Berkeley
UNIVERSITY OF CALIFORNIA

Imperial College London

07/22/2019

Accuracy Estimation of 2D SLAM Using Sensor Fusion of LiDAR and INS

A Study in A Project for CalUnmanned:

Autonomous Concrete Crack Detection with UAVs

by Jan Xu

University of California, Berkeley SID: ***3034407224***

Imperial College London CID: ***01076529***

Research Supervisor: Prof Raja Sengupta

Department of Civil and Environmental Engineering

University of California, Berkeley

Abstract

As interest in autonomous robot navigation grows, Self-Localization and Mapping (SLAM) using low-cost range and inertial sensors is becoming ever-increasingly popular within the scientific community. Many issues regarding accuracy, reliability, scalability and adaptability of the algorithms used are of question in many existing systems, which this paper seeks to investigate. Using inexpensive inertial sensors inside an Inertial Measurement Unit (IMU) and a 2D Light Detection and Ranging (LiDAR) device, we are establishing a sensor fusion system built based on an extended Kalman filter to estimate position and heading of the data collection package. The data was collected inside a structured experimental environment built to verify the results and estimate accuracy of the methods used. A full pose estimation relies on traditional navigation equations integrated with translation and rotation estimation based on LiDAR scan matching approaches, of which two have been tested in this study; a traditional point-to-point scan method and a custom-designed feature-to-feature approach based on extracted line features and keypoints such as corners and endpoints. Several test runs where implemented and presented, with the results achieving adequate standards with a relatively low error margin at best. A discussion on the reliability and adaptability of the algorithms used has also been included in terms of data collection and future works.

Acknowledgements

I want to take the opportunity in this section to thank my supervisor at UC Berkeley, Professor Raja Sengupta, for his invaluable insight, support and endorsement on this project. I would also like to extend my utmost gratitude to Xin Peng, a current PhD student at the CalUnmanned laboratory, for offering me endless feedback and ideas that helped me along the way. In addition, I would like to thank Jacob Gallego at the Student Machine Shop in Etcheverry Hall at UC Berkeley, as well as my friend Davide Asnaghi for helping me with providing physical resources for the experimental setup used in this study.

Table of contents

1 Introduction

Next generation's societies and industries will encounter an enormous uptake in autonomous technology. We are already seeing a large development in cutting-edge tech gadgets able to travel from origin to destination with no human intervention, including unmanned aerial vehicles, self-driving cars, submarines, delivery bots, indoor service robots and lunar rovers. The advancement in automated robot solutions is already able to replace or complement various functions in agriculture, building information modelling, civil and military transportation, delivery and logistics, and emergency and disaster relief amongst others. These systems necessitate reliable and accurate navigation systems for them to safely traverse through and collect information about complex environments.

Traditionally, navigation systems have been based on Inertial Measurement Units (IMU) and/or Global Navigation Satellite Systems (GNSS). An IMU is a device that includes accelerometers, gyroscopes, barometers and magnetometers, which can be used to determine the relative change in position and orientation of the system; this is also called *odometry*. A GNSS leverages satellite information to establish an absolute position of the system. However, the drawback of the former is the integration drift over time from the small incremental motions, and the latter suffers from low accuracy and occasional signal dropout in cluttered environments. One method of overcoming these issues is the fusion of both types of data; the IMU provides high-frequency odometry data for local dead reckoning, and the GNSS supplements with bounded, accurate absolute position [12], [14].

However, in urban or indoor environments where satellite signals are unattainable, this method proves futile. Alternative solutions have been proposed and implemented in systems employing so-called SLAM algorithms, using laser rangefinders [1], [19], [24], cameras [7], [27] and sonar [18]. Simultaneous Localization and Mapping (SLAM) refers to the coupled problem of mapping an unknown environment around a mobile instrument, whilst simultaneously determining the position of the instrument itself. These solutions are cheap, simple to implement and require only low computational resources. SLAM is fundamental in the design, operation and robustness of autonomous vehicles. One factor contributing to the recent boom in autonomous vehicle technology is the continuous decline of sensor complexity and cost; laser rangefinders in particular are currently very affordable and user-friendly, which is perfect for small start-ups and academic research teams to further develop, test and reinforce SLAM technology.

The goal of this study is to implement SLAM algorithms by fusing odometry and pose data from an IMU with range data from a Light Detection and Ranging (LiDAR) device. If implemented successfully, we will obtain geometrical estimates of a preordained trajectory and the surrounding environment. Ultimately, we would like to compare the obtained trajectory and map estimate with the ground truth values and make estimations of the accuracy attained from the SLAM result. To this end, a real-life experimental setup has been constructed such that the sensor data is collected under conditions reflecting ground truth as close as possible, using the available physical resources. With this data, we can represent and manipulate the state-space using simplified transition models, such as the Kalman Filter [16] or Extended Kalman Filter, in order to fuse information from the IMU and LiDAR to make accurate state estimations.

Ideally, the SLAM algorithms should be sufficiently optimized to be run on-line so that the vehicle can control itself based on the data it reads in real-time. However, for the purposes of this exercise where the main weight has been placed on evaluating the aptitude and accuracy of SLAM algorithms to ascertain the state of the vehicle and the surrounding features, we will not assess the computational performance of the algorithm used. The implication of this is that the data can be collected, stored off-line and processed separately from the robot operating system.

This paper will, however, attempt to make a qualitative evaluation of the suitability of the integrated LiDAR and INS system to adequately perform 2D SLAM, and raise several limitations in the algorithms proposed and tested as well as inaccuracies accredited to hardware. Since the sensors used are relatively cheap, the data they read will inevitably be noisy and prone to drift. Other error sources include issues with data association, calibration and bias estimation.

The report is divided in the following sections:

- Section 1, this section, introduces the topic, objectives and scope of this study;
- Section 2 highlights a number of related studies in the relevant field;
- Section 3 sets out mathematical notation and preliminaries required to understand the details of the techniques used;
- Section 4 explains the characteristics and setup procedures of the sensors used for the study;
- Section 5 walks through the system architecture and the governing equations of the trajectory estimation and environment mapping;
- Section 6 outlines the experimental setup and the procedure with which the data was collected;
- Section 7 presents the SLAM results and estimates the accuracy of the algorithms used;
- Section 8 discusses limitations and possible improvements of the SLAM algorithms in terms of the experiment;
- Section 9 concludes this report.

2 Related work

Many previous studies have been implemented regarding SLAM using various combinations of LiDAR and inertial sensors. It has been used widely for localization of unmanned aerial vehicles and ground vehicles in both simulated and real environments. Simulated experiments to ascertain the viability of navigation with LiDAR and IMU include [33] where the authors perform a simulated state estimation in a corridor environment and highlight the effects of sensor noises and biases, and [8] and [34] where an additional camera sensor has been employed to further reinforce accuracy of the pose estimation.

Studies in real-life experimental environments include [30] where the authors propose a custom version of the classic Iterative Closest Point (ICP) scan matching algorithm for localization, where a Principal Component Analysis was used to detect line features for mapping purposes. Kumar et al. [22] uses an integrated LiDAR and IMU system for indoor navigation and pipeline classification, applying the Polar Scan Matching (PSM) method presented in [9] with two orthogonally attached LiDAR sensors to establish a 3D position. A low-computational 2D SLAM approach with a LiDAR sensor integrated with a 3D inertial navigation system was put forward in [19], where each LiDAR scan is matched to the global map instead of the preceding scan, allowing for accurate localization in unknown environments with versatile application scenarios. The authors of [35] take advantage of the geometry of a known environment to accurately estimate the position of a ground vehicle with LiDAR sensors in urban environments where GNSS based positioning may be unavailable. Finally, [21] presents a novel technique to match laser range-finder scans based on cross-correlation, which is associated with high accuracy and low computational resources compared to traditional scan matching methods.

For this particular study, two scan matching algorithms have been employed for pose estimation based on the LiDAR sensor data. One is the traditional Iterative Closest Point scan matching algorithm, first introduced in [2] and [5]. The second algorithm was inspired by the works in [24] and [28], the former of which presents a novel line extraction algorithm that can be exploited for robot localization.

3 Preliminaries

The description of state-space representation and state transition models relies heavily on complex mathematical equations, and no two studies share the exact same notations and conventions. Therefore, before going further with detailing the SLAM methodology, it is imperative to outline the notation used throughout this report and present preliminary information that will be very helpful to digest the information in upcoming sections.

3.1 Mathematical notation

The following notation of scalar, vector and matrix quantities will be used:

Quantity	Font	Example
Scalar	Regular (non-bold) font	$T_k, r_{i,j}, \sigma_{lid,s}$
Vector	Bold, lowercase font	$\boldsymbol{z}_k, \boldsymbol{u}_{k-1}, \boldsymbol{r}_i$
Matrix	Bold, uppercase font	$\boldsymbol{P}_k, \boldsymbol{R}_b^n$

Table 3.1 Mathematical notations for scalar, vector and matrix quantities used in this report.

The $n \times n$ identity matrix is denoted as $\boldsymbol{I}_n$ and the $n \times m$ zero matrix is denoted as $\boldsymbol{0}_{n,m}$. The following subscript convention will be used:

Subscript	Description	Example
i	First index in an ordered series, not necessarily time-dependent	LiDAR scan i
j	Second index in an ordered series of the first ordered series, not time-dependent	Scan point j in LiDAR scan i
k	Time index in which the system state takes place, time-dependent	State $\boldsymbol{x}_k$ at time step k
f	Indicates the coordinate frame in which the quantity is expressed in	$\boldsymbol{x}_{k,f}$

Table 3.2 Subscript notations used in this report.

Any subscript notation not covered in the table above will be detailed in later sections.

3.2 Coordinate frames

Whereas we often want the solution in the 'global' frame in navigation problems, sensors will operate in their own 'local' coordinate frame. It is crucial to familiarize oneself with the relevant coordinate frames before discussing the accelerometer and gyroscope data. The four coordinate frames that play a key role in navigation, as described in [20], are the following:

- **Body frame**, b: the coordinate frame associated to the motions of the IMU. Its origin coincides with the center of the accelerometer triad and its axes are aligned to the casing. All inertial measurements by the IMU are resolved in this frame.
- **Navigation frame**, n : the coordinate frame associated with the local geographical region in which the navigation occurs. For the purposes of our application, this frame is defined stationary with respect to the Earth's surface. More specifically for the real-life experimentation, its origin is defined as the starting point of the experimental environment and its axes follow a locally defined North × East direction.
- **Inertial frame**, i : the stationary coordinate frame with the origin at the center of the Earth and its axis aligned with respect to the stars. The IMU measures linear acceleration and angular velocity in terms of the inertial frame.
- **Earth frame**, e: the coordinate frame that coincides with the inertial frame, but instead of staying fixed, it rotates with respect to the Earth's rotation. In other words, its axes are fixed to certain coordinate points expressed in longitude and latitude, with one axis coinciding with the North Pole, one axis with the equator and the prime meridian and one axis with the equator and the meridian 90° east of the prime meridian.

An illustration of the n, i and e frames can be seen in Figure 3.1, with superscripts indicating the coordinate frame of a quantity.

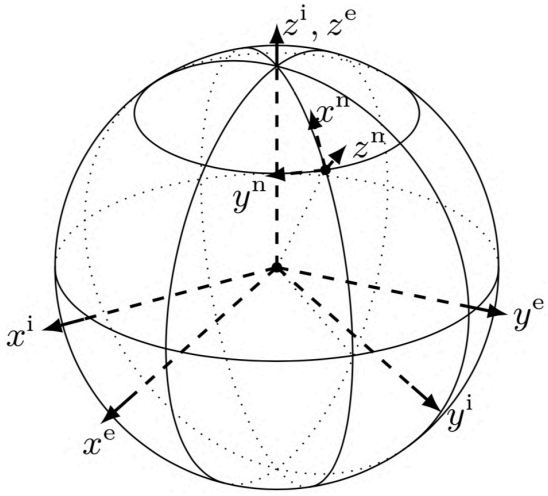


Figure 3.1 Illustration of the navigation frame n, the inertial frame i and the Earth frame e. Taken from [20].

Transformation of a vector quantity from one coordinate frame to another is most commonly done using a rotation matrix, usually denoted as $\boldsymbol{R}_b^n$ for instance. The subscript b indicates the coordinate frame that the original vector is expressed in, and the superscript n denotes the coordinate frame that the vector is transformed to. For instance, if one wants to transform a vector quantity $\boldsymbol{x}_{k,b}$ originally expressed in the body frame to the navigation frame, $\boldsymbol{x}_{k,n}$, the following equation is used:

$$\boldsymbol{x}_{k,n} = \boldsymbol{R}_b^n \boldsymbol{x}_{k,b}$$

The reverse can be done by transposing $\boldsymbol{R}_b^n$:

$$\boldsymbol{x}_{k,b} = (\boldsymbol{R}_b^n)^T \boldsymbol{x}_{k,n} = \boldsymbol{R}_n^b \boldsymbol{x}_{k,n}$$

3.3 Angle representation

Rotation in 3D state-space can be represented in various forms, such as Euler angles, direction cosine matrices and unit quaternions. Different representations are more suitable for expressing an angular orientation or rotational motion than others at different situations. This section will devote itself to equip the reader with the different ways of expressing angles and rotations.

3.3.1 *Euler angles*

As the easiest method to visualize orientation and rotation of a rigid body in state-space, Euler angles are constituted by a set of three *elemental rotations* about each axis in Euclidean space;

- *roll,* ϕ: rotation about the x-axis;
- *pitch*, θ: rotation about the y-axis;
- *yaw*, ψ: rotation about the z-axis.

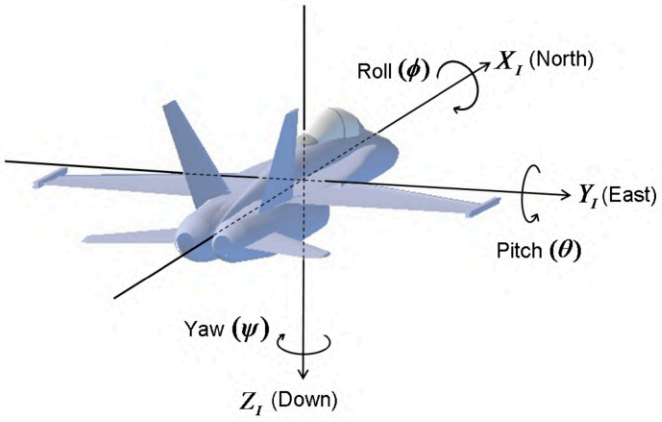


Figure 3.2 Illustration of Euler angles.

As seen in Figure 3.2, Euler angles have a very intuitive visualization and can also be linearized very effectively, which will be important for the state update process. It is also very easy to convert Euler angles to/from other angle representations which facilitates data manipulation. However, it suffers from a phenomenon called *gimbal lock* which occurs when one gimbal rotates such that it ends up parallel with another gimbal, "locking" the system in a configuration with one fewer degree of freedom. This can severely destabilize the system, which is why many navigation systems rely on other ways to describe orientations.

3.3.2 *Direction cosine matrix*

A direction cosine matrices (DCM) is a matrix representation of a space-state of size $n \times n$ in $\mathbb{R}^n$. Its vectors consist of the unit vectors that describe the current coordinate frame with respect to an origin coordinate frame and is very useful to transform vector quantities from one coordinate frame to another. It also allows for multiple successive rotations by matrix multiplying the rotation matrices with each other. The rotation matrix $\boldsymbol{R}_b^n$ described in Section 3.2 provides an example of a DCM representation, with its unit vectors of the body frame in terms of the navigation frame as the columns of the rotation matrix.

A 3D rotation about the x-axis, i.e. a roll motion ϕ, can be expressed as a DCM as such:

$$\boldsymbol{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$

Similarly, for a pitch motion θ and a yaw motion ψ:

$$\boldsymbol{R}_y = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

$$\boldsymbol{R}_z = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A generalized expression of a rotation about all axes can thus be described as:

$$\boldsymbol{R} = \boldsymbol{R}_z \boldsymbol{R}_y \boldsymbol{R}_x =$$
$$\begin{bmatrix} \cos(\theta)\cos(\psi) & -\cos(\phi)\sin(\psi) + \sin(\phi)\sin(\theta)\cos(\psi) & \sin(\phi)\sin(\psi) + \cos(\phi)\sin(\theta)\cos(\psi) \\ \cos(\theta)\sin(\psi) & \cos(\phi)\sin(\psi) + \sin(\phi)\sin(\theta)\sin(\psi) & -\sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi) \\ -\sin(\theta) & \sin(\phi)\cos(\theta) & \cos(\phi)\cos(\theta) \end{bmatrix}$$

Conversely, given a rotation matrix expressed by

$$\boldsymbol{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

one can find the Euler angles with these formulae:

$$\phi = atan2(r_{32}, r_{33})$$
$$\theta = atan2\left(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}\right)$$
$$\psi = atan2(r_{21}, r_{11})$$

3.3.3 *Unit quaternions*

Unit quaternions are very convenient tool to represent 3D orientations and rotations and are widely used in computer graphics, robotics and flight dynamics. Unlike Euler angles, they avoid the problem of gimbal lock and they are more

numerically stable and easier to compose than both DCMs and Euler angles. A quaternion q is a four-tuple number system described by the real parameters $\{q_w, q_x, q_y, q_z\}$ as such:

$$q = q_w + q_x\boldsymbol{i} + q_y\boldsymbol{j} + q_z\boldsymbol{k}$$

The quaternion units $\{\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}\}$ are an extension of the complex number unit i, that have the property that

$$\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1$$

and a unit quaternion has a norm of 1, i.e.

$$|q|^2 = q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1$$

The algebra of the quaternion is beyond the scope of this report, but a thorough explanation of its mechanics can be found in [15]. What is important is the simplicity of describing state dynamics of a rotating state using only four parameters, that can easily be converted to and from other angle representations. The IMU also directly outputs its own orientation in unit quaternions, thus converting this to a DCM gives a rotation matrix that transforms any vector quantity measured by the IMU from its body frame to the navigation frame. For a unit quaternion $q = q_w + q_x\boldsymbol{i} + q_y\boldsymbol{j} + q_z\boldsymbol{k}$, the equivalent DCM can be expressed as:

$$\boldsymbol{R} = (q_w^2 - \boldsymbol{q_v}^T\boldsymbol{q_v})\boldsymbol{I}_3 + 2\boldsymbol{q_v}\boldsymbol{q_v}^T + 2q_w[\boldsymbol{q_v}]_\times$$

where

$$\boldsymbol{q_v} = \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix}$$

and $[-]_\times$ is the skew-symmetric operator, such that, for a 3×1 vector,

$$[\boldsymbol{q_v}]_\times = \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix}_\times = \begin{bmatrix} 0 & -q_z & q_y \\ q_z & 0 & -q_x \\ -q_y & q_x & 0 \end{bmatrix}$$

The conversion from quaternions to Euler angles can be described with the following formulae:

$$\phi = atan2\left(2\left(q_w q_x + q_y q_z\right), 1 - 2\left(q_x^2 + q_y^2\right)\right)$$
$$\theta = arcsin\left(2\left(q_w q_y - q_x q_z\right)\right)$$
$$\psi = atan2\left(2\left(q_w q_z + q_x q_y\right), 1 - 2\left(q_y^2 + q_z^2\right)\right)$$

Conversely, Euler angles can be converted to quaternions:

$$q_w = \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right)$$
$$q_x = \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) - \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right)$$
$$q_y = \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right)$$
$$q_z = \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) - \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right)$$

4 Sensor characteristics

No real-life sensor is perfect; that is to say, the data that is outputted from the sensors will inevitably be biased and noisy. Therefore, in order to implement SLAM algorithms and estimate the state of the system and to map the environment accurately, it is of vital importance to appreciate the imperfections of the sensor data for data manipulation and bias correction. Extensive calibration of the sensors and mathematical models that take into account the uncertainties are tools that will certainly help us along the way.

This section of the report will describe the sensors used for this study in detail, which consist of an accelerometer, gyroscope and magnetometer embedded with the IMU, as well as a laser rangefinder that produces a scan representation of the immediate surroundings of the system. This section will also shed some light on the challenges faced when dealing with imperfect measurements, and what we can do to improve them.

4.1 IMU

The IMU is a multi-sensor device that has been used extensively in every kind of control system where odometry plays a part. It has a high output rate and can therefore be used to produce quick state estimations using the characteristics of the state dynamics. However, since the sensors have biases, the integrative and additive nature of the processed IMU data is very prone to so-called *sensor drift*. For instance, when taking the double integral of the linear acceleration measured by the IMU to estimate the position, small biases in the output data will almost certainly yield unbounded position errors that accumulate to the magnitude of several meters even after a few seconds [4].

Because of this, the user must take great care in calibrating the sensors properly, as well as taking

into account any further bias. Even so, the IMU data will still not be sufficiently accurate in order to rely on dead reckoning solely and must therefore be used in conjunction with additional sensors such as a LiDAR through sensor fusion. In the next subsections, each sensor in the IMU will be described in detail, followed by a discussion in IMU calibration and bias estimation.

4.1.1 *Accelerometer*

A Micro-Electro-Mechanical System (MEMS) accelerometer inside an IMU consists of a moving beam structure mounted to springs that respond to any applied acceleration. Illustrated in Figure 4.1, this acceleration changes the capacitance between the fixed and moving beam fingers, from which an acceleration can be measured. The accelerometer measures the specific force $\boldsymbol{f}_b$ along each axis of the body frame. Since the sensor operates in its own instantaneous rest frame, placing an accelerometer at rest on Earth's surface will yield an acceleration reading approximately 1g upwards. Hence, the measurements in the local inertial frame from an accelerometer must be converted to a linear acceleration $\boldsymbol{a}_n$ with respect to the navigation frame for our purposes. This can be approximated by resolving the specific force vector from the body frame to the navigation frame through a rotational transformation matrix, and removing the gravity component:

$$\boldsymbol{a}_n \approx \boldsymbol{R}_b^n \boldsymbol{f}_b - \boldsymbol{g}_n$$

where $\boldsymbol{g}_n = [0 \quad 0 \quad 9.80665]^T \, m/s^2$. There are other global forces at play, however, such as the Coriolis acceleration and centrifugal acceleration due to the Earth's rotation, that may influence the accelerometer readings further [20], which will be dealt with in the bias error compensation (see Section 4.1.5).

4.1.2 *Gyroscope*

The gyroscope is a device that measures the orientation and angular velocity of the body frame. As shown in Figure 4.2, a traditional gyroscopic sensor consists of a stationary spinning wheel due to conservation of angular momentum, and thus detects any rotational movement of the three gimbals attached to its exterior. An IMU outputs angular velocity readings in terms of the three axes of rotation; i.e. roll velocity, pitch velocity and yaw velocity, as well as its orientation in quaternion

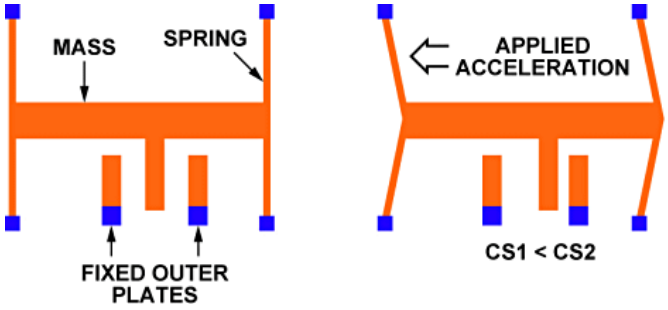


Figure 4.1 The structure of a MEMS accelerometer. Taken from [17].

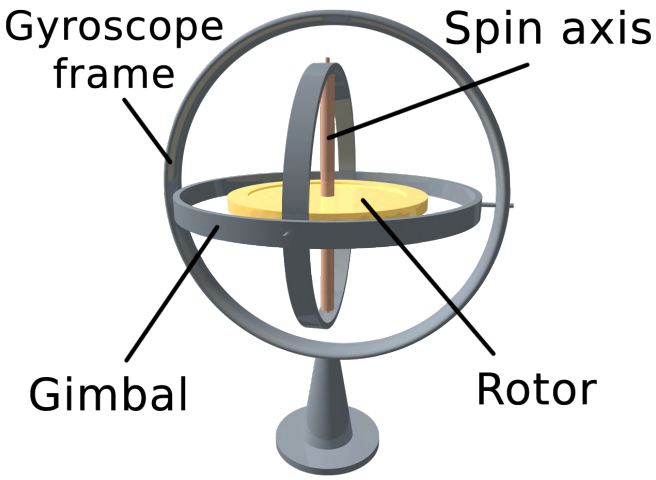


Figure 4.2 The structure of a traditional gyroscope.

form. Inexactness of the angular velocity data stems from the relatively small Earth rate due to Earth's rotation with respect to the stars, and the non-zero transport rate in case of a non-stationary navigation frame (not applicable for our study) [20]. Both of these quantities have been ignored, and for our purposes the angular velocity of the body frame with respect to the inertial frame has been approximated to be equal to the angular velocity of the body frame with respect to the navigation frame.

4.1.3 *Magnetometer*

The most conventional magnetometer is a compass, which measures the direction of a magnetic field at a given location. Although not explicitly outputted by the IMU as sensor data, this device assists in orientation readings, in particular for yaw position. This sensor is highly sensitive to any external magnetic fields apart from the ambient one from the Earth's core. Thus, it is extremely important to calibrate this device in order to produce accurate positioning, especially since the experimental setup will itself produce magnetic fields of its own due to electrical devices that power and control the sensors.

4.1.4 *IMU calibration*

Ideally, MEMS sensors like those embedded within an IMU should share the same 3D orthogonality of each sensor axis and have the correct scaling factors to convert the digital signals read by the sensors to real physical quantities [32]. However, low-cost sensors are usually affected by inaccurate scaling, axis misalignments and biases, such that the physical quantities produced by them may contain misrepresentations of the physical state. IMU calibration aims to identify the quantities required for accurate transformation from the digital realm to the physical.

Calibration of the IMU data is done at the setup phase of the sensors, using a drone configuration application called QGroundControl which is intended to be used for any Micro Air Vehicle Link (MAVLink) enabled control. Documentation can be found online at [11]. It provides a user interface on which the user can calibrate the sensors within the IMU device by placing it stationary on a flat surface on all its sides and rotating the device around all its axes. Careful attention of any external magnetic interference should be taken, especially for the setup of the magnetometer. Therefore, the calibration process should be performed outdoors with the experimental equipment setup to compensate for the magnetic interference induced by the electronic equipment itself.

4.1.5 *Bias error estimation*

As seen in Figure 4.3, even after calibration the IMU outputs measurements that are offset by slight biases for a given physical input. In many cases where low-cost IMUs are implemented, the bias is generally composed by two components that describe the behavior of bias over time: repeatability and stability [29]. The former refers to the initial bias being different each time during start-up, due to changes in physical properties of the IMU and initial conditions of the signal processing from digital to physical. The latter refers to changes in the bias as the IMU is run, which may be related to temperature and mechanical stress variations on the system.

The initial bias can be estimated by averaging across initial IMU measurements right at the power-up of the device, where the system is at rest in position before movement begins and averaging across these readings. In the case of output data for linear acceleration, for example, which also picks up gravitational acceleration in its readings, the related initial bias could be estimated by transforming a number of initial data points with the orientation state of the IMU from the body frame to the navigation frame. Taking the mean of the transformed data will then estimate both bias and gravity into a complete gravity vector $\boldsymbol{g}$ which is then subtracted from linear acceleration inputs in the sensor fusion stage.

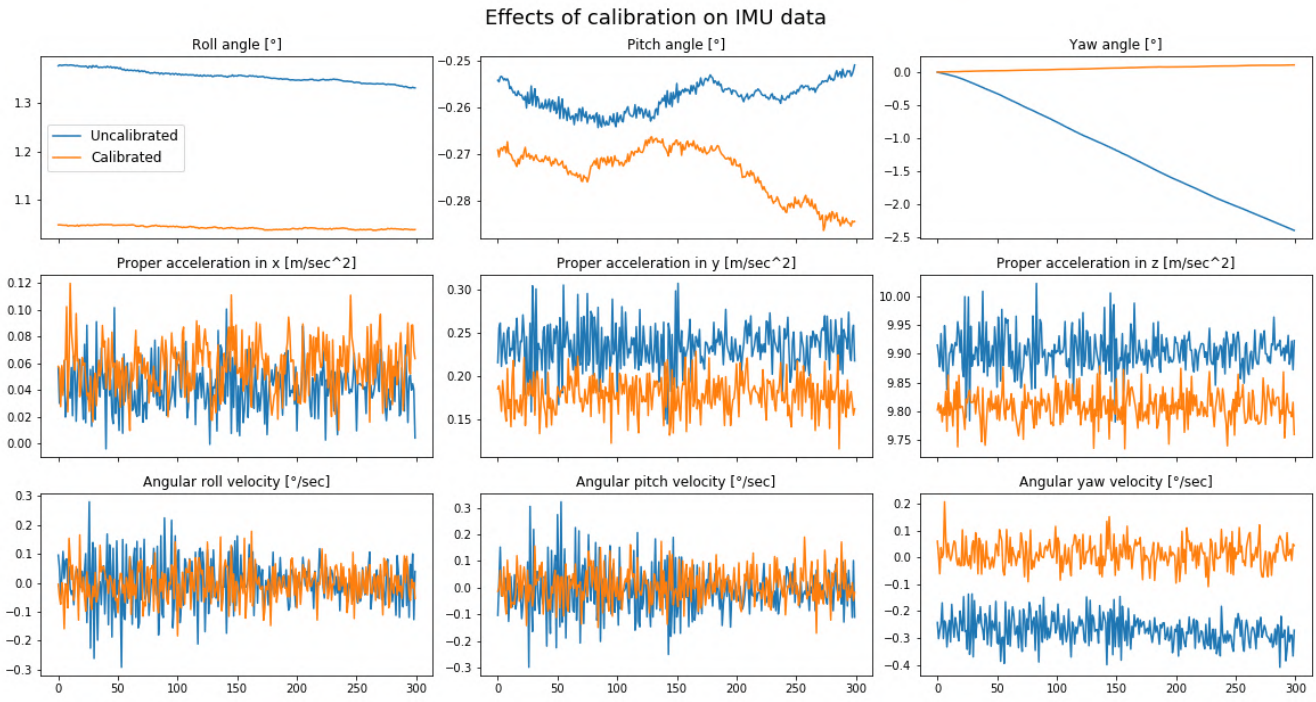


Figure 4.3 Effects of calibration on IMU data, based on 300 initial inertial readings from the IMU. Blue graphs: uncalibrated IMU data. Orange graphs: calibrated IMU data.

The running bias is estimated on-the-go during the sensor fusion process by including the sensor biases as part of the system state. These bias variables pertaining to linear acceleration and angular velocity can be modelled as a random walk process $\delta \boldsymbol{u}_k$ with additive white, zero-mean bias process noise $\boldsymbol{\chi}_k$ that has a covariance noise of $\boldsymbol{\Xi}_k$, such that

$$\delta \boldsymbol{u}_k = \delta \boldsymbol{u}_{k-1} + \boldsymbol{\chi}_k \in \mathbb{R}^6$$

The bias error can then be used to compensate the physical quantities outputted by the IMU, $\boldsymbol{u}_k$, to obtain the 'error-compensated' IMU readings $\tilde{\boldsymbol{u}}_k$, which are used in the sensor fusion model:

$$\tilde{\boldsymbol{u}}_k = \boldsymbol{u}_k - \delta \boldsymbol{u}_k$$

4.2 LiDAR

LiDAR stands for *Light Detection and Ranging* and is a type of optical rangefinder device that performs laser scanning on an arbitrary scan plane of the immediate surrounding of the system. It provides point clouds of high resolution and precision in a wide range of directions at a high sampling rate. This can then be processed using various algorithms to estimate the trajectory taken by the moving device whilst mapping the environment. LiDAR technology has been proven to provide more accurate maps and models than vision-based SLAM models [23] and since its inception, cheaper and more advanced LiDAR sensors are appearing on the market, making them ever-increasingly appealing and tractable in airborne and ground navigation and mapping applications.

The working principle of a LiDAR is to shoot short light pulses in each radial direction within its scan range from its scan origin, which travels towards the nearest distant object in its paths. It then bounces back and is registered by the LiDAR, which then measures the time it took for the light pulse to travel back and forth. By multiplying the travel time with the speed of light, the distance to the closest object in the given direction can be inferred. Every scan consists of a collection of scan points within the scan range, which comprise a so-called *point cloud*, denoted $\mathcal{P}_k$ for the kth time step with N_k scan points. The point cloud can be seen as a mapping of the immediate environment of the sensor (Figure 4.4) and consists of two arrays: one angle array $\{\phi_{k,i}\}_{i=1}^{N_k}$, which contains the angles from the LiDAR centerline of each scan point, and one range array $\{r_{k,i}\}_{i=1}^{N_k}$ which contains the radial distance from the LiDAR centerpoint to the nearest object that obstructs the laser beam in that scan direction. Multiple LiDAR devices have various scan distance and angular ranges. In our study, the LiDAR sensor used is constrained to scan points between 120 and -120 degrees from the LiDAR centerline. Therefore, the rear 120 degrees of the LiDAR form a blind zone which has to be taken into account.

To find the Cartesian coordinates of a scan point with range distance $r_{k,i}$ in the scan direction of $\phi_{k,i}$ from the LiDAR centerline, denoted $\boldsymbol{p}_{k,i}$, we can use the following formula:

$$\boldsymbol{p}_{k,i} = \begin{bmatrix} x_{k,i} \\ y_{k,i} \end{bmatrix} = \begin{bmatrix} r_{k,i} \cos(\phi_{k,i}) \\ r_{k,i} \sin(\phi_{k,i}) \end{bmatrix}$$

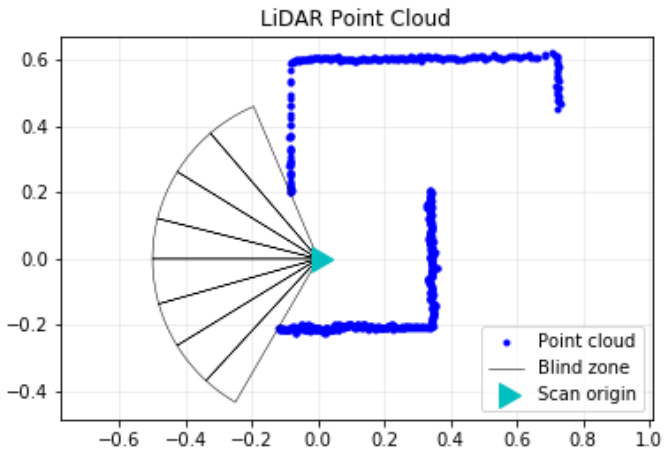


Figure 4.4 Example of a LiDAR point cloud.

4.2.1 *Scan matching*

It is possible to estimate the trajectory of the LiDAR system by associating scan points, or features inferred from the scan points, between the point clouds of multiple scans. The goal of laser scan matching is to find the translation and rotation of the current point cloud with respect to a reference scan such that the best overlap is achieved (Figure 4.5). Scan matching algorithms can perform both local and global scan matching; the former associates consecutive scan points with each other after pose initialization, and the latter aligns the current scan with respect to a map in a global coordinate frame. Multiple algorithms for this search exist in the literature, most of which can be categorized in three different approaches:

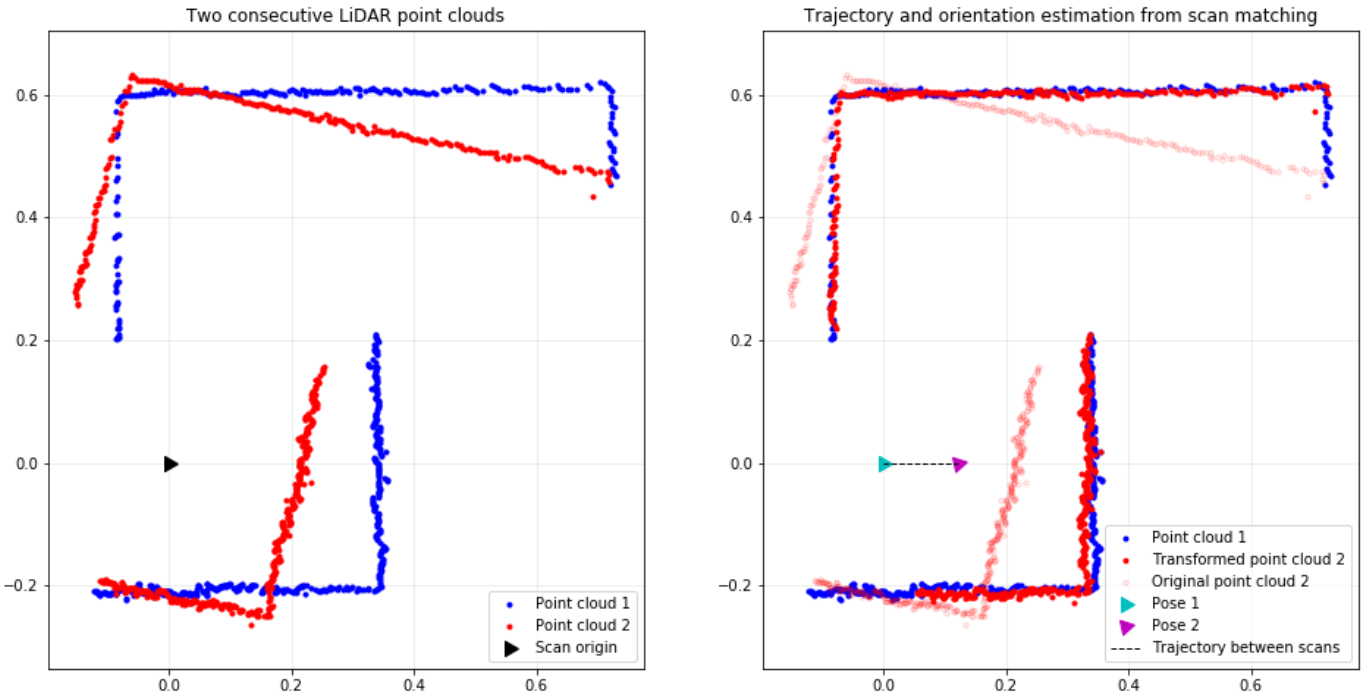


Figure 4.5 The working principle of LiDAR scan matching. Note the change in orientation of the pose, in addition to translation.

a) *Point-to-point scan matching:*
   i. Iterative Closest Point (ICP) [2][5], where the points with the smallest Euclidean distance between each other in two consecutive point clouds are associated;
   ii. Iterative Matching Range Point (IMRP) [26], where corresponding points are selected by matching each point in the scan to the center of the coordinate system of the reference scan;
   iii. Iterative Dual Correspondence (IDC) [26], which essentially combines the two previously mentioned approaches by using ICP for translation estimation and IMRP for orientation estimation.

b) *Point-to-feature scan matching:*
   i. Algorithm proposed in [6], which associates scan points to features such as lines as part of a pre-defined map;
   ii. Normal Distribution Transform (NDT) [3], in which features are represented by Gaussian distributions with their mean and variance estimated from scan points falling into cell grids.

c) *Feature-to-feature scan matching:*
   i. Feature matching based on extracted line segments, such as that proposed in [24];
   ii. Feature matching based on other corners and range extrema such as that proposed in [25].

Different scan matching algorithms will naturally yield different results in the accuracy of the trajectory and environment reconstruction and is also highly dependent on whether the environment is structured (i.e. contains extractable features) or complex. However, an extensive comparison of the performances between the available approaches is beyond the scope of this project. Instead, the experimentation performed in this study enables future researchers to explore different scan matching approaches in the SLAM problem according to the experimental setup produced for the purpose of this project.

5 Navigation system and SLAM

The SLAM procedure combines odometrical pose estimates using the high frequent and reliable in short-term IMU data with translation and rotation estimates through laser scan matching using the accurate and robust in long-term LiDAR point clouds. The bulk of the SLAM procedure is constituted by a Kalman filter; it functions as an optimization model which, given the inputs from the two sensors and the uncertainties attributed to the physical model and the sensor measurements, outputs the best estimate of the state. In other words, the state estimate is the one that minimizes the error or loss function defined by the Kalman filter.

This section will explain each step of the SLAM procedure implemented in detail. An overview of the system architecture in its entirety will be presented first. Next, the theoretical concept of Kalman filtering will be introduced, setting the framework within which the sensor fusion takes place. A description of the state-space representation follows this, listing the system state and covariance variables and proposing the linearized space-state model for state transition. Afterwards, the two algorithms for LiDAR scan matching studied in this research will be set forth. Lastly, the mapping procedure of the environment, based on LiDAR point clouds, will be outlined.

5.1 System architecture

The system architecture used in this study is outlined in Figure 5.1. The blue and green blocks, "State Prediction" and "State Correction", refer to the sensor fusion stages in the Kalman filter which will be described in the upcoming sections. Rectangular boxes with sharp corners represent physical quantities such as vectors, matrices or other types of data structures, whereas boxes with rounded corners describe various procedures.

5.2 Kalman filter

The Kalman filter is one of the most important algorithms throughout all disciplines of engineering. Initially introduced in 1960 by Rudolf E. Kalman [16], it is most widely used in time series analysis and dynamic problems, most relevant for this study in robot planning and control but also for signal processing, econometrics and computer vision. A Kalman filter uses the dynamic model of a system to update and estimate the state of the system based on known control inputs and sequential measurements from multiple sensors, such as IMU and LiDAR. The resulting state estimate is better, that is, more likely to be accurate, than estimating based solely on one type of sensor observation. The visible result of Kalman filtering is generally a smoothened version of the noisy unprocessed sensor data. The Kalman filter is computationally very efficient and can be run in real time along with the system.

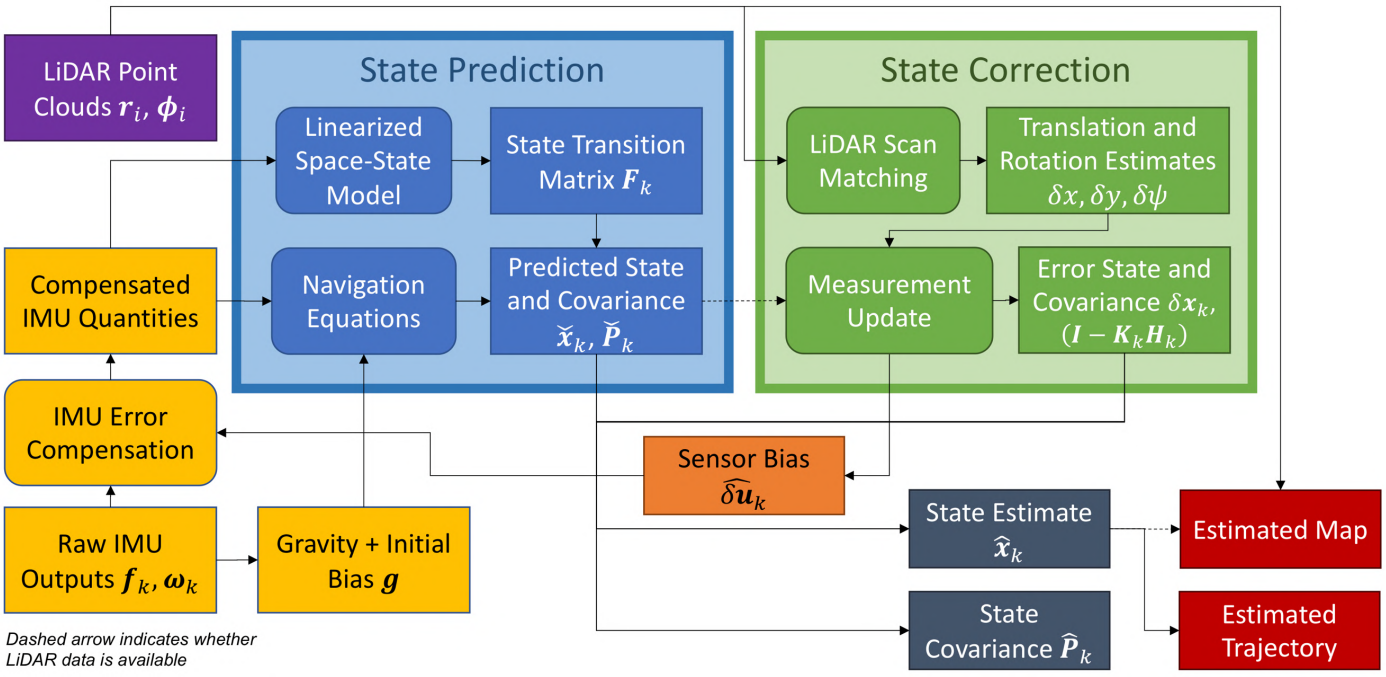


Figure 5.1 Diagram of system architecture of the SLAM model used in this study. Boxes with sharp corners represent physical quantities, whereas boxes with rounded corners describe procedures.

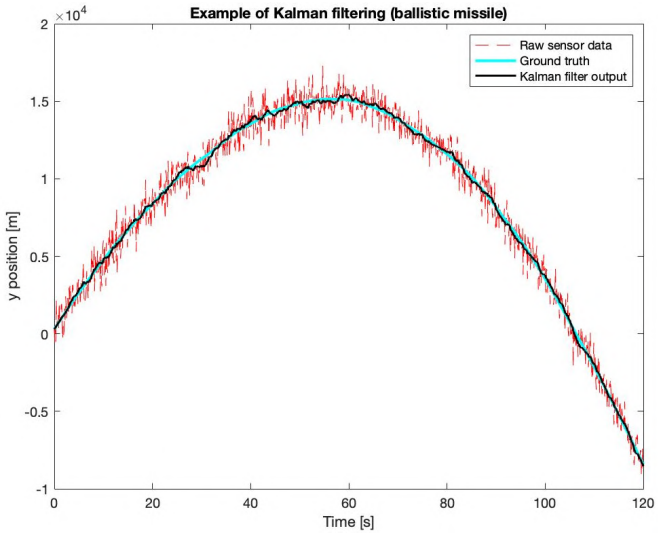


Figure 5.2 Simulated experiment of state estimation of a ballistic missile shot at an angle, to demonstrate the smoothening effects of Kalman filtering.

The way a Kalman filter finds the best estimate of the true system state $\boldsymbol{x}_k$ based on system dynamics and sensor measurements is through an optimal weight matrix, the *Kalman gain matrix* $\boldsymbol{K}_k$. In order to compute the matrix, a general notion of the uncertainty associated with the dynamics and measurements is required. The uncertainty is represented by the covariance of the system state $\boldsymbol{P}_k$, as well as the covariances associated with the process noise (from system dynamics) and observation noise (from sensors). In reality, the state changes continuously over time; in a Kalman filter procedure, we discretize the state evolution in the time domain instead and apply the state and covariance calculation for each time step recursively. Hence, the Kalman filter only requires information from its prior state along with the new observations in order to perform the calculations, which explains its efficiency and tractability in real-time applications.

The true state of the system $\boldsymbol{x}_k$ at time step k is based on the underlying dynamical model:

$$\boldsymbol{x}_k = \boldsymbol{F}_k \boldsymbol{x}_{k-1} + \boldsymbol{B}_k \boldsymbol{u}_k + \boldsymbol{w}_k$$

where $\boldsymbol{F}_k$ denotes the state transition matrix which represents the physical model that describes the state evolution with time, such as the laws of motion, $\boldsymbol{B}_k$ stands for the control-input model applied to the control input vector $\boldsymbol{u}_k$, such as acceleration or gyration, and $\boldsymbol{w}_k$ is process noise assumed to follow a zero-mean Gaussian distribution with covariance $\boldsymbol{Q}_k$: $\boldsymbol{w}_k \sim \mathcal{N}(0, \boldsymbol{Q}_k)$.

For a sensor measurement $\boldsymbol{y}_k$ at time step k of the true state $\boldsymbol{x}_k$, the following representation can be made:

$$\boldsymbol{y}_k = \boldsymbol{H}_k \boldsymbol{x}_k + \boldsymbol{v}_k$$

where $\boldsymbol{H}_k$ is the observation model that transforms the true state to the observed measurement and $\boldsymbol{v}_k$ is the observation noise, also assumed to be a zero-mean Gaussian with covariance $\boldsymbol{R}_k$: $\boldsymbol{v}_k \sim \mathcal{N}(0, \boldsymbol{R}_k)$. All the noise vectors are assumed to be mutually independent. Using this representative dynamical system model, the procedure for the Kalman filter calculation can be divided into two steps:

1) *State prediction:* The predicted (a priori) state $\breve{\boldsymbol{x}}_k$ at time step k is estimated based on the previous state estimate $\hat{\boldsymbol{x}}_{k-1}$ at time step $k-1$ and the state transition model $\boldsymbol{F}_k$:

$$\breve{\boldsymbol{x}}_k = \boldsymbol{F}_k \hat{\boldsymbol{x}}_{k-1} + \boldsymbol{B}_k \boldsymbol{u}_k$$

$\boldsymbol{B}_k$ and $\boldsymbol{u}_k$, associated to the control inputs, which are related to the linear acceleration quantities measured by the IMU in our experiment. The predicted (a priori) error covariance $\breve{\boldsymbol{P}}_k$ can be calculated as

$$\breve{\boldsymbol{P}}_k = \boldsymbol{F}_k \hat{\boldsymbol{P}}_{k-1} \boldsymbol{F}_k^T + \boldsymbol{L}_k \boldsymbol{Q}_k \boldsymbol{L}_k^T$$

where $\boldsymbol{L}_k$ is the noise gain matrix.

2) *State correction*: If sensor data becomes available such as LiDAR, then the observation will be used to "correct" the predicted state. This is done by computing the optimal Kalman gain matrix $\boldsymbol{K}_k$ as

$$\boldsymbol{K}_k = \breve{\boldsymbol{P}}_k \boldsymbol{H}_k^T \left(\boldsymbol{H}_k \breve{\boldsymbol{P}}_k \boldsymbol{H}_k^T + \boldsymbol{R}_k \right)^{-1}$$

which is then used to compute the error state, or correction:

$$\delta \boldsymbol{x}_k = \boldsymbol{K}_k (\boldsymbol{y}_k - \boldsymbol{H}_k \breve{\boldsymbol{x}}_k)$$

The corrected (a posteriori) state $\hat{\boldsymbol{x}}_k$, along with the corrected (a posteriori) covariance $\hat{\boldsymbol{P}}_k$, are then calculated using the formulae:

$$\hat{\boldsymbol{x}}_k = \breve{\boldsymbol{x}}_k + \delta \boldsymbol{x}_k$$
$$\hat{\boldsymbol{P}}_k = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k) \breve{\boldsymbol{P}}_k$$

It can be theoretically testified that the Kalman filter is the best linear unbiased estimator, or "BLUE" [31]. This means that for linear dynamical systems, the Kalman filter is guaranteed to output the most accurate state estimation given that the covariances are exactly known. However, most real-life systems are inherently non-linear, and therefore an extension to the linear estimator for non-linear problems is often required. In addition, covariances are rarely known precisely and thus several iterations of parameter tuning is required to optimize the model such that the error is minimized.

5.2.1 *Extended Kalman filter*

Consider the non-linear dynamical model

$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{w}_k$$
$$\boldsymbol{y}_k = h(\boldsymbol{x}_k) + \boldsymbol{v}_k$$

where f and h are the motion model function and observation model function, respectively. Both are non-linear and differentiable functions, and for this particular problem, the standard Kalman filter is unable to directly make state estimates represented by non-linear functions of either measurements or control inputs. One approach around this issue is by adopting an extension of the Kalman filter, the *extended* Kalman filter (EKF) which uses analytical methods to approximate a linearized version of the problem. The main approach makes use of a Taylor series expansion of a non-linear function to approximate a linear function. By taking a first-order Taylor approximation of the function about the nominal state $\hat{\boldsymbol{x}}_{k-1}$ and control input $\boldsymbol{u}_k$ of the motion model function f, we can compute the state transition matrix $\boldsymbol{F}_k$ as the Jacobian matrix

$$\boldsymbol{F}_k = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k-1}, \boldsymbol{u}_k}$$

and similarly, by making a Taylor approximation of the observation model function h about the predicted state $\breve{\boldsymbol{x}}_k$, the observation matrix $\boldsymbol{H}_k$ is the following Jacobian matrix:

$$\boldsymbol{H}_k = \left.\frac{\partial h}{\partial \boldsymbol{x}}\right|_{\breve{\boldsymbol{x}}_k}$$

This linearized model can then be implemented in the standard Kalman filter equations.

5.3 State-space representation

Define $\boldsymbol{x}_k$ as the system (navigation) state vector and $\boldsymbol{u}_k$ as the control (inertial measurement) input vector

$$\boldsymbol{x}_k = \begin{bmatrix} \boldsymbol{p}_k \\ \boldsymbol{v}_k \\ \boldsymbol{q}_k \end{bmatrix} \in \mathbb{R}^{10}$$

$$\boldsymbol{u}_k = \begin{bmatrix} \boldsymbol{f}_k \\ \boldsymbol{\omega}_k \end{bmatrix} \in \mathbb{R}^{6}$$

- $\boldsymbol{p}_k \in \mathbb{R}^3$ $[m]$ denotes position of system;
- $\boldsymbol{v}_k \in \mathbb{R}^3$ $[m/s]$ denotes velocity of system;
- $\boldsymbol{q}_k \in \mathbb{R}^4$ $[-]$ denotes attitude of system in quaternion representation (see Section 3.3);
- $\boldsymbol{f}_k \in \mathbb{R}^3$ $[m/s^2]$ denotes the specific force from the accelerometer in IMU;
- $\boldsymbol{\omega}_k \in \mathbb{R}^3$ $[rad/s]$ denotes the angular velocity from gyroscope and magnetometer in IMU in Euler angle representation (see Section 3.3).

The navigation equations, given by the non-linear dynamical equation $\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)$, can be expressed as

$$\boldsymbol{p}_k = \boldsymbol{p}_{k-1} + \Delta t \boldsymbol{v}_{k-1} + \frac{\Delta t^2}{2}(\boldsymbol{R}_b^n(\boldsymbol{q}_{k-1})\boldsymbol{f}_k - \boldsymbol{g})$$

$$\boldsymbol{v}_k = \boldsymbol{v}_{k-1} + \Delta t(\boldsymbol{R}_b^n(\boldsymbol{q}_{k-1})\boldsymbol{f}_k - \boldsymbol{g})$$

$$\boldsymbol{q}_k = \Omega(q(\Delta t \boldsymbol{\omega}_k))\boldsymbol{q}_{k-1}$$

- $\Delta t \in \mathbb{R}^1$ $[s]$ is the sampling period of data;
- $\boldsymbol{R}_b^n(\boldsymbol{q}) \in \mathbb{R}^{3\times3}$ $[-]$ is the rotation matrix of the unit quaternion $\boldsymbol{q} = [q_w \quad \boldsymbol{q}_v]^T$ with $\boldsymbol{q}_v = [q_x \quad q_y \quad q_z]^T$;
- $\boldsymbol{g} \in \mathbb{R}^3$ $[m/s^2]$ is the initial gravity and bias vector;
- $\Omega(\boldsymbol{q}) \in \mathbb{R}^{4\times4}$ $[-]$ is the matrix representation of the quaternion $\boldsymbol{q}$, such that

$$\Omega(\boldsymbol{q}) = \Omega\left(\begin{bmatrix} q_w \\ \boldsymbol{q}_v \end{bmatrix}\right) = q_w \boldsymbol{I}_4 + \begin{bmatrix} 0 & -\boldsymbol{q}_v^T \\ \boldsymbol{q}_v & -[\boldsymbol{q}_v]_\times \end{bmatrix}$$

  where $[\boldsymbol{q}_v]_\times$ is the skew-symmetric operator of $\boldsymbol{q}_v$ (see Section 3.3.3);
- $q(\boldsymbol{\theta}) \in \mathbb{R}^4$ $[-]$ is the Euler-to-quaternion transformation function of Euler angles $\boldsymbol{\theta} \in \{\phi, \theta, \psi\}$, as described in Section 3.3.3.

Having introduced the system variables and dynamics, we need a convention to describe the error dynamics that propagate in the navigation equations. By denoting the nominal state, i.e. the estimated state, as $\hat{\boldsymbol{x}}_k$, and the true state as $\boldsymbol{x}_k$, the error state $\delta\boldsymbol{x}_k$ is simply the difference:

$$\delta\boldsymbol{x}_k = \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k = \begin{bmatrix}\delta\boldsymbol{p}_k \\ \delta\boldsymbol{v}_k \\ \delta\boldsymbol{\theta}_k\end{bmatrix} \in \mathbb{R}^9$$

where $\delta\boldsymbol{\theta}_k$ is the attitude perturbation vector expressed in a small Euler angle sequence that rotates the system's attitude from $\hat{\boldsymbol{q}}_k$ to $\boldsymbol{q}_k$. The error dynamics are thus described as

$$\boldsymbol{z}_k = \boldsymbol{F}_k\boldsymbol{z}_{k-1} + \boldsymbol{L}_k\boldsymbol{n}_k$$

where $\boldsymbol{z}_k$ and $\boldsymbol{n}_k$ are the full error state vector and full noise vector, respectively:

$$\boldsymbol{z}_k = \begin{bmatrix}\delta\boldsymbol{x}_k \\ \delta\boldsymbol{u}_k\end{bmatrix} \in \mathbb{R}^{15}$$

$$\boldsymbol{n}_k = \begin{bmatrix}\boldsymbol{w}_k \\ \boldsymbol{\chi}_k\end{bmatrix} \in \mathbb{R}^{12}$$

The state transition matrix, $\boldsymbol{F}_k$, the noise gain matrix, $\boldsymbol{L}_k$, and the observation model matrix $\boldsymbol{H}_k = \boldsymbol{H}$ which remains constant are defined as follows:

$$\boldsymbol{F}_k = \begin{bmatrix} \boldsymbol{I}_3 & \Delta t\boldsymbol{I}_3 & \boldsymbol{0}_{3,3} & \boldsymbol{0}_{3,3} & \boldsymbol{0}_{3,3} \\ \boldsymbol{0}_{3,3} & \boldsymbol{I}_3 & \Delta t[\boldsymbol{R}_b^n(\boldsymbol{q}_k)\boldsymbol{f}_k]_\times & \Delta t\boldsymbol{R}_b^n(\boldsymbol{q}_k) & \boldsymbol{0}_{3,3} \\ \boldsymbol{0}_{3,3} & \boldsymbol{0}_{3,3} & \boldsymbol{I}_3 & \boldsymbol{0}_{3,3} & -\Delta t\boldsymbol{R}_b^n(\boldsymbol{q}_k) \\ \boldsymbol{0}_{3,3} & \boldsymbol{0}_{3,3} & \boldsymbol{0}_{3,3} & \boldsymbol{I}_3 & \boldsymbol{0}_{3,3} \\ \boldsymbol{0}_{3,3} & \boldsymbol{0}_{3,3} & \boldsymbol{0}_{3,3} & \boldsymbol{0}_{3,3} & \boldsymbol{I}_3 \end{bmatrix}$$

$$\boldsymbol{L}_k = \begin{bmatrix} \boldsymbol{0}_{3,3} & \boldsymbol{0}_{3,3} & \boldsymbol{0}_{3,3} & \boldsymbol{0}_{3,3} \\ \Delta t\boldsymbol{R}_b^n(\boldsymbol{q}_k) & \boldsymbol{0}_{3,3} & \boldsymbol{0}_{3,3} & \boldsymbol{0}_{3,3} \\ \boldsymbol{0}_{3,3} & \Delta t\boldsymbol{R}_b^n(\boldsymbol{q}_k) & \boldsymbol{0}_{3,3} & \boldsymbol{0}_{3,3} \\ \boldsymbol{0}_{3,3} & \boldsymbol{0}_{3,3} & \boldsymbol{I}_3 & \boldsymbol{0}_{3,3} \\ \boldsymbol{0}_{3,3} & \boldsymbol{0}_{3,3} & \boldsymbol{0}_{3,3} & \boldsymbol{I}_3 \end{bmatrix}$$

$$\boldsymbol{H}_k = [\boldsymbol{I}_3 \quad \boldsymbol{0}_{3,3} \quad \boldsymbol{0}_{3,3} \quad \boldsymbol{0}_{3,3} \quad \boldsymbol{0}_{3,3}]$$

5.3.1 *Bias compensation*

The running bias $\delta\boldsymbol{u}_k$, introduced in Section 4.1.5 and portrayed as the sole orange box in the system architecture diagram displayed in Figure 5.1, is estimated along with the system state in the full error state vector $\boldsymbol{z}_k$. It is important that this estimated bias error is subtracted from every IMU reading during the main filter loop.

5.4 LiDAR scan matching

For this study, two algorithms for LiDAR scan matching have been further investigated for the use in the SLAM problem; ICP, a classic point-to-point approach, and a novel feature-to-feature scan matching based algorithm inspired by the works in [13] and [24]. Both approaches are presented in this section.

5.4.1 *Iterative Closest Point (ICP)*

ICP is a point-to-point algorithm that has been widely used and studied in literatures since it was first introduced in [2]. It seeks the optimal rigid transformation between two consecutive scans locally, which is the one that minimizes the least squares criterion between corresponding points. The output of the procedure is then the optimal rotation matrix and translation vector that aligns the two registered LiDAR point clouds. Given the current point cloud $\mathcal{P}_k$ consisting of N_k number of scan points, i.e. $\mathcal{P}_k = \{\boldsymbol{p}_{k,i}\}_{i=1}^{N_k}$, and the previous point cloud $\mathcal{P}_{k-1}$ with N_{k-1} scan points, $\mathcal{P}_{k-1} = \{\boldsymbol{p}_{k-1,i}\}_{i=1}^{N_{k-1}}$, with each scan point $p_{k,i} \in \boldsymbol{p}_{k,i}, \forall i \in \{1 \dots N_k\}$ being represented by 2D Cartesian coordinates, the general procedure is as follows:

1. *Correspondence search:* for each scan point in $\mathcal{P}_k$, choose and match the scan point in $\mathcal{P}_{k-1}$ that has the closest Euclidean distance to the point. The index of the corresponding point in $\mathcal{P}_{k-1}$ to the scan point $p_{k,i}$ in $\mathcal{P}_k$ is assigned to $c(i)$. This can be done with a k-nearest neighbor code implementation.
2. *Rigid transformation estimation:* find the rotation matrix $\boldsymbol{R}_k^{k-1}$ and translation vector $\boldsymbol{t}_k^{k-1}$ that minimizes the least square error according to the objective function:

$$\min_{\boldsymbol{R}_k^{k-1}, \boldsymbol{t}_k^{k-1}, \{c(i)\}_{i=1}^{N_{k-1}}} \sum_{i=1}^{N_k} \left\| \left(\boldsymbol{R}_k^{k-1} p_{k,i} + \boldsymbol{t}_k^{k-1}\right) - p_{k-1,c(i)} \right\|_2^2$$

3. Transform the scan points in $\mathcal{P}_k$ with the obtained transformation arrays.
4. Iterate steps 1-3 until the solution has converged.

The resulting rotation matrix and translation vectors represent then the optimal transformation between the two sets of scan points. Despite being the workhorse of scan matching algorithms, ICP

still suffers from issues such as being trapped in local minima. This occurs frequently when outliers are present or for large rotations. Modifications in the code have been done to remove outliers during correspondence search. Weights can also be applied to each matched correspondence pair; however, this has not been implemented in this study.

5.4.2 *Feature-based scan matching approach*

Due to the structured environment of the experimental design, it is appropriate to also trial a scan matching approach based on line detection and keypoint identification. Lines are detected based on clustered sets in the point cloud that resembles a linear structure, whereas keypoints are unique objects consisting of corners and endpoints in the line environment that are tagged to a position in the global frame. The line extraction algorithm is largely inspired by [13], the keypoint identification process was developed by the author and the subsequent pose and orientation estimation based on the identified features where drawn from [24].

A line segment can be found by minimizing the orthogonal distance of each scan point from the line, by denoting it as a linear equation in 2D Euclidean space of the form $ax + by + c = 0$. The line fitting procedure consists thus of finding the line parameters $\{a, b, c\}$ that minimize the sum of squares of distances from the subset of points with coordinates (x_p, y_p) that form the line such that

$$\min_{a,b,c} \sum_{subset\ in\ \mathcal{P}_k} \left(\frac{|ax_p + by_p + c|}{\sqrt{a^2 + b^2}} \right)^2$$

which can be done using polynomial curve fitting of degree 1. Given a point cloud $\mathcal{P}_k$, the line extraction process works as follows:

1. *Seed-segment detection:* Fit a seed-segment consisting of a small number of successive scan points, starting with the first scan point. If the seed is valid (i.e. if the sum of squares of residuals to fitted line seed), proceed to next step. Otherwise, pick the next scan point as new starting point and repeat.
2. *Region growing:* Successively add adjacent scan points to detected seed-segment that are sufficiently close to the seed and refit the line to obtain updated parameters. When no valid points can be found at either end of line, proceed to next step.
3. *Line segment post-processing:* After all line segments have been extracted, further post-processing is necessary to ensure that no line segments overlap, have a too small distance and cover too few scan points. Short lines will either be discarded or merged with an adjacent line if their geometries are similar.
4. *Endpoint generation:* The endpoints of each line segment is determined as the point where the line segment intersects the normal of the outermost scan point at either end, with coordinates (x_0, y_0), such that the line endpoints (x_e, y_e) are:

$$x_e = \frac{b^2 x_0 - aby_0 - ac}{a^2 + b^2}$$
$$y_e = \frac{a^2 y_0 - abx_0 - bc}{a^2 + b^2}$$

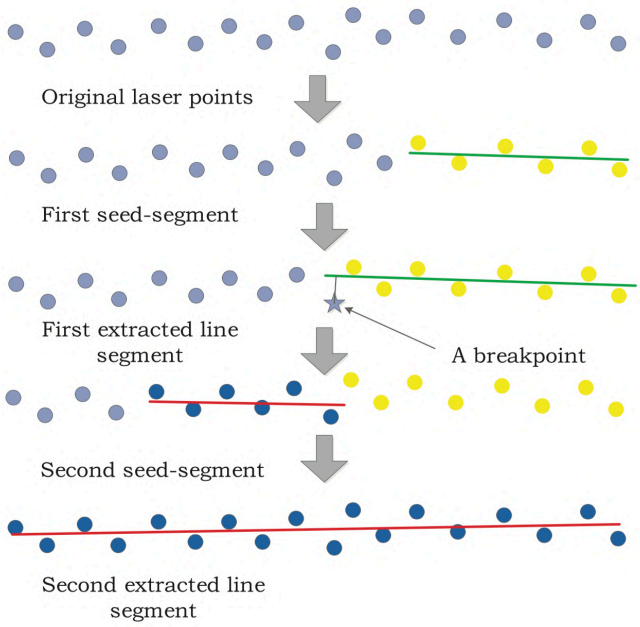


Figure 5.3 Visualization of the line segment extraction process. Taken from [13].

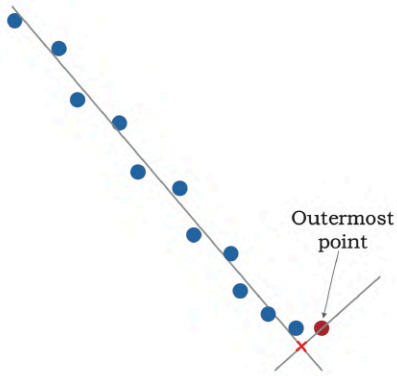


Figure 5.4 Visualization of the endpoint generation process, where the endpoint is the red cross. Taken from [13].

For pseudocode of the line extraction process, taken from [13], refer to Appendix A. After the line segments have been extracted, keypoints are identified as all corner points (intersection points of two non-parallel line segments) and all valid line endpoints. A line endpoint is invalid if:

a) The line is near-aligned with the laser beam, as depicted in the left-hand side of Figure 5.5, i.e. the c-parameter of the line segment is close to zero, as location accuracy of the line due to sparse laser range data is limited;
b) The endpoint is occluded by another line, such as point C occluded by point D in the right-hand side of Figure 5.5, to avoid an incorrectly assigned endpoint that is not at the end of the actual line;
c) The endpoint coincides with the outer fringes of the LiDAR scan angle range, i.e. at the blind zone depicted in Figure 5.6.

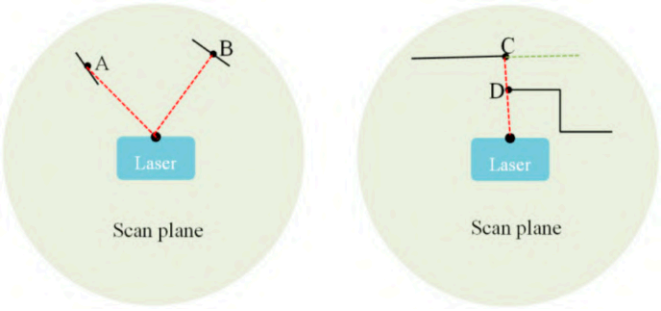


Figure 5.5 The black lines represent the line segments extracted from the LiDAR point cloud, the red dotted lines are the indicative laser beams and the green dotted lines show occluded line regions. Using the keypoint identification criteria established, points A and C are rejected as keypoints. Taken from [23].

Having identified all keypoints and line segments, we can begin carrying out translation and rotation estimation between consecutive scans:

1. *Rotation estimation:* First, line segments in two consecutive scans are matched and paired up by comparing the endpoints of each line segment between the two scans and establishing correspondence if the endpoints are within a threshold distance from each other. Then, the headings of all the lines are calculated in Euclidean 2D space using the inverse tangent function. Lastly, the change in rotation between corresponding pairs of line segments are averaged to estimate the yaw rotation angle $\Delta\psi$ between scans. This change in angle is converted into the 2D rotation matrix:

$$\boldsymbol{R}_k^{k-1} = \begin{bmatrix} \cos(\Delta\psi) & -\sin(\Delta\psi) \\ \sin(\Delta\psi) & \cos(\Delta\psi) \end{bmatrix}$$

Translation estimation: Using the yaw rotation angle from previous step, the coordinate frame of the current line segment and keypoint set is rotated such that the coordinate frames of both scans are aligned. Following this, a similar correspondence search to that of line segment endpoints are done for keypoints, and the difference in x and y coordinates are averaged across all corresponding keypoints. The resulting change in x and y coordinates form the translation estimates Δx and Δy, respectively: $\boldsymbol{t}_k^{k-1} = [\Delta x \quad \Delta y]^T$.

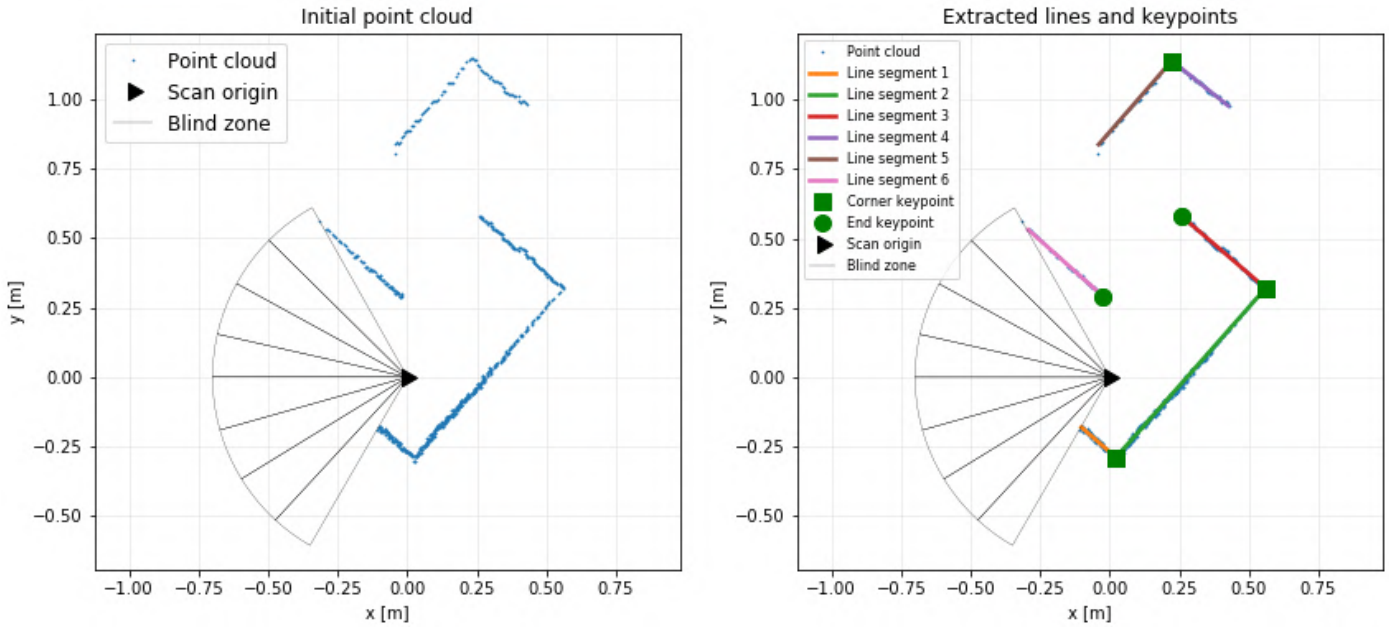


Figure 5.6 Left plot depict the initial point cloud from one LiDAR scan, whereas the right plot shows the extracted line segments and keypoints.

5.5 Mapping

The LiDAR scan matching process produces incremental rotational and translational changes in the system state, specifically the position state variables $x_k, y_k \in \boldsymbol{p}_k$ and the yaw angle ψ_k (or its corresponding 2D rotation matrix). Using the obtained optimal scan-to-scan transformations and the processed LiDAR scans, we can build a map of the environment.

In order to build a map, a global navigation frame must be initialized as reference frame. This was taken as the first LiDAR scan frame with its origin set as the origin of the reference frame. Each scan that follows will then, after processing, be transformed back to the reference frame using either the scan-to-scan translation and rotation arrays obtained from the scan matching algorithms or the position and heading state variables directly.

5.5.1 *Iterative Closest Point (ICP)*

For each point cloud, optimal rotation and translation are stored in a cache with rotation matrices and translation vectors. Since two consecutive rotations are equivalent to one rotation of the matrix product of the two rotation matrices, we can write the following as the transformation of each scan point expressed in the current scan frame, $p_{k,i}^{[k]}$ to the same scan point expressed in the global reference frame $p_{k,i}^{[0]}$:

$$p_{k,i}^{[0]} = \prod_{j=1}^{k} \boldsymbol{R}_j^{j-1^T} p_{k,i}^{[k]} - \sum_{j=1}^{k} \left(\prod_{m=1}^{j} \boldsymbol{R}_m^{m-1^T} \right) \boldsymbol{t}_j^{j-1}$$
$$\forall p_{k,i}^{[k]} \in \mathcal{P}_k$$

We can thus use this global transformation formula to transform the point cloud to the reference frame and stack the transformed scan points onto each other and successively render the features of the environment.

5.5.2 *Feature-based scan matching approach*

The feature-based approach makes use of the system state of the sensor body itself and constructs a rotation matrix with its heading state, and a translation vector based on its Cartesian coordinates in the navigation frame. Using these, the endpoints of the extracted lines of the current point cloud are transformed to the reference frame and the parameters of the rotated lines are recalculated. In order to avoid clutter, correspondence between transformed lines in the current scan and the existing line features in the global frame is found. Then, for corresponding pairs of line segments, a weighted average is applied of the line parameters $\{a, b, c\}$ expressed in the reference coordinate system, with the weights applied depending on how many times a global line has been refitted. For instance, if a line in the global line with parameters $\{a_l^g, b_l^g, c_l^g\}$ has been observed and refitted N_l times, and it is observed again in the next scan with the parameters of the transformed line in the new scan $\{a_l^t, b_l^t, c_l^t\}$, the new parameters $\{a'^g_l, b'^g_l, c'^g_l\}$ are fitted such that

$$a'^g_l = \frac{N_l\, a_l^g + a_l^t}{N_l + 1}, b'^g_l = \frac{N_l\, b_l^g + b_l^t}{N_l + 1}, c'^g_l = \frac{N_l\, c_l^g + c_l^t}{N_l + 1}$$

In the case that no such correspondence is found, the transformed line segment from the current scan will be added to the global map. However, in order to avoid rogue lines cluttering the final map, all line features in the global frame that have not been refitted more than 100 times, i.e. $N_l < 100$, at the end of the SLAM process will be removed.

6 Experimental setup

In order to perform a SLAM accuracy estimation of the sensor data, we required the means to collect and verify the data. As such, an experimental setup was designed and built using to collect data that resembles ground truth as much as possible to minimize errors during data collection. In order to simplify calculations, it has been assumed that the coordinate frame of the LiDAR coincides with the coordinate frame of the experimental model. Furthermore, the body frame of the IMU was sufficiently close to the body frame of the LiDAR; hence, additional transformations between the LiDAR and the INS coordinate frames have been dismissed. A detailed description of the sensor devices and tools used for the experiment and the data collection procedure is outlined in the upcoming subsections.

6.1 Device specifications

The following subsections will detail the sensors and electronic equipment used in the experimental setup for data collection.

6.1.1 *Pixhawk*

The inertial sensors used in the experiment were embedded within a Pixhawk 1 autopilot module, depicted in Figure 6.1, originally manufactured by 3DR. It comes with a 180MHz, 32-bit processor, 256 KB RAM and 2MB flash memory and is connected to the computer board via USB. Documentation for the module exists online at [10]. The sensors included in the Pixhawk that were used in this study have an output frequency of 50Hz and consisted of the following:

- ST Micro L3GD20H 16-bit gyroscope
- ST Micro LSM303D 14-bit accelerometer and magnetometer
- Invensense MPU 6000 3-axis accelerometer and gyroscope

6.1.2 *LiDAR*

The LiDAR sensor used was an URG-04LX Laser Rangefinder, also depicted in Figure 6.1. It is designed and manufactured by Hokuyo and is connected to the computer board via USB, which was very ideal for our purposes due to its light weight and ease of operability. Technical specifications are tabulated in Table 6.1.

Specifications: Hokuyo URG-04LX	
Min range	20mm
Max range	5600mm
Range resolution	1mm
Accuracy	60-1000mm: ±30mm 1000-4095mm: ±3%
Scan angle	240°
Angular resolution	0.36°/scan
Output rate	10Hz (10 scans/sec)
Weight	160 g

Table 6.1 Specifications for the Hokuyo URG-04LX laser rangefinder used in the experimental setup.

6.1.3 *Computer board + battery*

The experimental setup was run via Robot Operating System (ROS) middleware packages in an Ubuntu Kinetic Linux operative system. The software was installed on an Intel NUC Kit D54250WYK computer board with a 2.6GHz Intel® Core™ i5-4250U processor and 4GB DDR3 PC3-12800 RAM. The computer board was strapped onto the side of the experimental model and powered by a Gens Ace lithium polymer battery of 5500MAh, 11.1V and 25C, which was closely attached to the sensors in the experimental box (see

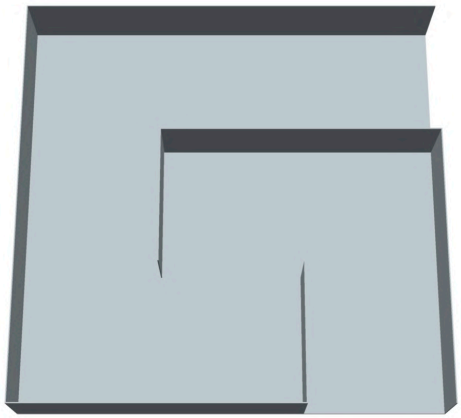
Figure 6.1 Left: Hokuyo URG-04LX LiDAR sensor used in the experiment. Right: Pixhawk 1 autopilot module with inertial sensors used in the experiment.

Figure 6.3(b)). It is worth mentioning that the proximity of the battery to the IMU sensors will undoubtedly cause electromagnetic disturbances, which is why the calibration stage of the IMU (see Section 4.1.4) with the entire setup mounted is crucial.

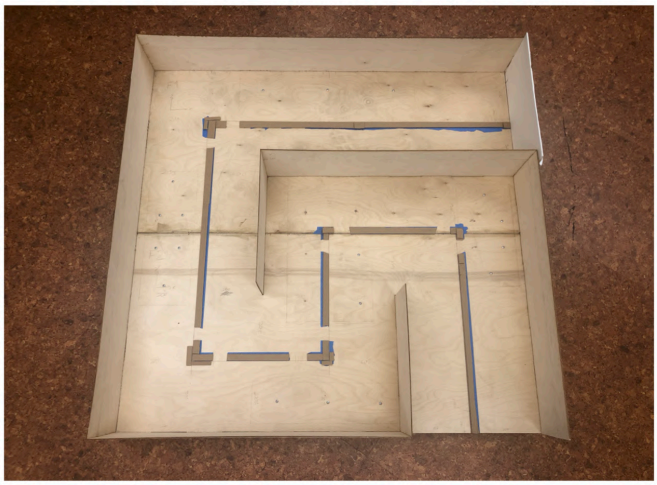
6.2 Experimental environment

In order to collect the data, we needed a stationary and sufficiently undisturbed environment such that we can accurately model the features digitally and compare our SLAM results with the ground truth. As such, a 120cm square, 20 cm tall, three blocks by three blocks maze, was designed for our test environment. The justification for a maze structure was to see if the sensors can manage to navigate themselves in straight lines, as well as gyrate around corners. The choice of size of the maze was based around the scan range of the LiDAR sensor; with the incumbent design, the minimum distance from the laser rangefinder to any obstacle would theoretically be at least 200mm. This was deemed reasonable for the purposes of the experiment, both since the minimum scan distance is sufficiently large such that the LiDAR would yield relatively accurate readings, and that any real-life SLAM-based autonomous vehicle or robot would only safely operate at a large distance from the closest obstacle (ideally it would have been larger for these purposes, but indoor constraints haltered further expansion of the design). A digital 3D rendering of the maze design drawn in AutoCAD is shown in Figure 6.2(a).

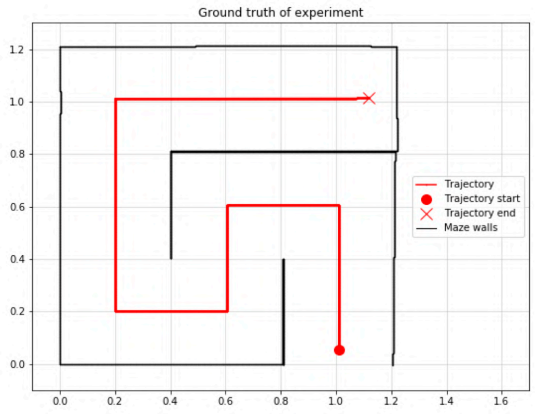
In order to avoid the risk of deforming the walls during testing and thus invalidating the ground



(a)



(b)



(c)

Figure 6.2(a) 3D design of maze in AutoCAD. Each wall and floor piece were separately drawn for the laser cutter.

(b) The actual experimental environment constructed by gluing and screwing laser cut pieces of plywood, with an additional foamboard retrofitted at the end of the maze. Note the cardboard strips that delineate the trajectory for the experimental box.

(c) A numerical rendering of the experimental environment after carefully measuring the dimensions of the real maze.

truth measurements, the wall material had to be sufficiently firm and flexible. As such, plywood was considered the best material for this purpose. Each wall and floor piece of the maze structure was individually drawn with finger joints at each joining edge, and then uploaded to the laser cutter with sheets of plywood. The resulting pieces of plywood were subsequently joined together, glued and fastened with screws. Initial SLAM trials showed com-plications with the open end at the top-right corner; this was resolved by retrofitting a piece of foam-board, covering up the end. The trajectory was marked out by strips of cardboard lined on its side. The real maze is shown in Figure 6.2(b).

After construction of the maze, the wall geometry was precisely measured at the LiDAR scan height (132mm above the floor surface), with initial deflections of the plywood walls taken into account. The trajectory was also measured carefully in relation to the maze walls. The measurements were then used to numerically reproduce the maze walls in Cartesian coordinates, with the bottom-left corner as origin of the coordinate frame and the trajectory start point at $(1.010, 0.056)$ $[m]$. The plot of the ground truth is displayed in Figure 6.2(c).

6.3 Data collection

We needed to ensure accuracy and consistency of the data collection using the sensors in relation to the experimental environment. For this, a box model with a sliding lid was designed in AutoCAD (see Figure 6.3(a)) that could precisely hold the LiDAR sensor, IMU, battery pack and computer board in place as one single experimental package. The box with lid geometry was chosen since it allows for a plane, horizontal platform for the sensors to read data from. The LiDAR could also be stacked on top of the IMU, which simplifies data association since any intersensor transformation of data would be negligible. The box and lid were separately 3D printed with ABS plastic and the complete sensor package can be seen in Figure 6.3(b). The idea was for the computer board to be placed inside the main box underneath the lid, however due to miscalculations of the box dimen-sions a makeshift solution was to attach the board to the side of the box using double-sided tape.

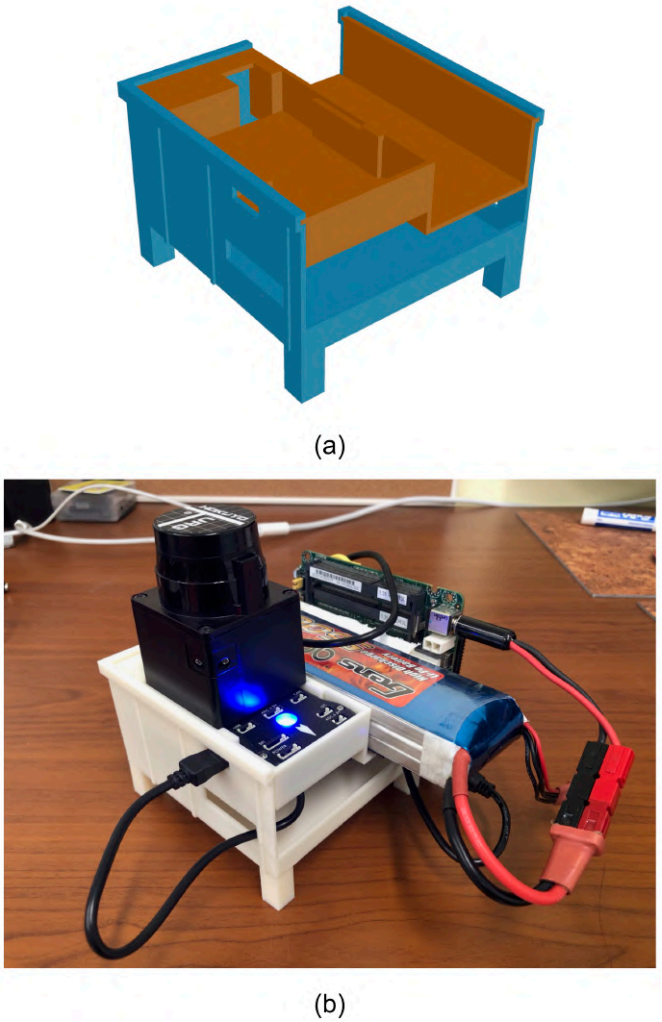
To ensure that the LiDAR scans actually are taken at the marked trajectory, a tracer pin was printed as part of the main box structure. The tip of the pin coincides with the centerline of the LiDAR sensor,

which implies that any LiDAR scan taken from a position where the tip of the tracer pin touches the ground truth trajectory, it can be rightfully assumed that the origin of the obtained point cloud lies as a point on the trajectory. An illustration of this feature is shown in Figure 6.4. With all the sensors and electronic equipment mounted onto the box model, the IMU sensors were calibrated as per Section 4.1.4. The model was subsequently placed at the start of the trajectory in the experimental environment, facing forwards along the trajectory. Using ROS commands, the sensors began registering IMU and LiDAR data which were stored in a local bagfile. The model was left idle for a few seconds upon start-up to collect samples of stationary data which was used for bias estimation. Then, the box was manually pushed through the maze environment, with the tracer pin lodged closely to the taped cardboard that outlines the intended trajectory. At every corner point, a hold and turn movement was made in order to avoid large yaw angles. Careful attention was made as to not obstruct the scan lines with any foreign objects. Ultimately, the sensor data stored in the bagfile was converted to a raw .csv-format, which was imported into a Python IDE for SLAM implementation and reconstruction of the trajectory and environment. For EKF solutions, the process noise covariance $\boldsymbol{Q}_k = \boldsymbol{Q}$ and measurement noise covariance $\boldsymbol{R}_k = \boldsymbol{R}$, both of which remain constant, where $\boldsymbol{\sigma}_f = diag([\sigma_s \quad \sigma_s \quad \sigma_s])$ with subscript s indicating the relevant state variable:

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{\sigma}^2_{imu,f} & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} \\ \mathbf{0}_{3,3} & \boldsymbol{\sigma}^2_{imu,\omega} & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} \\ \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \boldsymbol{\sigma}^2_{imu,\delta u_f} & \mathbf{0}_{3,3} \\ \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \boldsymbol{\sigma}^2_{imu,\delta u_\omega} \end{bmatrix}$$

$$\boldsymbol{R} = \begin{bmatrix} \sigma^2_{lid,p} & 0 & 0 \\ 0 & \sigma^2_{lid,p} & 0 \\ 0 & 0 & \sigma^2_{lid,\psi} \end{bmatrix}$$

Standard deviation	$\sigma_{lid,f}$	$\sigma_{lid,\omega}$	$\sigma_{lid,\delta u_f}$	$\sigma_{lid,\delta u_\omega}$	$\sigma_{lid,p}$	$\sigma_{lid,\psi}$
Value	0.05	10	0.0005	0.01	0.01	0.5
Unit	m/s²	°/s	m/s²	°/s	m	°



(a)

(b)

Figure 6.4(a) 3D design of the box model with sliding lid in AutoCAD. Blue: main box holding cables and computer board. Orange: sliding lid holding sensors and battery pack.

(b) ABS plastic 3D printed box model with sensors, computer board and battery mounted.

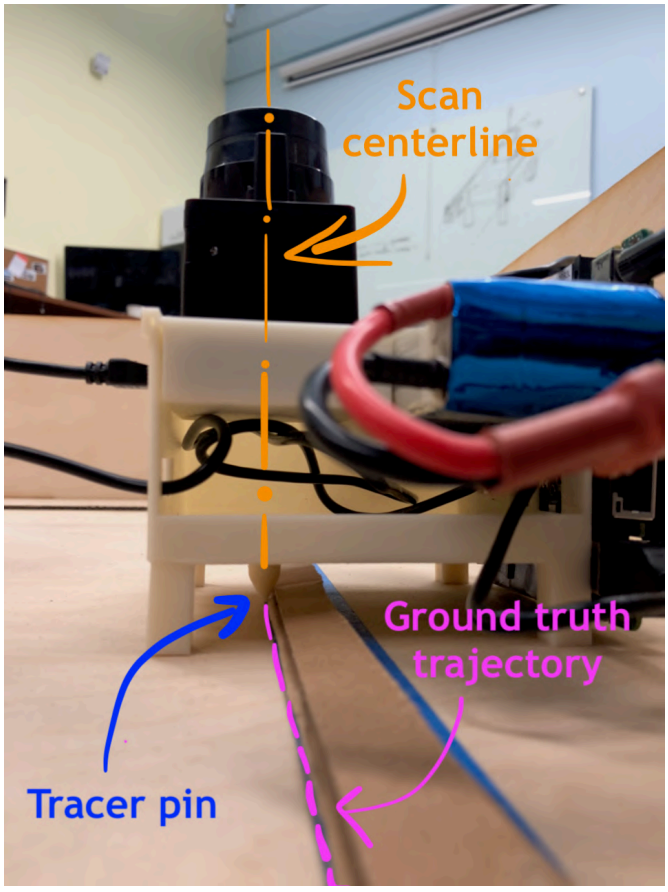


Figure 6.3 Illustration of the tracer pin lodged against the ground truth trajectory, ensuring that each scan is taken from the centerline.

7 Results

After the data was collected, stored in .csv-files and the SLAM implementation was performed on a Python IDE, the result was visualized in a map with Cartesian coordinate system in the global navigation frame. Graphs and tables of residual distance which indicate the deviation error from ground truth of both the estimated trajectory and mapped environment were also included to quantify the performance of the algorithms used. We have included both results from the Kalman filter sensor fusion process as well using the LiDAR data alone with no inertial measurements.

7.1 Estimated trajectory and map

Figure 7.1 shows the estimated trajectory and map of the environment for one test run using four various data processing configurations; a point cloud based method using the ICP algorithm, with and without inertial measurements, and the feature-based approach described in Sections 5.4.2 and 5.5.2, with and without inertial data. A total of five test runs were conducted; the resulting map and trajectory plots are compilated in Appendix B.

As seen, the SLAM process is able to capture the general shape of the maze walls and trajectory. For the particular test results that are shown in Figure

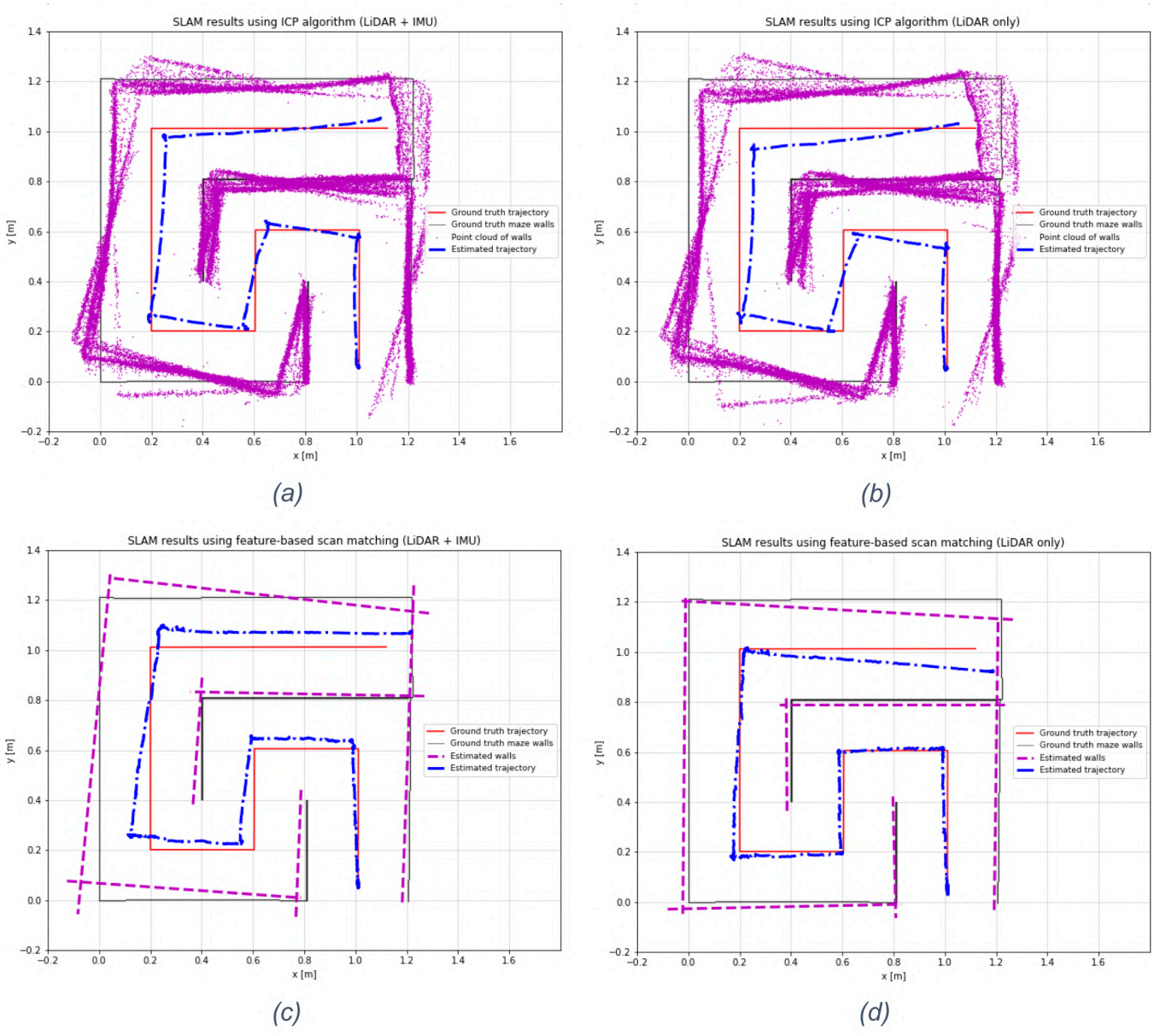


Figure 7.1(a) SLAM results fusing navigation solutions with ICP implementation in an extended Kalman filter, with a down-sampled point cloud of 33,340 scan points, 1/8th of the total global point cloud.

(b) SLAM results based solely on LiDAR data with an ICP implementation, with a down-sampled point cloud of 33,340 scan points, 1/8th of the total global point cloud.

(c) SLAM results fusing navigation solutions with feature-based scan matching approach in an extended Kalman filter.

(d) SLAM results based solely on LiDAR data with a feature-based scan matching approach in an extended Kalman filter.

7.1, the EKF sensor fusion process yields better results for the trajectory and environment using the ICP algorithm, whilst on the other hand it produces worse results using the feature-based algorithms. However, this varies greatly on the test runs due to different control of the experimental box through the maze which inevitably induce inaccuracies in the physical measurements during data collection. The geometry of the walls is slightly distorted using the ICP algorithm, as a result of the point clouds from each individual LiDAR scan not being post-processed to the same extent as the feature-based scan matching approach. In contrast, the feature-based algorithm is more susceptible to line misalignments and "rogue" line segments from line segments extracted in individual scans incorrectly or inadequately matched with the line segments in the global frame. It also handles straight corridors, such as the northernmost corridor at the end of the trajectory, slightly worse where there are relatively few keypoints and line features.

7.2 Comparison and accuracy estimation

To gain a better understanding of the errors, we can define the residual distance of the estimated trajectory from the ground truth trajectory as a metric of deviation error (root mean squared error, RMSE). More specifically, the distance of each position state will be matched with the closest point in the ground truth trajectory. The same has been done for the estimated wall features (scan points in the global point cloud for the ICP algorithm and interpolated points between line ends in the global frame for the feature-based matching approach). Tables 7.1 and 7.2 show the deviation errors for the trajectory and the maze walls, respectively, for the four different data processing configurations of the five test runs. Neither of the two scan matching methods trialed in the experiment can be said to be more favorable than the other in terms of accuracy, even though the average RMSE of the ICP runs are on average lower than that of the feature-based calculations. However, given the large variation in the error metric,

Test run	ICP (LiDAR+IMU)	ICP (only LiDAR)	Feature-based (LiDAR+IMU)	Feature-based (only LiDAR)
1	0.0250	0.0324	0.0444	0.0307
2	0.0556	0.0560	0.0376	0.0421
3	0.0543	0.0374	0.0665	0.1212
4	0.0559	0.0439	0.0561	0.0667
5	0.0506	0.0555	0.0940	0.0884
Average	**0.0483**	**0.0450**	**0.0597**	**0.0698**

Table 7.1 Residual distance or deviation error (in meters) of estimated environment structure from ground truth for five test runs and four different SLAM configurations. The map plots pertaining to each test run are compiled in Appendix B.

Test run	ICP (LiDAR+IMU)	ICP (only LiDAR)	Feature-based (LiDAR+IMU)	Feature-based (only LiDAR)
1	0.0358	0.0443	0.0261	0.0202
2	0.0574	0.0654	0.0330	0.0369
3	0.0366	0.0456	0.0751	0.1145
4	0.0534	0.0550	0.0585	0.0619
5	0.0639	0.0561	0.0739	0.0892
Average	**0.0494**	**0.0533**	**0.0534**	**0.0645**

Table 7.2 Residual distance or deviation error (in meters) maze walls of estimated trajectory from ground truth trajectory for five test runs and four different SLAM configurations. The map plots pertaining to each test run are compiled in Appendix B.

the small sample size and the fact that large errors may as well arise from inadequate data collection methods (such as bumps, vibrations or a misplaced hand in front of scan lines), it is hard to say whether a SLAM configuration performs better than another.

What we can say, however, is that the errors start to accumulate as the experiment progresses, as seen in Figure 7.2. Here, progression graphs of the deviation error are shown for the five test runs (one for each row) and for each SLAM configuration (one for each column). The errors tend to stay relatively close to zero at the beginning and grow with each time step or LiDAR scan, indicating a need for loop closure at the end of the test. Clearly, the results are within an acceptable range at a low-budget setup, however more sophisticated algorithms are crucial before implementation in any real-life system.

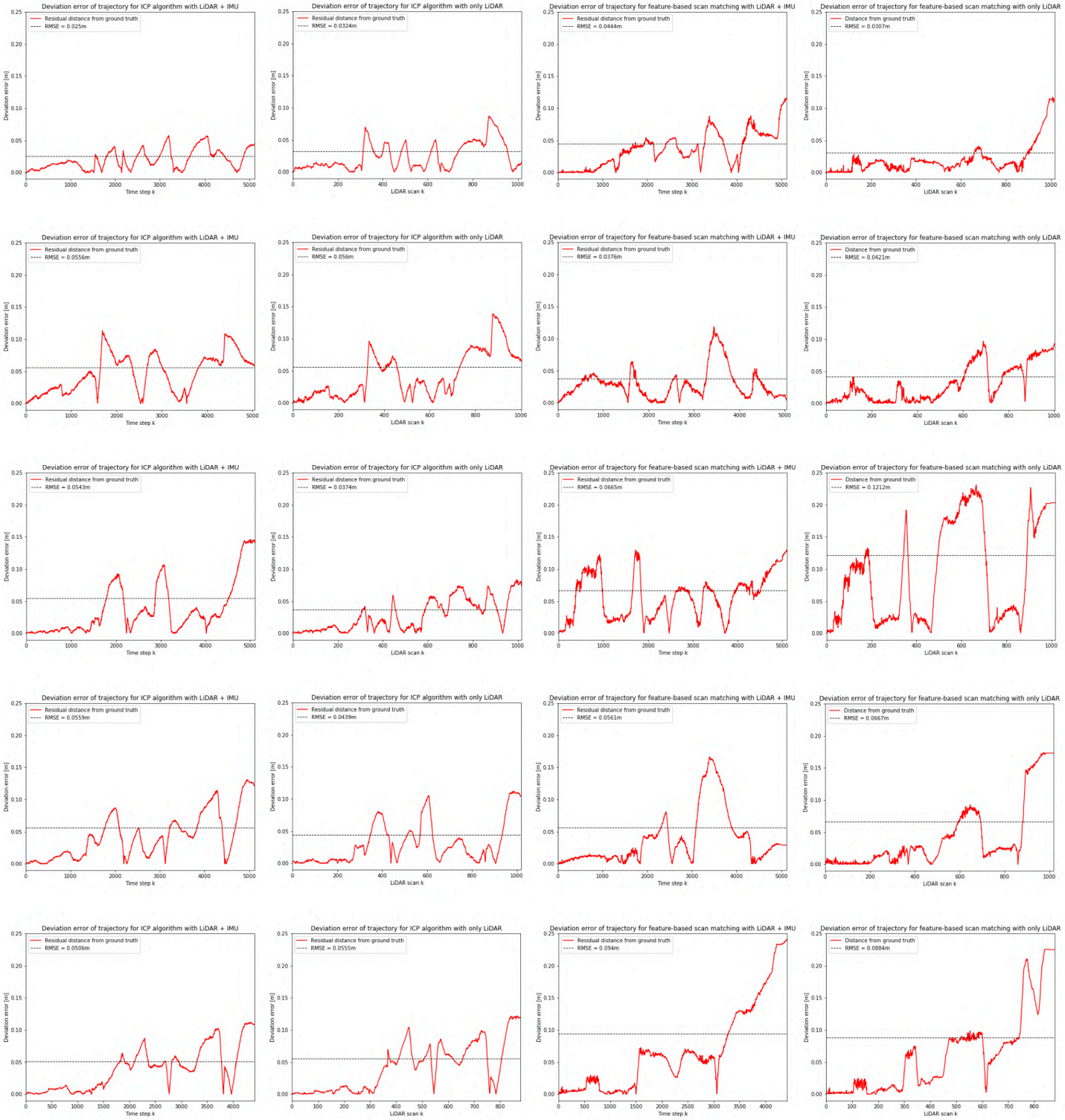


Figure 7.2 Deviation error graphs showing the residual distance of estimated trajectory from ground truth distance for each time step in EKF implementations, and each LiDAR scan for non-sensor fusion tests. Each row represent test run 1 through 5, in that order, and each column are test runs using ICP with IMU, ICP without IMU, feature-based scan matching approach with IMU and feature-based scan matching approach without IMU.

8 Discussion

This particular study has been making an attempt on performing accurate SLAM in a small-scale, self-designed environment using a laser rangefinder and inertial sensors inside an IMU. Despite having managed to capture the general outline of the environment as well as the walls, all within an error margin of 4.5 to 7 cm, we are unaware whether there exist more suitable algorithms for this particular type of environment. Had there been more time, more approaches would have been trialed for the scan matching step as well as the sensor fusion process, and more attention would have been set aside to fine-tuning the hyperparameters set for the covariances in the EKF process. Furthermore, it would also be beneficial to try other environment types and scales, to ensure that the algorithms also work in larger environments with different types of features apart from straight lines and corners. For a full 3D rendering of the environment, a visual-based sensor such as camera sensors or a second LiDAR must be used, since a single static LiDAR scanner only provides information in two dimensions. Further room for improvement could accommodate these considerations.

Ultimately, the main observation of this study was that the results tended to vary greatly depending on the test run made. At the time of data collection, even though the trajectory and environment remain static, the data collection process does not. Oftentimes, it was hard to manually collect the data as the experimental box would bump into the cardboard strip lines on the floor of the maze. This abruptness of motion is not reflected in the sensor data as an anomalous of motion and may yield high inaccuracies during post-processing of the data. Further investigations should be made into how to identify anomalous motions and treat them effectively in any SLAM process.

Another observation made from the test results was that the deviation errors seemed to pile up during the experimental run as we move away from the known initial position. In cases where only relative positioning systems such as IMU and LiDAR are used in a SLAM algorithm, which entails that pose estimations are a cumulative process since the pose change is estimated and added rather than the pose itself, any errors induced in that process will also add up. Accurate positioning systems using only sensor that measure relative pose changes like inertial sensors and LiDAR necessitate a tethering system such as loop closure which means propagating the residual error backwards when the system recognizes a feature that it has previously seen, thus “closing the loop”. Ultimately, further sensors and features, in particular those that leverage absolute positioning such as GNSS or known positions of certain features, could be considered in future implementations of the SLAM process.

9 Conclusion

This study has focused on implementing 2D SLAM on a small-scale structured maze environment with the aid of a low-cost and lightweight LiDAR sensor and inertial sensors inside an IMU. The data streams were fused together in an integrated navigation filter built on an extended Kalman filter to achieve the optimal state estimations with the obtained data. Two alternatives of the scan matching stage were tested, one using ICP, a traditional point-to-point scan method, and the other using a custom-designed line and keypoint feature extraction-based approach. The test was run multiple times and the results were accordingly presented in this report.

The estimated mappings in the test runs were successfully able to capture the features of the real environment adequately, with slight anomalies occurring when the data was lacking sufficient features to match with preceding scans, as well as physical errors during data collection. The state estimations performed in all experiments were also able to follow the intended trajectory to an acceptable degree, provided that the trajectory is well-represented in the collected data. Further work should be investigating solving the issues regarding reliability and adaptability of the algorithms implemented and adding loop closure functions to propagate mapping errors over time for this particular environment. In the long-run, more experiments should be performed using different scan matching approaches, sensor fusion implementations and test environments.

References

[1] J. Alexandersson & O. Nordin, *Implementation of SLAM algorithms in a small-scale vehicle using model-based development*. Department of Electrical Engineering, Linköping University, Sweden, 2017.

[2] P. J. Besl & N. D. McKay, ”A method for registration of 3-D shapes” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2), pp. 239-256, 1992.

[3] P. Biber & W. Straßer, “The Normal Distributions Transform: A New Approach to Laser Scan Matching” in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, NV, USA, pp. 2743-2748, 2003.

[4] A. K. Brown, “Test Results of a GPS/Inertial Navigation System Using a Low-Cost MEMS”, in *Proceedings of 11th Annual Saint Petersburg International Conference on Integrated Navigation System*, St. Petersburg, Russia, 2004.

[5] Y. Chen & G. Medioni, “Object modelling by registration of multiple range images” in *Image and Vision Computing*, 10(3), pp. 145-155, 1992.

[6] I. J. Cox, “Blanche–an experiment in guidance and navigation of an autonomous robot vehicle” in *IEEE Transactions on Robotics and Automation*, 7(2), pp. 193-204, 1991.

[7] A. J. Davison, I. D. Reid, N. D. Molton & O. Stasse, “MonoSLAM: Real-Time Single Camera SLAM” in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6), 2007.

[8] H. Deilamsalehy & T. C. Havens, “Sensor Fused Three-dimensional Localization Using IMU, Camera and LiDAR” in *2016 IEEE SENSORS*, Orlando, FL, USA, 2016.

[9] A. Diosi & L. Kleeman, “Fast Laser Scan Matching using Polar Coordinates” in *The International Journal of Robotics Research*, 26(10), pp. 1125-1153, 2007.

[10] Dronecode, *Pixhawk 1 Flight Controller*. [online] Available at: https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk.html (Accessed: 07/18/2019), 2019.

[11] Dronecode, *QGroundControl User Guide*. [online] Available at: https://docs.qgroundcontrol.com/en/ (Accessed: 07/08/2019), 2019.

[12] J. A. Farrell, *Aided Navigation: GPS with High Rate Sensors*. McGraw-Hill, New York, NY, USA, 2008.

[13] H. Gao, X. Zhang, Y. Fang & J. Yuan, ”A line segment extraction algorithm using laser data based on seeded region growing” in *International Journal of Advanced Robotic Systems*, 15, pp. 1-10, 2018.

[14] M. S. Grewal, L. R. Weill & A. P. Andrews, *Global Positioning Systems, Inertial Navigation, and Integration, 2nd Edition*. John Wiley & Sons, Inc., New York, NY, USA, 2007.

[15] Y. Jia, *Quaternions and Rotations*. [online] Available at: http://graphics.stanford.edu/courses/cs348a-17-winter/Papers/quaternion.pdf (Accessed: 07/05/2019), 2013.

[16] R. E. Kalman, “A New Approach to Linear Filtering and Prediction Problems” in Transactions of the ASME–Journal of Basic Engineering, 82(D), pp. 35-45, 1960.

[17] A. Khenkin, *Sonic Nirvana: MEMS Accelerometers as Acoustic Pickups in Musical Instruments*. [online] Available at: https://www.fierceelectronics.com/embedded/sonic-nirvana-mems-accelerometers-as-acoustic-pickups-musical-instruments (Accessed: 07/08/2019), 2009.

[18] L. Kleeman, "Advanced Sonar and Odometry Error Modeling for Simultaneous Localisation and Map Building" in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, NV, USA, pp. 699-704, 2003.

[19] S. Kohlbrecher, O. von Stryk, J. Meyer & U. Klingauf, "A Flexible and Scalable SLAM System with Full 3D Motion Estimation" in *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, 2011.

[20] M. Kok, J. D. Hol, T. B. Schön, "Using Inertial Sensors for Position and Orientation Estimation" in *Foundations and Trends in Signal Processing*, 11(1-2), pp. 1-153, 2017.

[21] J. Konecny, M. Prauzek, P. Kromer & P. Musilek, "Novel Point-to-Point Scan Matching Algorithm Based on Cross-Correlation" in *Mobile Information Systems*, 2016(15), pp. 1-11, 2016.

[22] G. A. Kumar, A. K. Patil, R. Patil, S. S. Park & Y. H Chai, "A LiDAR and IMU Integrated Indoor Navigation System for UAVs and Its Application in Real-Time Pipeline Classification" in *Sensors*, 17(6), 2017.

[23] J. Li, R. Zhong, Q. Hu & M. Ai, "Feature-Based Laser Scan Matching and Its Application for Indoor Mapping" in *Sensors*, 16(1265), 2016.

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

[25] K. Lingemann, H. Surmann, A. Nüchter & J. Hertzberg, " Indoor and outdoor localization for fast mobile robots" in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, pp. 2185-2190, 2004.

[26] F. Lu & E. Milios, "Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans" in *Journal of Intelligent and Robotic Systems*, 18(3), pp. 249-275, 1997.

[27] R. Mur-Artal & J. D. Tardós, "ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras" in *IEEE Transactions on Robotics*, 33(5), pp. 1255-1262, 2017.

[28] V. Nguyen, A. Martinelli, N. Thomas & R. Siegwart, "A Comparison of Line Extraction Algorithms using 2D Laser Rangefinder for Indoor Mobile Robots" in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Edmonton, Alta., Canada, 2005.

[29] NovAtel, *IMU Errors and Their Effects*. [online] Available at: https://www.novatel.com/assets/Documents/Bulletins/APN064.pdf (Accessed: 07/12/2019), 2014.

[30] R. Opromolla, G. Fasano, G. Rufino, M. Grassi & A. Savvaris, "LIDAR-Inertial Integration for UAV Localization and Mapping in Complex Environments" in *2016 International Conference on Unmanned Aerial Systems*, Arlington, VA, USA, 2016.

[31] Y. Pei, S. Biswas, D. S. Fussell & K. Pingali, *An Elementary Introduction to Kalman Filtering*, 2017.

[32] D. Tedaldi, A. Pretto & E. Menegatti, “A Robust and Easy to Implement Method for IMU Calibration without External Equipments” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2014.

[33] W. Travis, A. T. Simmons & D. M. Bevly, ”Corridor Navigation with a LiDAR/INS Kalman Filter Solution” in *IEEE Proceedings. Intelligent Vehicles Symposium*, Las Vegas, NV, USA, 2005.

[34] S. Yun, Y. J. Lee & S. Sung, “IMU/Vision/Lidar Integrated Navigation System in GNSS Denied Environments” in *2013 IEEE Aerospace Conference*, Big Sky, MT, USA, 2013.

[35] S. Zhao & J. A. Farrell, “2D LIDAR Aided INS for Vehicle Positioning in Urban Environments” in *2013 IEEE International Conference on Control Applications*, Hyderabad, India, 2013.

Appendix A

Require: $N_p, \epsilon, \delta, S_{num}, P_{min}$
Initialization: $flag = ture$
for $i = 1 \rightarrow (N_p - P_{min})$ **do**
  $j \leftarrow i + S_{num}$
  fit ***Seed***(i, j)
  for $k = i \rightarrow j$ **do**
    obtain the predicted point P'_k
    $d_1 \leftarrow distance\ from\ P_k\ to\ P'_k$
    if $d_1 > \delta$ **then**
      $flag = false$
      break
    end if
    $d_2 \leftarrow distance\ from\ P_k\ to$ ***Seed***(i, j)
    if $d_2 > \epsilon$ **then**
      $flag = false$
      break
    end if
  end for
  if $flag == true$ **then**
    return ***Seed***(i, j)
  end if
end for

Figure A.2 Algorithm for seed detection used in the feature-based scan matching approach. Taken from [13].

Require: ***Seed***$(i, j), N_p, P_{min}, L_{min}, \epsilon$
Initialization: ***Line***$(P_b, P_f) \leftarrow$ ***Seed***$(i, j), L_l = 0, P_l = 0$
$P_f = j + 1, P_b = i - 1$
while distance(P_f, ***Line***) $< \epsilon$ **do**
  if $P_f > N_p$ **then**
    break
  else
    refit ***Line***(P_b, P_f)
  end if
  $P_f \leftarrow P_f + 1$
end while
$P_f \leftarrow P_f - 1$
while distance(P_b, ***Line***) $< \epsilon$ **do**
  if $P_b < 1$ **then**
    break
  else
    refit ***Line***(P_b, P_f)
  end if
  $P_b \leftarrow P_b - 1$
end while
$P_b \leftarrow P_b + 1$
obtain L_l, P_l from ***Line***(P_b, P_f)
if ($L_l \geqslant L_{min}$) & ($P_l \geqslant P_{min}$) **then**
  return ***Line***(P_b, P_f) with ***Parameters***(a, b, c)
end if

Figure A.1 Algorithm for region growing used in the feature-based scan matching approach. Taken from [13].

Require: N_l (the number of line segments)
for $i = 1 \rightarrow N_l - 1$ **do**
  $j \leftarrow i + 1$
  EndPoint index of ***Line***$_i$:(m_1, n_1)
  EndPoint index of ***Line***$_j$:(m_2, n_2)
  if $m_2 \leqslant n_1$ **then**
    for $k = m_2 \rightarrow n_1$ **do**
      $d_k^i =$ distance(P_k, ***line***$_i$)
      $d_k^j =$ distance(P_k, ***line***$_j$)
      if $d_k^i < d_k^j$ **then**
        continue
      else
        break
      end if
    end for
    $n_1 \leftarrow k - 1$
    $m_2 \leftarrow k$
  else
    break
  end if
  refit ***Line***(m_1, n_1)
  refit ***Line***(m_2, n_2)
end for
return line segments without overlap region

Figure A.3 Algorithm for overlap processing used in the feature-based scan matching approach. Taken from [13].

Appendix B

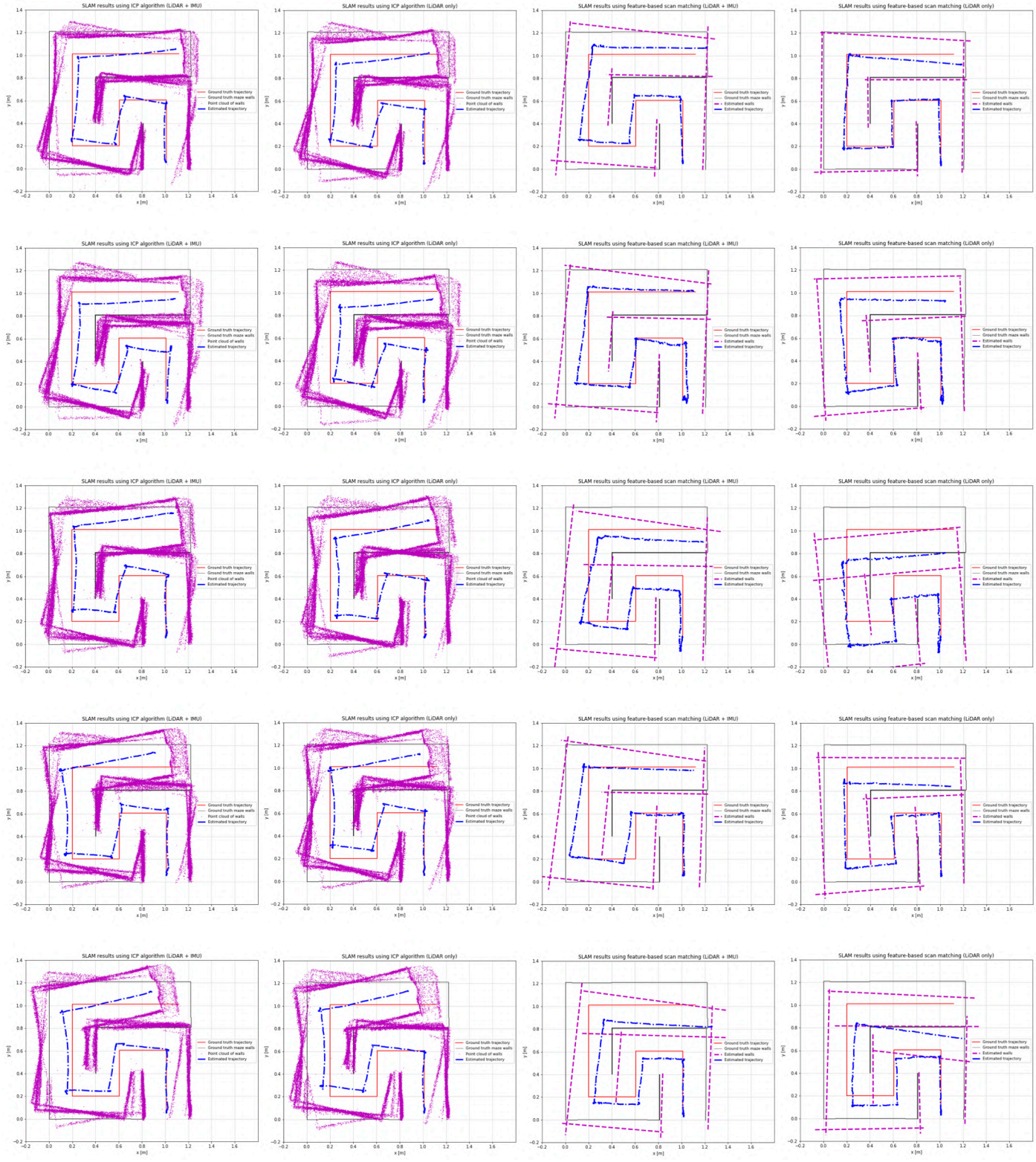


Figure B.1 Estimated trajectory and environment mapping plots, with ground truth to compare performances of each SLAM algorithm configuration. Each row represent test run 1 through 5, in that order, and each column are test runs using ICP with IMU, ICP without IMU, feature-based scan matching approach with IMU and feature-based scan matching approach without IMU.