Netherlands, Rep. MW0502P, May, 2018 [Online]. Available:https://www.xsens.com/hubfs/Downloads/Manuals/MTw_Awinda_User_Manual.pdf

[37] "RPLIDAR-A1 360° Laser Range Scanner - Domestic Laser Range Scanner|SLAMTEC." [Online].Available:https://www.slamtec.com/en/Lidar/A1. [Accessed: 11-Jul-2020]

[38] M. H. Rashid, *Power Electronics: Circuits, Devices & Applications*, 4th ed. Upper Saddle River, NJ: Pearson, 2014.

[39] S. Slamtec, "RPLiDAR A1 - Introduction and Datasheet," 2009 [Online]. Available: https://www.generationrobots.com/media/rplidar-a1m8-360-degree-laser-scanner-development-kit-datasheet-1.pdf

[40] "Tracking camera T265 – Intel RealSense Depth and Tracking Cameras." [Online]. Available: https://www.intelrealsense.com/tracking-camera-t265/. [Accessed: 11-Jul-2020]

[41] "Intel® Movidius™ Vision Processing Units (VPUs)." [Online]. Available: https://www.intel.com/content/www/us/en/products/processors/movidius-vpu.html. [Accessed: 11-Jul-2020]

[42] Intel, "Intel ® RealSense ™ T265 Tracking Camera - Datasheet," Santa Clara, CA, USA, Rep. 572522-004, March, 2019 [Online]. Available:https://www.intel.com/content/dam/support/us/en/documents/emerging-technologies/intel-realsense-technology/IntelRealSenseTrackingT265Datasheet.pdf

[43] "Qt | Cross-platform software development for embedded & desktop." [Online]. Available: https://www.qt.io/. [Accessed: 11-Jul-2020]

[44] "OpenStreetMap." [Online]. Available: https://www.openstreetmap.org/. [Accessed: 11-Jul-2020]

[45] "ROS.org | Powering the world's robots." [Online]. Available: https://www.ros.org/. [Accessed: 11-Jul-2020]

[46] "Intro to ROS — ROS Tutorials 0.5.1 documentation." [Online]. Available: http://www.clearpathrobotics.com/assets/guides/kinetic/ros/Intro to the Robot Operating System.html. [Accessed: 11-Jul-2020]

[47] "Documentation - ROS Wiki." [Online]. Available: http://wiki.ros.org/.

[Accessed: 11-Jul-2020]

[48] “msg - ROS Wiki.” [Online]. Available: http://wiki.ros.org/msg. [Accessed: 11-Jul-2020]

[49] “sensor_msgs Msg/Srv Documentation.” [Online]. Available: http://docs.ros.org/melodic/api/sensor_msgs/html/index-msg.html. [Accessed: 11-Jul-2020]

[50] “nav_msgs/Odometry Documentation.” [Online]. Available: http://docs.ros.org/melodic/api/nav_msgs/html/msg/Odometry.html. [Accessed: 11-Jul-2020]

[51] C. Bishop, *Pattern Recognition and Machine Learning*, 1st ed. Springer-Verlag New York, 2006.

[52] X. Yun, E. R. Bachmann, H. Moore, and J. Calusdian, “Self-contained Position Tracking of Human Movement Using Small Inertial/Magnetic Sensor Modules,” in *IEEE International Conference on Robotics and Automation*, 2007, pp. 2526–2533, doi: 10.1109/ROBOT.2007.363845.