Implementation of Multi-sensor Perception System for Bipedal Robot

Thesis

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By
Chayapol Beokhaimook
Graduate Program in Mechanical Engineering

The Ohio State University
2021

Thesis Committee:
Professor Ayonga Hereid, Advisor
Professor Keith Redmill

ABSTRACT

Bipedal robots are becoming more popular in performing tasks in an environment that is designed for humans. For this purpose, most bipedal robots are equipped with various sensors to sense the robot's environment. From the measurements of the sensors, a perception system is implemented to translate and convert the raw data into a meaningful format corresponding to the tasks and also provide safety for humans, properties in the environment as well as the robot itself. This thesis presents the implementation of a perception system using various sensors available to a bipedal robot, Digit, to obtain objectively useful information of the environment as well as the state of the robot itself. Various methods of data processing were applied to available sensor measurements, then a mapping algorithm was implemented to generate a 3D model of the environment. Simultaneous localization and mapping (SLAM) algorithm was also implemented to perform mapping and provide odometry for localization in the absence of an external source of odometry. We found that performing SLAM using Light Detection and Ranging sensor (LiDAR) performs exceptionally well on the bipedal robot in closed indoor space. Additionally, state estimation is implemented with Invariant Extended Kalman filter using inertial measurement data and the assumption of contact points to predict the state of the robot over time. The performance of position estimation from Invariant Extended Kalman filter and odometry from LiDAR SLAM is compared with the default state estimator from Digit itself which are demonstrated through an experiment with ground truth reference.

Dedicated to my family.

ACKNOWLEDGEMENTS

Firstly, I would like to thank my advisor, Dr. Ayonga Hereid, who guided me and provided me with all the great advice both in researches and in real life. He has deep insights into robotics and is very helpful when I encountered any problem during my study. I appreciate his compassion and support for his students during the difficult times of the pandemic. Also, I would like to thank Dr. Keith Redmill for accepting to be one of the thesis committees.

Special thanks to my lab mates who helped me with my research and provide insights into problems I have. I would like to thank Guillermo, who helped me with all kinds of problems from debugging codes to aiding me with my experiments. Also, Victor, who helped me with math and programming among other things. Finally, thank you Archit and Caffrey, for helping me with various tasks since I joined the laboratory.

Most importantly, I would like to thank my parents and my brother for their heartfelt support during my Master's degree journey, especially during this difficult time from the pandemic. I wouldn't have completed this journey without all their support.

VITA

September 2019 ..... B.Eng., Mechanical and Aerospace Engineering,
Tohoku University, Sendai

Fields of Study

Major Field: Mechanical Engineering (Robotics)

Table of Contents

ABSTRACT . . . . . **ii**

DEDICATION . . . . . **iii**

ACKNOWLEDGEMENTS . . . . . **iv**

VITA . . . . . **v**

LIST OF TABLES . . . . . **ix**

LIST OF FIGURES . . . . . **xi**

LIST OF ALGORITHMS . . . . . **xii**

1 INTRODUCTION **1**

1.1 Related work 2

1.1.1 Sensor Data Processing 3

1.1.2 3D Scene Reconstruction 3

1.1.3 Simultaneous Localization and Mapping 4

1.1.4 State estimation 5

1.2 Objective of the Thesis 6

1.3 Organization of the Thesis 8

2 SENSORS AND DATA MANIPULATION **9**

2.1 Sensor components 10

2.1.1 Depth cameras . . . 10
2.1.2 3D LiDAR sensor . . . 11
2.1.3 Visual camera . . . 12
2.1.4 Inertial measurement unit . . . 14
2.2 Point cloud manipulation . . . 14
2.2.1 Voxel filtering . . . 14
2.2.2 Ground segmentation . . . 15
2.2.3 Point cloud fusion . . . 19
2.3 Summary . . . 20

3 MAPPING AND LOCALIZATION . . . 21
3.1 3D scene reconstruction using OctoMap . . . 21
3.1.1 Octree . . . 21
3.1.2 Probabilistic Sensor Fusion . . . 22
3.1.3 Occupancy grid map and application . . . 25
3.2 Simultaneous localization and mapping (SLAM) using LiDAR sensor . . . 26
3.2.1 LiDAR Odometry and Mapping . . . 26
3.2.2 Feature point extraction . . . 28
3.2.3 Feature correspondence . . . 29
3.2.4 Motion Estimation . . . 31
3.2.5 LiDAR mapping . . . 33
3.3 Summary . . . 33

4 STATE ESTIMATION . . . 35
4.1 Extended Kalman Filtering . . . 36
4.1.1 Propagation step . . . 37
4.1.2 Update step . . . 38
4.2 Lie groups theory introduction . . . 40
4.2.1 Matrix Lie Groups . . . 40

4.2.2 Uncertainty and error definition . . . 42
4.2.3 Important theorems for Invariant EKF . . . 43
4.3 Invariant Extended Kalman Filtering . . . 44
4.3.1 System and observation models . . . 45
4.3.2 Error definition . . . 45
4.3.3 Uncertainty evaluation . . . 47
4.3.4 Summary . . . 48
4.4 Contact measurement implementation of InEKF . . . 50
4.4.1 State space definition . . . 51
4.4.2 System dynamics . . . 52
4.4.3 Error formulation . . . 54
4.4.4 Measurement and update model . . . 57
4.4.5 Kalman gain calculation . . . 57
4.5 Experiment on Digit . . . 59
4.5.1 Experiment setup . . . 59
4.5.2 Experimental results . . . 60
4.6 Summary . . . 66

5 CONCLUSION . . . 67
5.1 Conclusion . . . 67
5.2 Future work . . . 69
5.2.1 Verification of state estimation assumption and parameters . . . 69
5.2.2 Experimental Implementations . . . 69
5.2.3 Integration of LiDAR odometry to Invariant EKF filter . . . 69

BIBLIOGRAPHY . . . 75

List of Tables

2.1 Number of points in point cloud before and after voxel grid filtering 15

List of Figures

2.1 Digit bipedal robot developed by "Agility Robotic" 9
2.2 Depth cameras at the pelvis 10
2.3 Point cloud obtained from depth cameras 11
2.4 LiDAR sensor and visual cameras installed on the torso of Digit. 12
2.5 Point cloud from LiDAR sensor 13
2.6 Filtered point cloud 16
2.7 Point cloud are segmented in to ground cloud(Green) and obstacle cloud(Red) 18
2.8 Fused point cloud from LiDAR (White), obstacle cloud (Red) and Ground cloud (Green) 20

3.1 Octree data structure where each cubic volume is segmented into 8 sub-volumes and each cells will store occupancy values [19] 22
3.2 3D scene reconstruction using OctoMap 24
3.3 Projected 2D map with discrete grid values as occupancy grid map 25
3.4 Overview of the LOAM system [45] 27
3.5 Principal to match features with their correspondences depicted in [45] . . . 30
3.6 LOAM constructs the 3D amp of a room within indoor environment using point cloud. 34

4.1 Architecture of EKF depicted in [4] 39
4.2 Mapping between spaces in matrix Lie groups depicted in [4] 41
4.3 Invariant EKF architecture depicted in [4] 50

4.4 IMU frame and contact frame positions on Digit 51
4.5 Experiment setup with motion capture system 60
4.6 Estimate of x position over time from different sources compared to ground truth 61
4.7 Estimate of y position over time from different sources compared to ground truth 62
4.8 Estimate of X-Y position from different sources compared to ground truth . 63
4.9 Estimate of linear velocity in x-direction from different sources 64
4.10 Estimate of linear velocity in y-direction from different sources 65

List of Algorithms

1 RANSAC algorithm 18
2 Extended Kalman Filter 40

CHAPTER 1

INTRODUCTION

Recently, autonomous navigation and behavior have been a very popular research topic. The importance and application of autonomous behavior are becoming more pronounced in industrial production, medical procedure and search and rescue operations, etc. These technologies can replace people or traditional measures in mundane and dangerous tasks. The basic requirements for autonomous navigation are having a good understanding of the external state of the environment, knowing the accurate internal state of the agent within the environment as well as having a good understanding of the task. These complex systems are constructed from designing multi-layer modules such as control, manipulation, and perception module. Then, all modules are integrated to form a complex robust autonomous system which is one of the biggest challenges in an engineering perspective.

Perceiving the environment is the very first step toward autonomous behaviors. The idea of perception might seem simple for humans because we can understand what we see through our eye or we can interpret the sounds we hear from our own cumulative experiences, but for a machine, these processes are not straightforward. A machine has to sense the information of the environment and then interpret what it perceived before performing any action. This information was acquired through various types of sensors such as camera, LiDAR (light detection and ranging) sensors, RADAR (radio detection and ranging), IMU (inertial measurement unit), and RGB-D camera, etc. Each sensor provides us with different kinds of data of the environment, for example, LiDAR and depth camera provides us with the spatial structure of the environment in the form of point clouds or visual camera which

output images in the form of pixels, etc. In an autonomous system or robotic system, there are usually multiple sensors collecting different types of data simultaneously to get a better understanding of the environment.

After an agent sensed the information of the environment through sensors, it has to fuse those data in a meaningful way and estimate the state of the environment as well as the agent itself. For example, the point cloud from continuous scans of a LiDAR can be used to reconstruct a three-dimensional scene of the environment and localize itself within the constructed scene at the same time. This problem is referred to as SLAM (Simultaneous localization and mapping). SLAM problem has been used in various kinds of applications. One of the most famous applications is in autonomous vehicles, where navigation relies heavily on the spatial structure of the environment. We can also use internal sensors such as IMU to estimate the state of the agent through linear acceleration and angular velocity. This robust estimation is commonly done through the Kalman family of filters such as the Extended Kalman filter (EKF) to reduce the drift or error that may accumulate over time.

There is various kind of agents that depends on the perception system. One of the most popular fields in the autonomous robot, especially the robots that operate in a dynamic environment such as drones, wheeled mobile robots, quadruped, and bipedal robots. Most mobile robots are designed to safely operate and transverse in an unknown environment with dynamic objects which makes accurate and robust perception a very crucial part of those systems. This thesis will focus on implementing the perception system on the humanoid bipedal robot which has become one of the popular choices in human-robot interaction tasks due to their human-like appearance and the ability to deal with various conditions of ground clearance.

1.1 Related work

There are many layers to the perception module in robotic or autonomous applications. Those aspects range from sensor fusion, data processing, mapping, object detection, and state estimation, etc. Many interesting methods and approaches to solve corresponding fields were proposed in the past few years along with the improvement of quality and cost

of advanced sensors like LiDAR and the state-of-the-art optimization algorithm.

1.1.1 Sensor Data Processing

Sensor data processing is the first procedure that has to be executed when we obtain measurements from various sensors. For depth-related data or point cloud data, one of the most used libraries was Point Cloud Library or PCL [36]. The library is based on linear algebra calculation in Eigen library [15]. It consists of useful tools for manipulating point cloud data, of which, their performance was demonstrated in various papers [26, 22]. Those tools include point cloud registration which is used to align 3D point cloud from one scan with another set of point cloud[18], ground segmentation tools used to extract ground plane or other geometrical shapes using RAndom SAmpling Consensus method or RANSAC [11] from a point cloud scene [22, 24, 34], and filtering tools to remove unwanted data and downsampling the point cloud to reduce the size of data. Those are some examples of the point cloud processing technique that is commonly used recently.

Other than depth-related data, another common form of sensing is through image or visual scenes. These kinds of data have countless applications such as obstacle detection for autonomous vehicles [41], object recognition using visual camera [13] and doing feature-based mapping method called ORB-SLAM [28]. Most camera has 3 color channels, but some camera has a depth channel in addition to RGB to perceive the depth information and embed on into the image pixel simultaneously, such camera is called RGB-D camera which can be used in 3D scene reconstruction of the environment [46].

1.1.2 3D Scene Reconstruction

Three-dimensional information of the environment is one of the most important areas in autonomous systems. To represent spatial characteristics of an environment, there are various kinds of representations proposed, such as raw point cloud through point cloud registration mentioned in Section 1.1.1, elevation map [21] or multi-level surface map of the terrain [43]. Another popular method is to discretize the space into cubic volumes called voxels. This idea was implemented in 3D SLAM applications in [30], however, due to fact that free space

and unknown area are not modeled into the system, it cannot be used in the application where free space and unknown space need to be represented. Another implementation of the mentioned approach where free space and unknown space are taken into consideration is called "Octomap" [19] which represents the 3D space as components of volumetric structure like octree with probabilistic occupancy value which yield high accuracy and efficiency. Furthermore, OctoMap can give simplified representation such as occupancy grid map where 3D information is projected on 2D plane or height map which is occupancy grid map with height information, which are more practical and is sufficient in general tasks due to smaller data size.

1.1.3 Simultaneous Localization and Mapping

Simultaneous Localization and Mapping or SLAM is a complex problem where a map and localization process occurs at the same time. This is important for navigation purposes where the external reference of state such as GPS is not available. There are multiple methods to handle SLAM problems including the various combination of sensors and optimization methods. Camera is a common choice of sensor due to the availability and relatively low cost of the sensor. ORB-SLAM [28], which was briefly mentioned in Section 1.1.1, is one of the most famous open-source SLAM algorithm that deals with a multitude of visual cameras including monocular, stereo, and fisheye cameras. Another algorithm is MonoSLAM [9] where the main focus is to obtain the trajectory of the agent using only one monocular camera.

Recently, the cost of more advanced sensors such as LiDAR and Radar is much lower than before, so more SLAM algorithm that is based on these sensors were proposed. The simple variation of LiDAR such as 2D sensor can be used to create 2D map of the environment while localizing the robot in the self-created map with the help of loop-closure technique [17, 31]. A more common choice of LiDAR sensor is the three-dimensional LiDAR which composes of multiple bands or channels of rotating laser to collect a set of point clouds with 3D coordinates. A state-of-the-art 3D LiDAR SLAM algorithm is proposed in [45] called Lidar Odometry and Mapping (LOAM), where the system consists of two algorithms

running in parallel to estimate the motion of the LiDAR and to accurately map the point cloud to a map. This method achieve high performance in estimating odometry with 0.55% error on KITTI dataset [12] and ranked as one of the best SLAM algorithms on the KITTI benchmark. There are other variations that are derived from LOAM [38, 39] where IMU is introduced into the system to improve the odometry estimation in SLAM.

Other than choices of sensors, the main approach to SLAM differs for different research. One of them is graph-based SLAM [14], where states or pose of the agent is formulated as nodes with edges representing the constraint between each node, then a map can be constructed using the spatial configuration and the associated measurements modeled by the edges. Another approach that has become more popular recently is using optimization-based SLAM where probabilistic inferences are considered such as convex optimization SLAM [23]. However, the idea of these optimization-based methods will not be covered in the scope of this thesis.

1.1.4 State estimation

State estimation is a crucial process in control modules. For a linear system, the state can easily be predicted using Kalman filter introduced in 1960. The idea is to fuse measurement information into the deterministic propagation model with a known control input to get accurate state prediction. However, in most cases the system is nonlinear, so a variant of Kalman filter called Extended Kalman Filter (EKF) with better convergence property [40, 20] is used instead of linear Kalman filter. EKF will linearize the error model using first-Taylor expansion on the nonlinear functions in system dynamics. The application of EKF is very broad, some examples include attitude estimation [42] and multi-sensor fusion in UUV (Unmanned Underwater Vehicle) [33].

Generally, for legged robots, the state estimation uses EKF with IMU measurement fused with nonlinear observers to predict the trajectory, position, orientation as well as control parameters [35]. For a humanoid robot, sometimes foot contact information, where contact point velocity is assumed to be zero, is incorporated into the EKF state estimation as well [5, 10]. However, due to the linearization through Taylor approximation in EKF, the

filter is sometimes inconsistent and led to divergences in actual applications. Therefore, the idea of using geometrical properties of Lie group theory to represent the state was originally introduced in [6] as Invariant Extended Kalman filter (InEKF) and further explored in [7, 2, 1, 25, 3, 4]. InEKF utilizes the group affine properties in matrix Lie groups to assume the log-linear property of the nonlinear error [8], resulting in the error being independent of the previous state estimate. This improves the convergence properties in the EKF, resulting in strong local convergence of estimation. The idea is implemented on a bipedal robot along with contact information [16], the method achieves better convergence properties than another state-of-the-art variation of EKF for quaternion-based orientation system called quaternion-based EKF[37].

1.2 Objective of the Thesis

Utilizing available sensors on a bipedal robot, we will design a framework and implement a perception system for the robot. Raw sensor measurements from each sensor are processed into useful data through various means such as filtering, segmentation, and sensor fusion. The processed sensor data is then used to create a 3D map of the environment using the odometry of the robot relative to a fixed frame provided by the robot itself. Additionally, we will implement the LiDAR SLAM algorithm to perform mapping and localization in a situation where external odometry is unreliable or unavailable. Finally, we will implement a state estimation algorithm based on invariant observer design variation of Extended Kalman Filter in matrix Lie group. Then, we will evaluate the performance of state estimation using external ground truth reference.

In summary, the primary goal of the this thesis are as follows

- Design the perception system pipeline for bipedal robot
- Process and fuse raw sensor measurements into useful information for further applications
- Construct 3D scene of the environment and represent the information through probabilistic inference

- Implement SLAM algorithm for creating 3D map along with local localization in the generated map
- Implement Invariant Extended Kalman Filter (InEKF) to accurately estimate the state of the system represented in matrix Lie group

In this thesis, we demonstrated the methods of processing the raw sensor measurements obtained from the sensors equipped on the Digit bipedal robot using various techniques implemented in the Point Cloud Library (PCL) introduced in [36]. This includes the down-sampling of point cloud based on voxels, ground plane segmentation using RANSAC iterative process [11] and fusion of point cloud from various sources such as depth cameras and LiDAR. This is the fundamental process to lay the groundwork for following applications using depth-based information or specifically, point clouds.

From the pre-processed point cloud, we implemented 3D occupancy-based mapping technique, OctoMap [19] with the point cloud and odometry information of the robot relative to a world fixed frame from the default state estimator of the robot as the inputs to the mapping algorithm. The point clouds are registered in voxels along with the probabilistic occupancy value presented in the map. Using sensor measurements from Digit, we successfully reconstruct the 3D model of the indoor room, where Digit was operating in real-time.

Considering when the state estimation fails or is not available, we implemented a SLAM algorithm using a LiDAR sensor called LOAM [45] to estimate the motion and position of the bipedal robot along with constructing a 3D map of the environment with the localized position of the robot in it. This provides Digit with the spatial understanding of the environment which is crucial for further navigation purposes. The position estimate obtained from LOAM is demonstrated in an indoor experiment to have better performance and accuracy than the default state estimator from the robot itself.

Additionally, the Invariant Extended Kalman Filter is implemented on Digit to use noisy inertial measurements from IMU to propagate the system and the position of the foot contact point with the floor from the kinematics of the robot as measurements to predict and update the states of Digit over time. Although the results are shown to be not as good as

expected, the InEKF lays the groundwork for state estimation in Digit and further improves the performance of the state estimation over classical nonlinear filtering methods that is Extended Kalman Filter.

1.3 Organization of the Thesis

Chapter 2 presents the bipedal robot used in this thesis along with its sensor components and configurations. Then, we introduce point cloud processing methods to remove noise or unwanted cloud, extract obstacles or objects from a point cloud scene and fuse the point cloud data from multiple sensors sources.

Chapter 3 presents the key principle of the OctoMap mapping algorithm which is used to generate a probabilistic occupancy map of the environment from point cloud input. We demonstrate the implementation on the bipedal robot as well as the applications of reduced dimension map output from the algorithm. Additionally, we discuss the core idea of the LiDAR Odometry and Mapping (LOAM) method as well as the implementation on hardware.

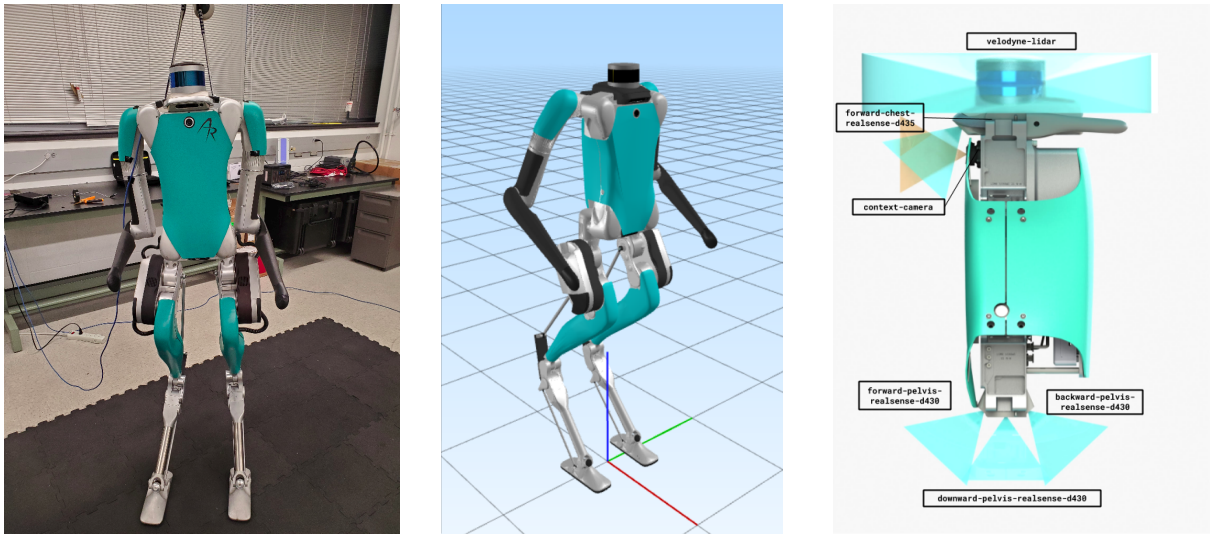
Chapter 4 introduce the formulation of Extended Kalman Filter and how we can incorporate Lie group theory into the filter, exploiting group affine properties to formulate Invariant Extended Kalman filter with invariant observer design and improved convergence property. Then, we implement InEKF filter on the bipedal robot and evaluate the accuracy of state estimation through hardware experiment provided with ground truth reference.

Chapter 5 concludes the core idea of the thesis, highlights important remarks, and discusses the future work of this thesis.

CHAPTER 2

SENSORS AND DATA MANIPULATION

In this chapter, we introduce the bipedal robot used in this research called Digit developed by "Agility Robotics" company along with the sensors equipped on the robot. The sensor measurements were processed with various methods to remove noise and unwanted data before being used in applications such as the reconstruction of the 3D environment and Simultaneous Localization and Mapping (SLAM).



(a) Robot hardware (b) Digit in simulation (c) Perception sensors

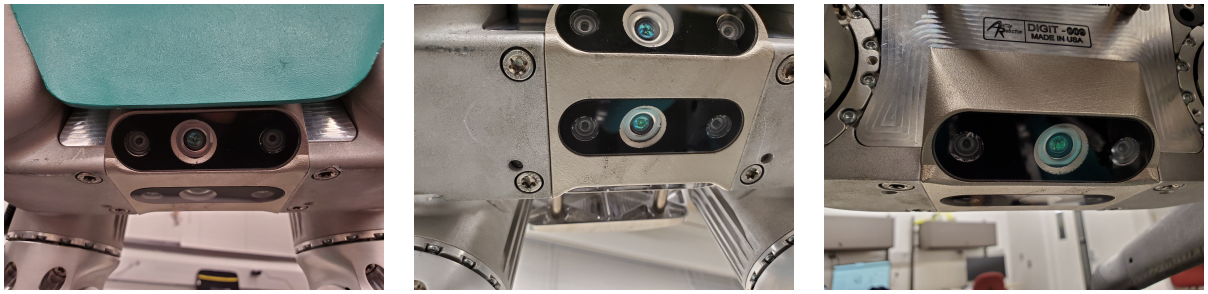
Figure 2.1: Digit bipedal robot developed by "Agility Robotic"

2.1 Sensor components

Digit is a bipedal robot developed by"Agility robotics" (see Figure 2.1) to its successful predecessor from the same company, Cassie. It featured added upper torso, arms, and sensors to accommodate higher mobility and capability requirements for more complex tasks. Digit is equipped with 3 depth cameras (Intel RealSense D430) at the pelvis, an RGB-D camera (Intel RealSense D435) at the upper chest, a LiDAR (Velodyne VLP-16 or Puck) on top of the torso, and an RGB camera at the chest. The position of sensors on the robot is depicted in Figure 2.1c. All conventions described in this thesis will follow REP-0103 conventions, in which axis orientations are x-forward, y-left, and z-upward, and units are in SI unit unless specified otherwise.

2.1.1 Depth cameras

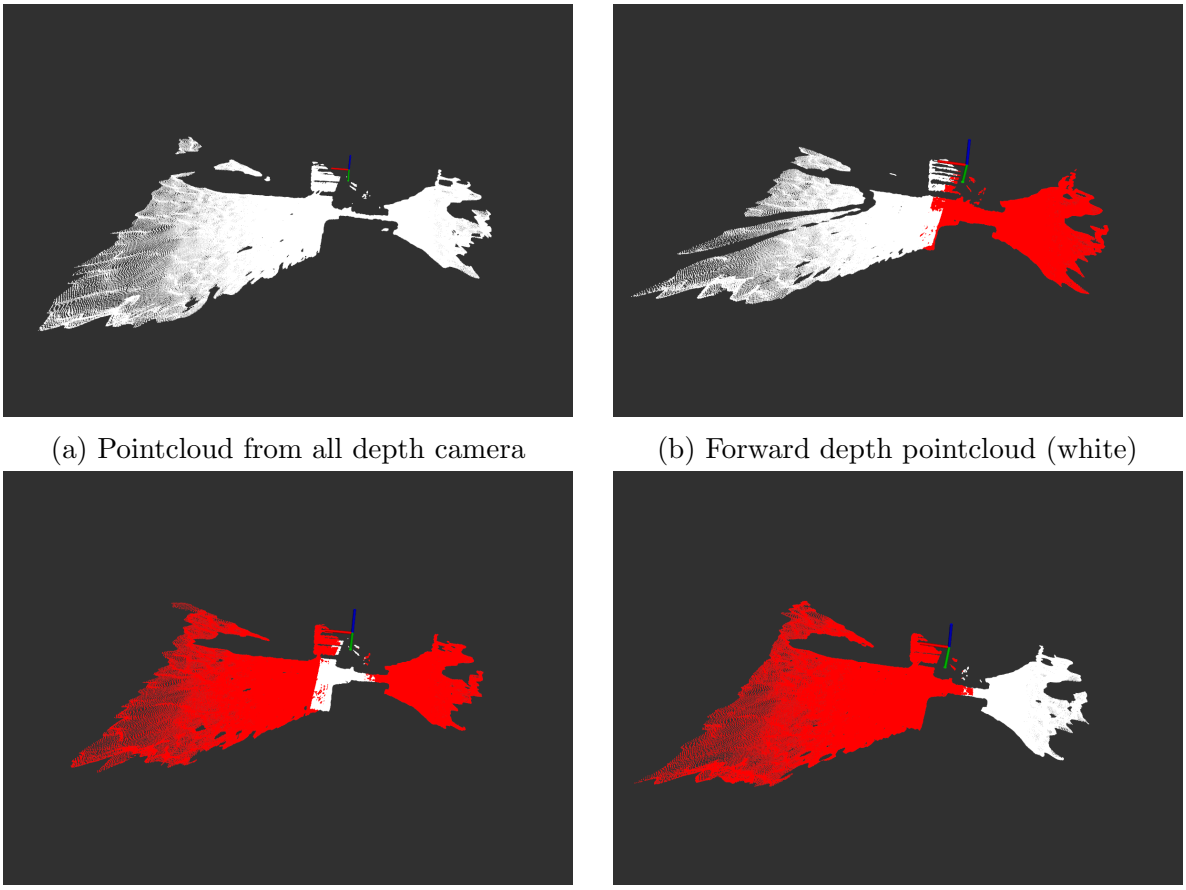
Three Intel RealSense D430 depth cameras installed at the pelvis (see Figure 2.1c) are used to monitor the surrounding ground for obstacles and identify the stepping clearance for the robot. Each camera has the field of view of $85° \times 58°$ in horizontal and vertical axis respectively. The pitch angle of each depth camera on the pelvis are $45°, 90°$ and $135°$ respectively as shown in Figure 2.2. They capture most of the ground area in front, under, and behind the robot (see Figure 2.3). However, there are blind angles on the side of each leg lowering the safety in lateral movement. The reliable range for each depth camera is up to 2-3 meters and the accuracy will drop significantly for objects detected at 3 meters or



(a) Forward camera (b) Downward camera (c) Backward camera

Figure 2.2: Depth cameras at the pelvis

further. Therefore, the acceptable perceivable range from the pelvis is less than 3 meters. These cameras provide data in form of dense point clouds.



(a) Pointcloud from all depth camera

(b) Forward depth pointcloud (white)

(c) Downward depth pointcloud (white)

(d) Backward depth pointcloud (white)

Figure 2.3: Point cloud obtained from depth cameras

2.1.2 3D LiDAR sensor

Velodyne VLP-16 or Puck LiDAR sensor is attached on the top of Digit's torso (see Figure 2.4a). With 16 laser channels and 360° horizontal field of view, this sensor can capture accurate distance to objects up to 100 meters. This gives the robot a general idea of the

spatial features of the environment that the robot is operating in and provide Digit the ability to detect large objects, walls, or human around itself. For the vertical field of view, the sensor has $(-15°, +15°)$ field of view (FOV), which results in a blind angle region near the robot, but this issue is subsidized by the presence of depth cameras mentioned earlier. The data structure of the sensor is in the form of rings of point clouds, each associated with the channels of laser (see Figure 2.5).

(a) LiDAR sensor (b) Visual cameras

Figure 2.4: LiDAR sensor and visual cameras installed on the torso of Digit.

2.1.3 Visual camera

Digit has 2 visual cameras which are located at the chest section of the torso (see Figure 2.4b). Intel Realsense D435 RGB-D camera is installed at a tilted pitch angle (facing slightly downward as seen in Figure 2.1c) to capture the scenes of space in front of the robot, resembling how people look at the floor while walking. The field of view for this camera is $85° \times 58°$ in horizontal and vertical axis respectively The camera also provides depth information, however, the field of view is mostly overlapped with the depth camera at the pelvis and since the RGB-D camera is located higher than the depth camera, the depth

information of the detected objects or ground is more prone to error. The only advantage of the depth information from the RGB-D camera is that it can cover some area within the blind angle region above the depth camera. The second camera is a monocular RGB camera (TIS DFM 27UP) located on the chest and facing forward. The RGB camera has a wider field of view to monitor the scenes directly in front of the robot. With a wider field of view, the image is slightly distorted which needed to be accounted for before using in any further application.

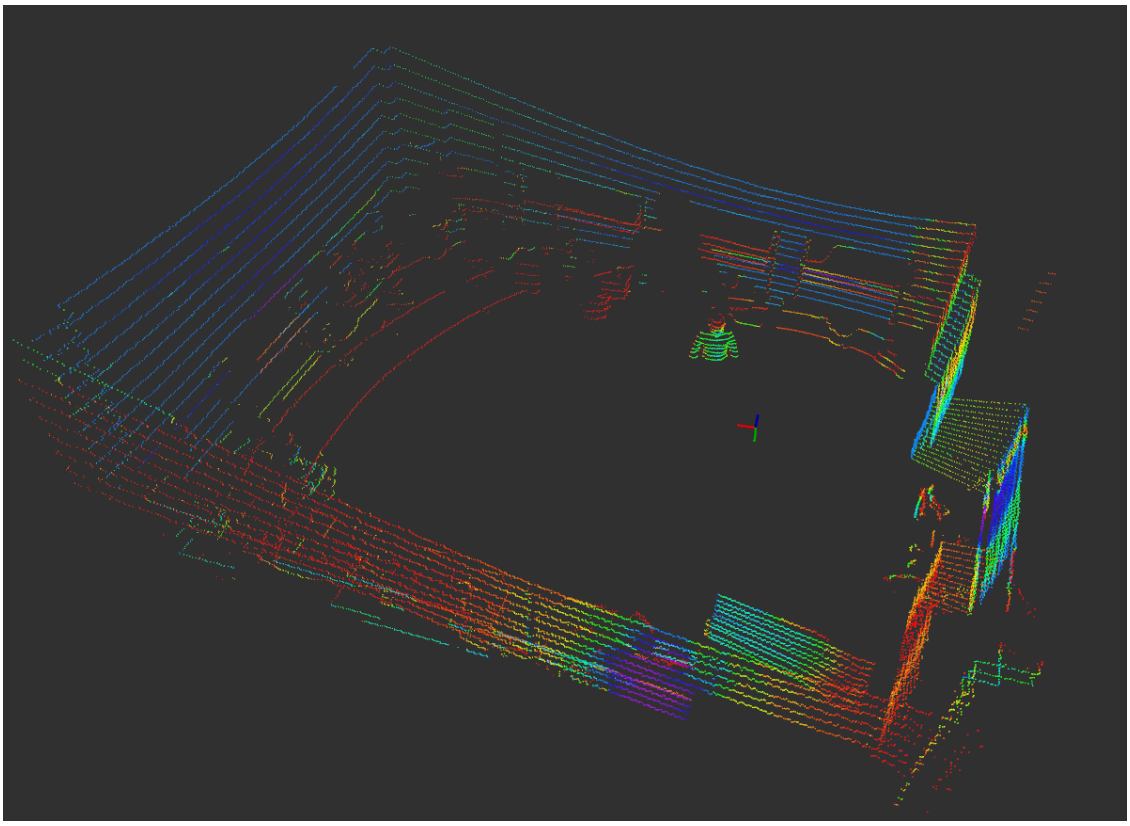


Figure 2.5: Point cloud from LiDAR sensor

2.1.4 Inertial measurement unit

Digit has a built-in 9-axis inertial measurement unit (IMU) located inside the torso and coincides with the position of the base link of the robot. The orientation of the IMU is $-90°$ rotated in pitch angle from the default orientation of the base link of the robot. To clarify, the x-axis of the IMU is pointing upward and the z-axis is pointing backward relative to the robot. This sensor provides linear acceleration and angular velocity information along with its orientation which is calculated from the measurement of a magnetometer.

2.2 Point cloud manipulation

Raw sensor measurements are usually noisy and not appropriate to be used directly. Sometimes sensors can capture unwanted data or redundant information. Therefore, we have to pre-process the sensor measurement before using them in mapping, localization, or state estimation.

2.2.1 Voxel filtering

The point cloud we obtained from the depth cameras is very dense making further processes computationally costly which is not feasible for real-time operations. Therefore, the measurements from the depth camera need to be downsampled using VoxelGrid filter [36].

A voxel is a 3D box in the space, each containing a number of point clouds. We downsample them by calculating the spatial average or centroid of the point clouds within the same voxel, then we can use that centroid as the representative of all points within that voxel. This method may be slower than directly using the center of each voxel as the approximation, but it can represent the underlying surface much more accurately. The number of the output point cloud from voxel grid filtering depends entirely on the size of voxel or leaf size. Smaller leaf size gives a larger number of output point clouds and retain more information from the original point cloud. Therefore, using optimal leaf size is crucial for the optimal trade-off between computational cost and the amount of information retained.

We used a leaf size of 0.02 meter per voxel dimension which retained all the crucial information of the object from the scan and from 10 sample scans the average point cloud

Table 2.1: Number of points in point cloud before and after voxel grid filtering

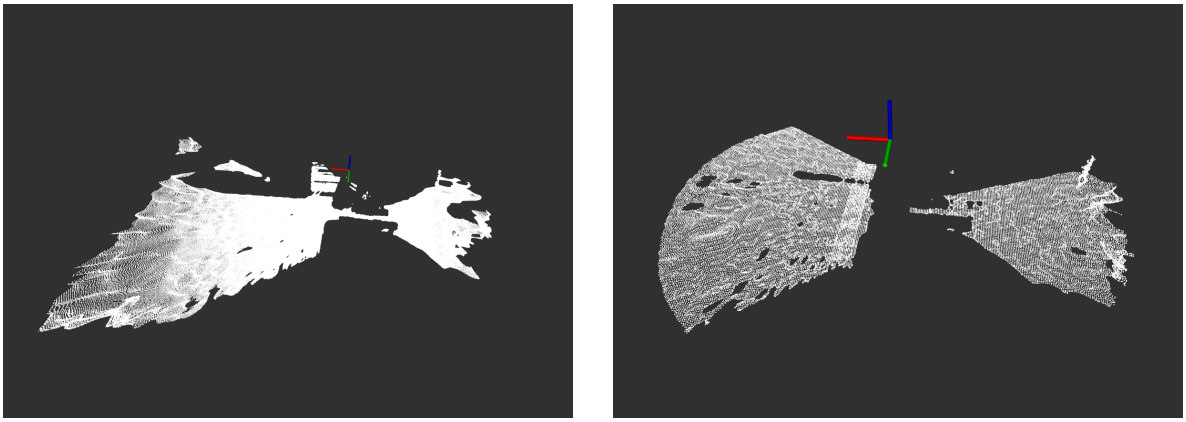
Sample Scan	Input cloud size (points)	Output cloud size (points)	Reduction (%)
1	640371	49899	92.21
2	541798	38285	92.93
3	527150	42283	91.98
4	570801	58701	89.72
5	639343	56623	91.14
6	567481	55318	90.25
7	623727	57544	90.77
8	578479	55319	90.44
9	627272	58780	90.63
10	626255	58967	90.58
Average			91.07

reduction is at 91.07% as shown in Table 2.1 which significantly reduce the computation load in each scan.

In addition to the dense point cloud, the point clouds further from the depth camera are also noisy and sometimes outlier points showed up as well, so we have to filter out unwanted point clouds from the measurement to reduce the error of the data. We filtered out point clouds that are 2.7 meters or further from the depth camera to maintain the balance between accuracy and detection range of the camera itself. Due to the reflection of some objects, the depth camera will sometimes capture points that are underground or stray points that are not within the range of the sensor. These outliers are removed with pass-through filters, which will filter out unwanted points according to specified conditions. The final result of filtering is shown in Figure 2.6.

2.2.2 Ground segmentation

Point clouds we obtained from the depth camera can help Digit identify where the available stepping region are or where the obstacles are located. However, the system cannot directly extract that information from the given point clouds. Therefore, we need to differentiate the region that allows stepping as ground cloud and the obstacles region as obstacle cloud. This process is done after fusing the filtered point cloud from all depth cameras by transforming



(a) Raw point cloud data (b) Filtered point cloud data

Figure 2.6: Filtered point cloud

the reference frame of each point cloud to base link frame coordinate using transformation given by `tf` topic in Robot Operating System(ROS) platform and then, concatenate all point clouds together.

After fusing the point clouds, we perform ground cloud segmentation by using an iterative process algorithm called Random Sampling and Consensus (RANSAC) proposed in [11]. This algorithm is used to separate outliers from inliers within a group of data points. The overview of the process is shown in Algorithm 1. In our case, it is used to extract the best plane from 3D point clouds (P). The algorithm will randomly select 3 points to construct a geometrical plane model in the following form

$$ax + by + cz + d = 0. \tag{2.1}$$

Then, the absolute distance (D_i) from each point in the 3D point clouds (P) to the plane is calculated using the following equation

$$D_i = \left| \frac{ax_i + by_i + cz_i + d}{\sqrt{a^2 + b^2 + c^2}} \right|, \text{ for } i \in \{1, 2, .., n\}, \tag{2.2}$$

where n is the total number of points in the point cloud (P).

Next, the algorithm will detect all the points $p_i \in P$ that has absolute distance to plane within a certain distance threshold and considered them as inliers, denoted as P_I:

$$P_I = \{p_i \in P |\ D_i < D_{\text{threshold}}\}. \tag{2.3}$$

This process is repeated iteratively to get the plane model that fits the largest number of point or inliers in the point clouds P and also having lowest standard deviation for the distance (D_i) in case of ties. The number of iterations (N) are determined statistically as the following equation:

$$N = \text{round}\left(\frac{\log(1-\alpha)}{\log(1-(1-\epsilon)^3)}\right), \tag{2.4}$$

where

- N is the number of trials or observation needed to achieve at least one good plane,
- α is the minimum probability to find at least one good plane from the point cloud (usually lies between 0.90 and 0.99) ,
- ϵ is the probability of error allowed in each observation which is calculated by

$$\epsilon = 1 - \frac{n_{max}}{n_{total}}, \tag{2.5}$$

  where n_{max} is the maximum probable number of points in a plane and n_{total} is the total number of points in the point cloud.

The example of segmentation results of the point cloud from Digit are shown in Figure 2.7 and the RANSAC algorithm is summarized in Algorithm 1 where

- points2plane($\cdot$) is a function to generate plane model from three arbitrary points;
- distance2plane($\cdot$) is a function to calculate the distance of a point to a plane;
- standard-deviation($\cdot$) is a function to calculate the standard deviation of a point cloud.

Algorithm 1 RANSAC algorithm

$maxFit = 0,\ minStd = \infty$
$bestPlane = [0, 0, 0]$
$i = 0$
$N = round(log(1 - \alpha))/(log(1 - (1 - \epsilon)^3))$

while $i <= N$ **do**
    r = 3 randomly selected points from point cloud P
    plane = points2plane(r)
    D = distance2plane(plane,P)
    inliers = find(abs(D) < threshold)
    Std = standard-deviation(inliers)
    if (length(inliers) < maxFit) or (length(inliers) = maxFit and Std < minStd) **then**
        maxFit = length(inliers)
        minStd = Std
        bestPlane = plane
    end if
end while

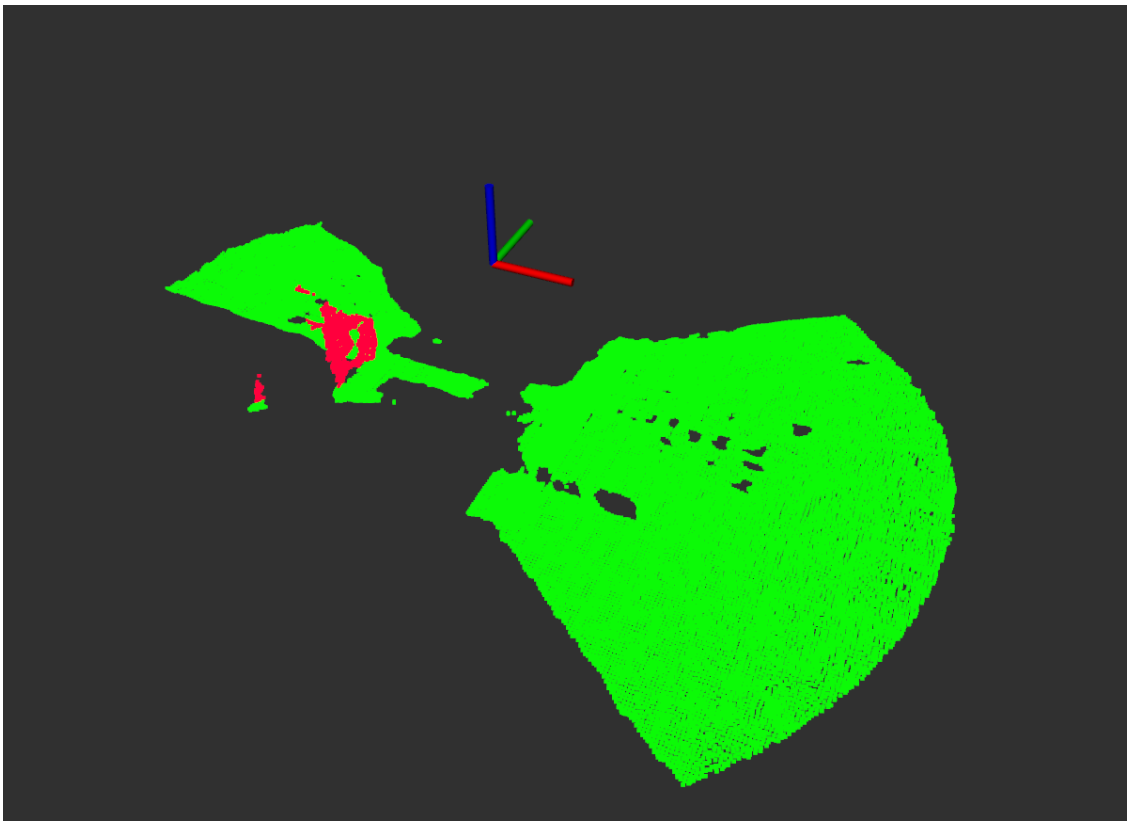


Figure 2.7: Point cloud are segmented in to ground cloud(Green) and obstacle cloud(Red)

2.2.3 Point cloud fusion

To utilize all the sensors' point cloud measurement, we will fuse the point cloud from depth cameras and LiDAR sensors together by transforming the point cloud from each sensor frame to a single reference frame on the robot (see Figure 2.8). Since LiDAR and depth cameras are all attached to the torso of the robot which is considered a single rigid body, we can get a static homogeneous transformation matrix between each sensor and the robot from the information given by internal `tf` topic in ROS provided by Digit itself. Then, the transformation we obtained is used to transform the point cloud from each sensor to the base link frame of the robot which is located at the pelvis of Digit.

$$\begin{bmatrix} \mathbf{p_b} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R_{bc}} & \mathbf{t_{bc}} \\ \mathbf{0^T} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p_c} \\ 1 \end{bmatrix}, \tag{2.6}$$

where

- $\mathbf{p_b}$ is the position vector of a point in base link reference frame,
- $\mathbf{p_c}$ is the position vector of a point in camera or sensor reference frame,
- $\mathbf{R_{bc}}$ is the static rotation matrix from base link orientation to camera/sensor orientation obtained from the kinematics of the robot,
- $\mathbf{t_{bc}}$ is the static translation vector from base link orientation to camera/sensor orientation obtained from the kinematics of the robot.

Since the there is no intersection area of point cloud between depth cameras and LiDAR, we can directly concatenate the transformed point cloud together to get the combined point cloud measurement from all available depth perception sensor.

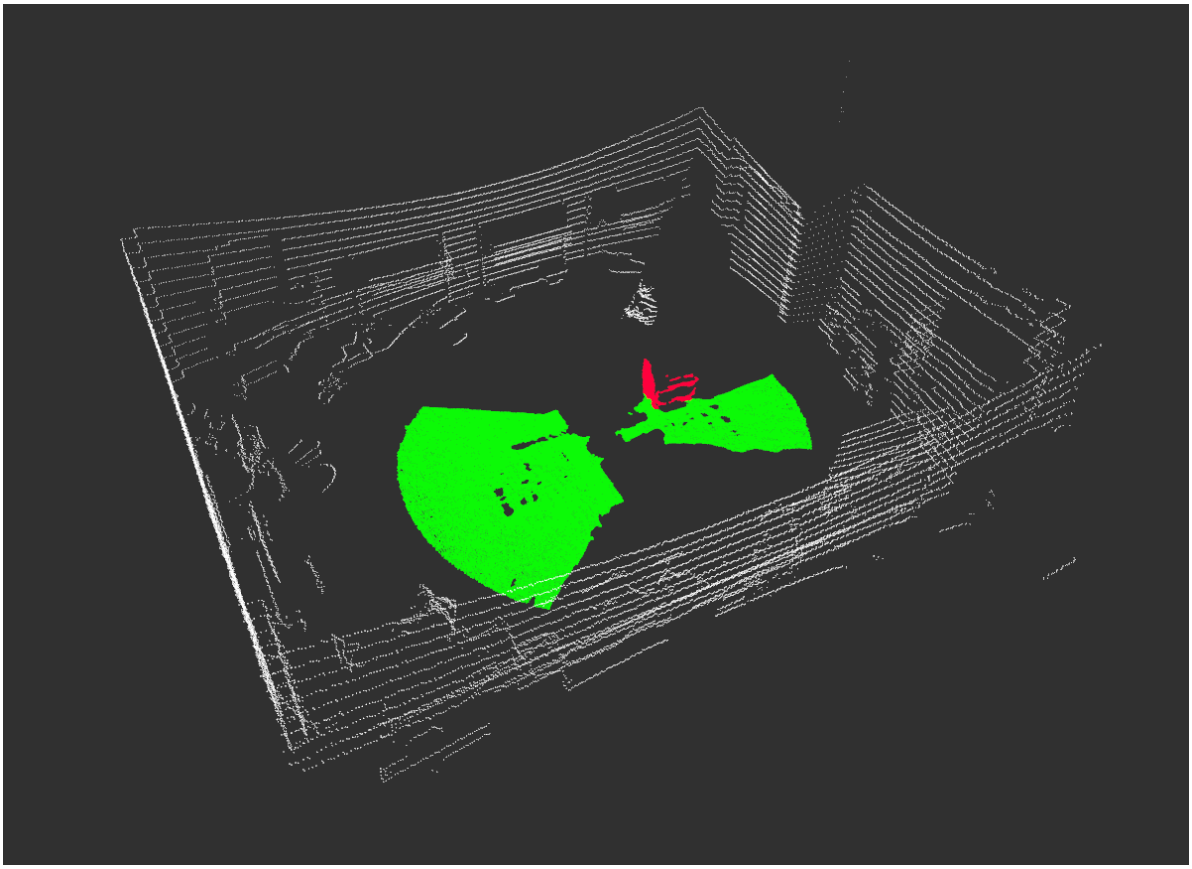


Figure 2.8: Fused point cloud from LiDAR (White), obstacle cloud (Red) and Ground cloud (Green)

2.3 Summary

We introduce the sensor components on Digit and the types of information we can get from the sensors. Then, we presented various methods of point cloud manipulation to process depth-related sensor data such as downsampling the point cloud to reduce the computational load and memory usage, filtering, and segmentation to fix noisy measurements and remove unwanted point cloud that may cause an error in further applications. Then, we use the kinematic information of the robot to transform and fuse the measurement from each sensor together. In the next chapter, we will introduce the applications of point cloud such as 3D mapping and Simultaneous localization and mapping (SLAM).

CHAPTER 3

MAPPING AND LOCALIZATION

Individually, the point cloud measurement from each scan is not very meaningful in practical application. To get a better understanding of the spatial layout of the environment, this chapter introduces the method to reconstruct and visualize the 3D structure of the objects and landscape in the environment. For this purpose, we used OctoMap, which is a framework to construct 3D models using octrees and probabilistic occupancy estimation detailed in [19].

3.1 3D scene reconstruction using OctoMap

3.1.1 Octree

Octrees are the main data structure used to partition 3D space in OctoMap framework. They consist of nodes in a hierarchical tree. Each node represents a cubic volume called voxels and the size of the voxel is called leaf size. The voxels are recursively divided into 8 sub-volumes until the smallest voxel reaches the specified minimum leaf size which is the resolution of the octree (see Figure 3.1).

Octree uses Boolean values to represent the occupancy of a volume or voxel. Each node has discrete values assigned to them which are either occupied, free or unknown. This can be used to identify free space that the robot can operate in which will be beneficial for path planning and trajectory optimization later on. When a voxel is identified as occupied, the corresponding node in the octree is initialized as occupied. To differentiate free voxel and unknown voxel, the ray-casting technique is used. The area between the sensor and measured endpoint is identified as free space and initialized as such. All other uninitialized voxels are

considered as unknown space. Additionally, if all the children nodes of a parent node have the same state, that node can be pruned to reduce the computational cost. However, for the robots operating in a highly dynamic environment, the discrete values alone will not be sufficient to cope with the changes in the environment. Therefore, OctoMap introduced an approach to model probability into the occupancy consideration which will be explained in the following section.

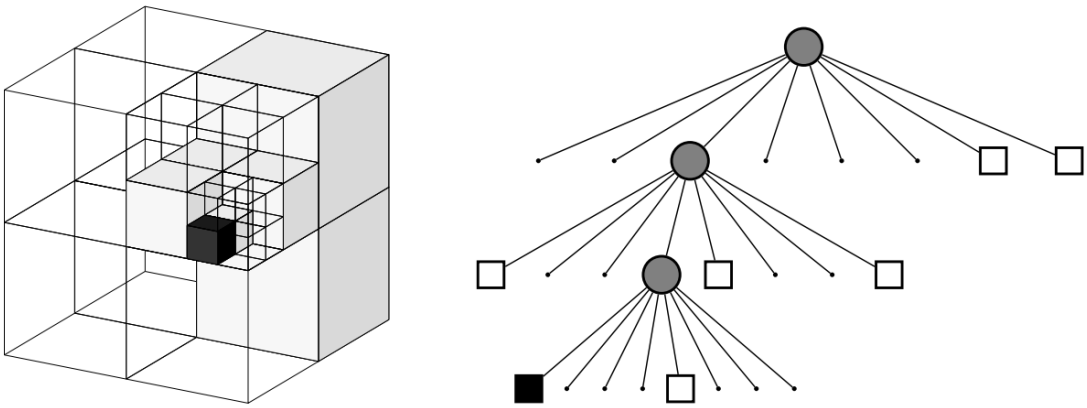


Figure 3.1: Octree data structure where each cubic volume is segmented into 8 sub-volumes and each cells will store occupancy values [19]

.

3.1.2 Probabilistic Sensor Fusion

For every time step that the robot receives point cloud measurements from sensors, the point cloud's reference frame is transformed to a world-fixed frame using odometry information provided by the robot's state estimation. Then, the measurements from various sources are fused into a single point cloud measurement (z_t) and are integrated into the prior occupancy nodes. Then, the probabilistic value is assigned to each node instead of discrete values. The calculation of the probability that a node is occupied is introduced in [27] where uncertainty is accounted for in the sensor measurements. The probability of a node being occupied given

all previous measurements is as follows

$$P(n|z_{1:t}) = \left[1 + \frac{1 - P(n|z_t)}{P(n|z_t)} \cdot \frac{1 - P(n|z_{1:t-1})}{P(n|z_{1:t-1})} \cdot \frac{P(n)}{1 - P(n)}\right]^{-1}, \tag{3.1}$$

where

- $P(n|z_{1:t})$ is the estimated probability that node n is occupied given all previous measurements up to time t ($z_{1:t}$),
- $P(n|z_t)$ is the estimated probability that node n is occupied given current measurement z_t,
- $P(n|z_{1:t-1})$ is the previously estimated probability that node n is occupied given measurements up to time $t-1$,
- $P(n)$ is prior probability.

All probability takes values in range [0.0, 1.0]. The prior probability $P(n)$ that a node is occupied is usually assumed to be 0.5, which means a node is equally probable that it is occupied or unoccupied. Therefore, in [19], equation (3.1) can be written in log-odds notation, resulting in update rule being in addition form, reducing the computational load.

$$L(n|z_{1:t}) = L(n|z_{1:t-1}) + L(n|z_t), \tag{3.2}$$

with $L(n) = \log\left[\frac{P(n)}{1-P(n)}\right]$. Due to computational optimality, these log-odds values ($L(n|z_{1:t})$) are stored in each node instead of the occupancy probabilistic value ($P(n|z_{1:t})$). Log-odds value can be translated back to probabilistic values as needed.

It is also common to impose a minimum threshold for occupancy probabilistic value $P(n|z_{1:t})$ for node n to be considered as occupied in order to take account for the noises from sensor measurements. Therefore, the probability that a node is occupied needs to be high enough to be initialized as occupied. This inferred that to change the state of a node with m occupied observations, it needs at least m unoccupied observations to change the state

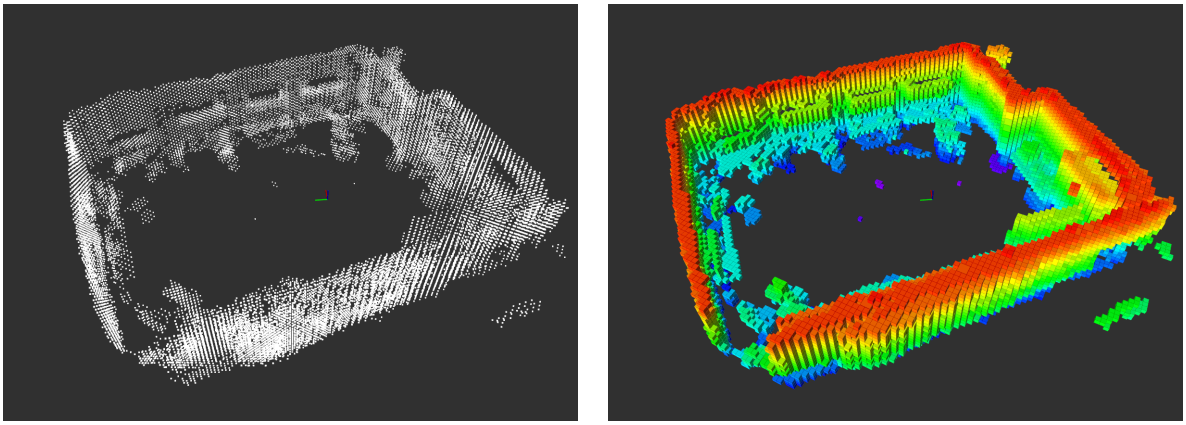
of that particular node. However, for navigation applications in a dynamic environment, we need the update and adapt to the changes in the environment as fast as possible. Therefore, [44] proposed an approach to limit the number of updates needed to change the state of the voxel. This method is called clamping update policy, in which lower and upper limit of the log-odds values are imposed, so the update equation from (3.2) became the following equation.

$$L(n|z_{1:t}) = \min(\min(L(n|z_{1:t-1}) + L(n|z_t)), l_{\max}), l_{\min}), \tag{3.3}$$

where $l_{\min}$ and $l_{\max}$ are lower and upper limit of log-odds value respectively.

The clamping update policy ensures that the value of log-odds value are bounded to a lower and upper limit. This provide the OctoMap the ability to quickly adapt to changes in the environment and then update the state of the nodes as such.

Octomap is implemented on Digit using the processed point cloud from Chapter 2 which is depicted in Figure 2.8 as input to provide a dynamic 3D map model of the environment which is updated over time when new measurements are obtained. The point cloud and voxel presentation of the output map is shown in Figure 3.2.



(a) OctoMap reconstruct 3D scene of a room within indoor environment using point cloud

(b) OctoMap represents the 3D point cloud with occupied voxels in a height map

Figure 3.2: 3D scene reconstruction using OctoMap

3.1.3 Occupancy grid map and application

In some mobile robot applications, three-dimensional information is not compulsory and it might even unnecessarily increase the computational load as well. The 3D map is sometimes projected to a lower dimension such as 2D plane, e.g. occupancy voxels are projected onto a plane (usually x-y plane), converting 3D volume into occupancy grids. At least one voxel in the same x and y coordinate has to be occupied for the occupancy grid to be considered as occupied, while all voxels have to be unoccupied for a corresponding grid to be considered free, otherwise, the grid is marked as unknown. The collection of occupancy grids are called 2D occupancy grid map (see Figure 3.3). This map is useful in grid-based path planning applications such as A* and D* along with their variations. The global path can be obtained and updated after each time step that new measurements are provided.

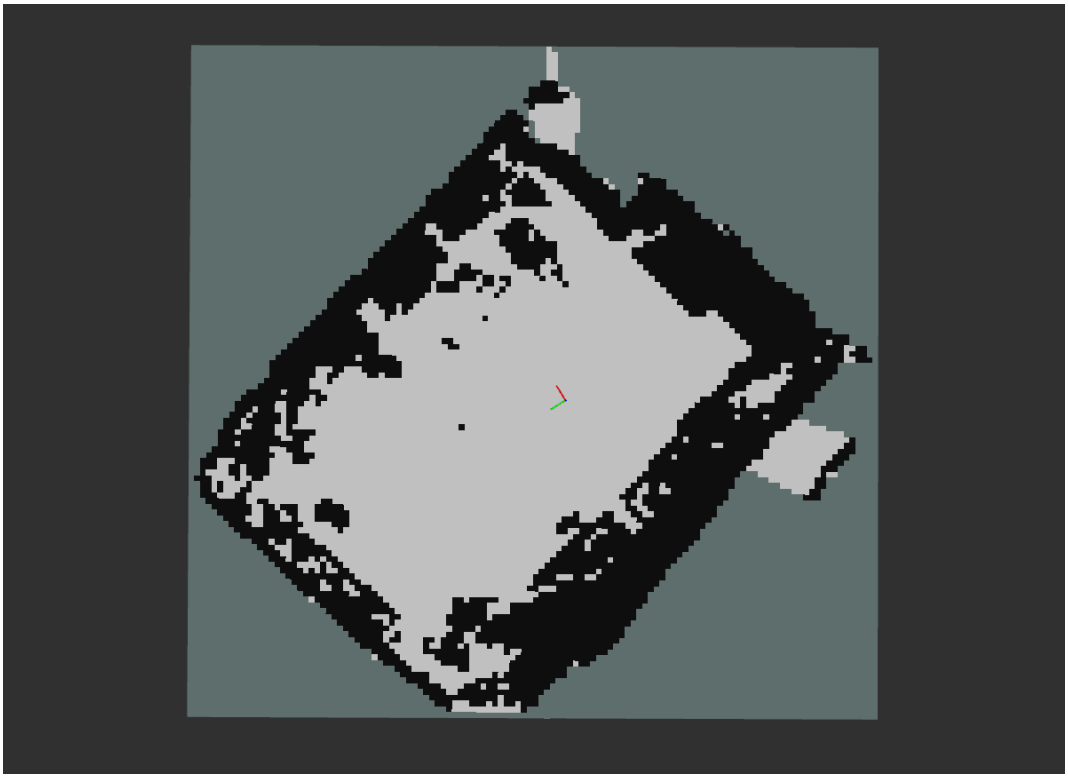


Figure 3.3: Projected 2D map with discrete grid values as occupancy grid map

However, for local path planning and motion planning processes in legged robots such as bipedal or quadruped robots, 2D occupancy grid map is not enough to capture all the information needed. Therefore, a height occupancy map or 2.5D map are often used instead. Height maps are occupancy grid maps with object height information stored in each grid in addition to the discrete occupancy values (see Figure 3.2b). This information is helpful in tasks like stepping over an obstacle or identifying the task-space or work-space of the robot.

3.2 Simultaneous localization and mapping (SLAM) using LiDAR sensor

In the previous section, we discussed the 3D environment reconstruction using OctoMap while assuming the odometry of the robot was provided from external sources such as wheel encoders and GPS/GNSS. However, if the external source of odometry is not available, the previously presented approach will not be functional.

To solve the issue, we can obtain odometry or states of the agent from sensor measurements through the Simultaneous localization and mapping (SLAM) approach. SLAM consists of the task of building a map of the environment while estimating the pose of the agent at the same time. The challenge in SLAM problems is that both mapping and localization have to be done at the same time since building a map requires the state estimate of the agent and localization needs a map for the agent to be localized in. This is initially known as the chicken-and-egg problem. SLAM is a very active field of research and there are various techniques and methods to solve this problem. There are many kinds of sensors used in SLAM problems such as visual cameras, LiDAR and RADAR, etc. Sometimes, IMU is also used to estimate the motion of the agent and reduce the drift in the state estimation, making the prediction more robust.

3.2.1 LiDAR Odometry and Mapping

Due to limitations from the low frame rate of the RGB camera relative to the movement of the bipedal robot, the camera is not a very good choice of the sensor to do SLAM since it will incur a large amount of drifting due to low-frequency updates which also affect the

ability of the robot to perform feature tracking. As for the RGB-D camera on the upper chest, the viewing angle of the camera is facing slightly downward, so the number of feature points extracted from each scene is low, consequently resulting in poor performance in feature tracking.

LiDAR can provide high frequency and wide-angle range measurements and the sensor is also insensitive to texture and the ambient lighting in the environment. The error from LiDAR measurements is also relatively constant regardless of the distance measured. Due to the aforementioned advantages, LiDAR is one of the most popular sensor choices for SLAM applications.

One of the state of the art approach called LiDAR Odometry and Mapping or LOAM is introduced by Ji Zhang in [45]. The approach suggested using two algorithms running in parallel to optimize a large number of variables simultaneously. Those algorithms are LiDAR odometry running at high frequency to estimate the velocity of the agent with relatively lower accuracy and LiDAR mapping which performs fine feature points matching and mapping at a lower frequency.

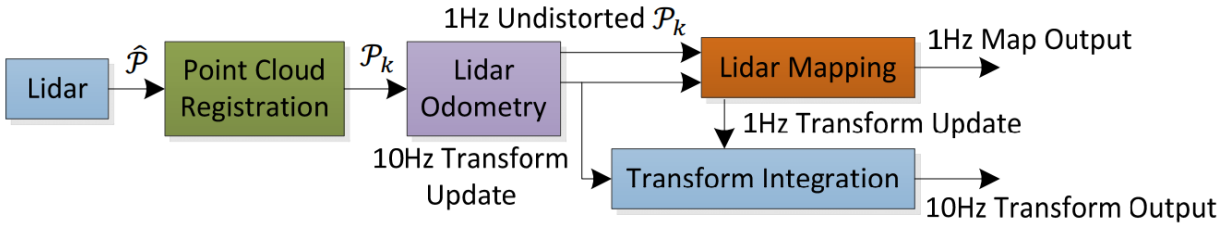


Figure 3.4: Overview of the LOAM system [45]

The system overview is shown in Figure 3.4. First, each point in point cloud from the k^{th} sweep of LiDAR ($\hat{P}$) are registered in the LiDAR reference frame (L) to get the point cloud P_k. Then, the LiDAR odometry algorithm will use the point clouds from two consecutive sweeps to estimate the pose transformation of the sensor between each sweep or time step

at the frequency of 10 Hz. The obtained transformation will be used to correct the drift in point cloud P_k. The outputs from the LiDAR odometry algorithm are further used in the LiDAR mapping algorithm, where the undistorted point cloud will have its features matched with its correspondence in the current map and the point clouds will be registered into the map accordingly. LiDAR mapping algorithm runs at a lower frequency of 1 Hz ensuring the robustness of the map output. The transformation output from both algorithms is fused together to get the average transform update of 10 Hz. The detailed procedures are explained in [45] as follows.

3.2.2 Feature point extraction

When a point cloud is obtained, LOAM extracts feature points from sharp edges and planar surface patch from the point cloud. To identify whether a point belongs to an edge or a planar surface features, the term smoothness of local surface (c) is introduced.

$$c_i = \frac{1}{|S| \cdot ||\boldsymbol{X}^L_{(k,i)}||} \cdot \sum_{j \in S, j \neq i} (\boldsymbol{X}^L_{(k,i)} - \boldsymbol{X}^L_{(k,j)}), \tag{3.4}$$

where

- c_i is the smoothness value at point P_i,
- S is the set of consecutive points returned from the same sweep as point $P_{k,i}$ in point cloud P_k,
- $\boldsymbol{X}^L_{(k,i)}$ is the coordinate of point of interest P_i in the point clouds from the k^{th} sweep which was registered in LiDAR frame L,
- $\boldsymbol{X}^L_{(k,j)}$ is the coordinate of consecutive points P_j in the point clouds from the k^{th} sweep which was registered in LiDAR frame L.

The smoothness value of each point in the point cloud (P_k) is used to sort the points from lowest c value to highest c value. Points that have the highest c values are considered edge points (E) and points with the lowest c values are planar points (H). To ensure that

the feature points are evenly distributed in the point cloud, the point cloud from each scan is split into 4 sub-regions. Each region can have at most 2 edge points and 4 planar points. Additionally, the selected feature points cannot have surrounding points that were already selected as feature points to prevent redundancy in feature selection and the selected feature point cannot be on a surface that is parallel to the laser coming out of LiDAR during measurement.

3.2.3 Feature correspondence

Next, LOAM will associate each feature point with their correspondences from previously obtained data. The point clouds obtained during each sweep are projected to the timestamp at the end of a sweep as P_k. In the next sweep $k+1$, we will extract the feature points using the methods mentioned earlier to get a set of edge points (E_{k+1}) and a set of planar points (H_{k+1}). We will project those feature points to the beginning of the sweep $k+1$ using currently estimated transform to get projected feature points $\tilde{E}_{k+1}$ and $\tilde{H}_{k+1}$. For each of the points in $\tilde{E}_{k+1}$ and $\tilde{H}_{k+1}$, we will find their closest neighbors in P_k using 3D KD-tree. For an edge point in $\tilde{E}_{k+1}$, we need to find an edge line correspondence in P_k which consist of 2 points (j, l) which are the nearest neighbor points from different scans during the k^{th} sweep. For a planar point, we need to find a surface patch consisting of 3 points (j, l, m), where two of them are closest neighbor points from the same scan and another is from the consecutive scans to the first two points with in the k^{th} sweep (see Figure 3.5).

After the correspondences are found for all the feature points ($\tilde{E}_{k+1}$, $\tilde{H}_{k+1}$), we can calculate distance from each point to their correspondence which are edge points to edge line (d_E) and planar points to planar surface patch (d_H) as follows.

$$d_E = \frac{\left|(\boldsymbol{X}^L_{(k+1,i)} - \boldsymbol{X}^L_{(k,j)}) \times (\boldsymbol{X}^L_{(k+1,i)} - \boldsymbol{X}^L_{(k,l)})\right|}{\left|\boldsymbol{X}^L_{(k,j)} - \boldsymbol{X}^L_{(k,l)}\right|}, \tag{3.5}$$

where

- $\boldsymbol{X}^L_{(k+1,i)}$ is the coordinate of interested feature point i in sweep $k+1$ expressed in L

frame,

- $\boldsymbol{X}^L_{(k,j)}, \boldsymbol{X}^L_{(k,l)}$ are the coordinate of correspondences points j, l for edge line in sweep k expressed in L frame.

$$d_H = \frac{\left| \begin{matrix} \boldsymbol{X}^L_{(k+1,i)} - \boldsymbol{X}^L_{(k,j)} \\ (\boldsymbol{X}^L_{(k,j)} - \boldsymbol{X}^L_{(k,l)}) \times (\boldsymbol{X}^L_{(k,j)} - \boldsymbol{X}^L_{(k,m)}) \end{matrix} \right|}{\left| (\boldsymbol{X}^L_{(k,j)} - \boldsymbol{X}^L_{(k,l)}) \times (\boldsymbol{X}^L_{(k,j)} - \boldsymbol{X}^L_{(k,m)}) \right|}, \tag{3.6}$$

where

- $\boldsymbol{X}^L_{(k+1,i)}$ is the coordinate of interested feature point i in sweep $k+1$ expressed in L frame,

- $\boldsymbol{X}^L_{(k,j)}, \boldsymbol{X}^L_{(k,l)}, \boldsymbol{X}^L_{(k,m)}$ are the coordinate of correspondences points j, l, m for planar patch in sweep k expressed in L frame.

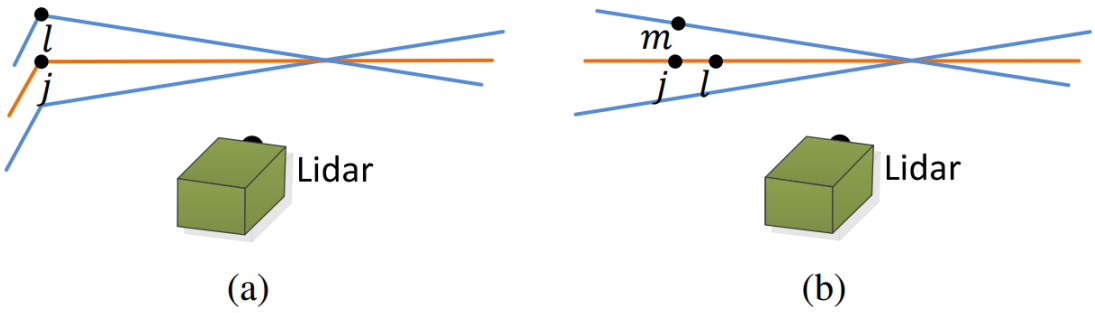


Figure 3.5: Principal to match features with their correspondences depicted in [45]

3.2.4 Motion Estimation

Motion estimate is the last process in the LOAM's LiDAR odometry algorithm where the transform $\boldsymbol{T}_{k+1}^{L}$ is constantly updated and is used to reduce the distortion of the point cloud and output the undistorted point cloud to be used in the LiDAR mapping algorithm respectively. The algorithm will estimate the motion of the LiDAR between each sweep from the measurements we obtained. First, we have to find the transform between the coordinate of the projected points $(\tilde{E}_{k+1}, \tilde{H}_{k+1})$ and the actual coordinate of the points extracted from the point cloud (E_{k+1}, H_{k+1}).The linear velocity and angular velocity are assumed to be constant during a sweep. Let the time stamp at the beginning of $k+1^{th}$ sweep be denoted as t_{k+1} and the current timestamp be denoted as t. Let $\boldsymbol{T}_{k+1}^{L}$ be the rigid transform from time t_{k+1} to current time t. This 6 DOF transform vector consists of translation and Euler angle rotation $([\boldsymbol{r}_{k+1}^{L}, \boldsymbol{\theta}_{k+1}^{L}]^T = [r_x, r_y, r_z,\ \theta_x, \theta_y, \theta_z]^T)$ of the LiDAR. Given transformation $\boldsymbol{T}_{k+1}^{L}$, due to constant velocity assumption, we can linear interpolate to get the transform of a point i from sweep $k+1$ from time t_{k+1} to the timestamp of point i (t_i) as

$$\boldsymbol{T}_{(k+1,i)}^{L} = \frac{t_i - t_{k+1}}{t - t_{k+1}} \boldsymbol{T}_{(k+1)}^{L} \ , \tag{3.7}$$

where $\boldsymbol{T}_{(k+1,i)}^{L}$ is the transformation of point i between time $[t_{k+1}, t_i]$

From transform $\boldsymbol{T}_{(k+1,i)}^{L} = [\boldsymbol{r}_{k+1,i}^{L}, \boldsymbol{\theta}_{k+1,i}^{L}]^T$, we can construct rotation matrix $\boldsymbol{R}$ using Rodrigues formula [29]:

$$\boldsymbol{R} = \boldsymbol{I} + \hat{\omega}\sin(\theta) + \hat{\omega}^2(1 - \cos(\theta) \ , \tag{3.8}$$

where

- $\boldsymbol{I} \in \mathbb{R}^{3\times3}$ is an identity matrix,
- θ is the magnitude of rotation obtained from the norm of $\boldsymbol{\theta}_{k+1,i}^{L}$ as $\theta = ||\boldsymbol{\theta}_{k+1,i}^{L}||$,
- ω is a unit vector of the rotation axis obtained from vector $\boldsymbol{\theta}_{k+1,i}^{L}$ as $\omega = \boldsymbol{\theta}_{k+1,i}^{L} / ||\boldsymbol{\theta}_{k+1,i}^{L}||$,

- $\hat{\omega}$ is a the skewed symmetric matrix of ω.

Therefore, we can get the actual coordinate of a point i ($\boldsymbol{X}^L_{(k+1,i)}$) in (E_{k+1}, H_{k+1}) from given coordinate of projected point i ($\tilde{\boldsymbol{X}}^L_{(k+1,i)}$) in $(\tilde{E}_{k+1}, \tilde{H}_{k+1})$ as

$$\boldsymbol{X}^L_{(k+1,i)} = \boldsymbol{R}\tilde{\boldsymbol{X}}^L_{(k+1,i)} + \boldsymbol{r}^L_{k+1,i} \ , \tag{3.9}$$

From (3.5) and (3.5), we calculated the distance of projected feature points in $(\tilde{E}_{k+1}, \tilde{H}_{k+1})$ to their correspondences. Combining with (3.9), we obtain geometric relationships between actual feature points $i \in E_{k+1}, H_{k+1}$ to their correspondences as

$$f_E(\boldsymbol{X}^L_{(k+1,i)}, \boldsymbol{T}^L_{k+1}) = d_E, \ i \in E_{k+1} \tag{3.10}$$

$$f_H(\boldsymbol{X}^L_{(k+1,i)}, \boldsymbol{T}^L_{k+1}) = d_H, \ i \in H_{k+1} \ , \tag{3.11}$$

Combining (3.10) and (3.11), we can get a nonlinear function as

$$f(\boldsymbol{T}^L_{k+1}) = \boldsymbol{d} \ , \tag{3.12}$$

where each row of $\boldsymbol{f}$ represents a feature point and vector $\boldsymbol{d}$ contains the corresponding distance from the feature point to its correspondence.

From (3.12), the Jacobian matrix ($\boldsymbol{J}$) with respect to $\boldsymbol{T}^L_{k+1}$ can be calculated and (3.12) is optimized by minimizing $\boldsymbol{d}$ to zero using Levenberg-Marquardt nonlinear optimization method introduced in [32] to get the transform $\boldsymbol{T}^W_k$

$$\boldsymbol{T}^L_{k+1} \leftarrow \boldsymbol{T}^L_{k+1} - (\boldsymbol{J}^T\boldsymbol{J} + \lambda \text{diag}(\boldsymbol{J}^T\boldsymbol{J}))^{-1}\boldsymbol{J}^T\boldsymbol{d} \ , \tag{3.13}$$

where λ is specified in the Levenberg-Marquardt method. Finally, the obtained transform $\boldsymbol{T}^W_k$ is used to remove the drift from the point cloud to generate undistorted point cloud P_{k+1}.

3.2.5 LiDAR mapping

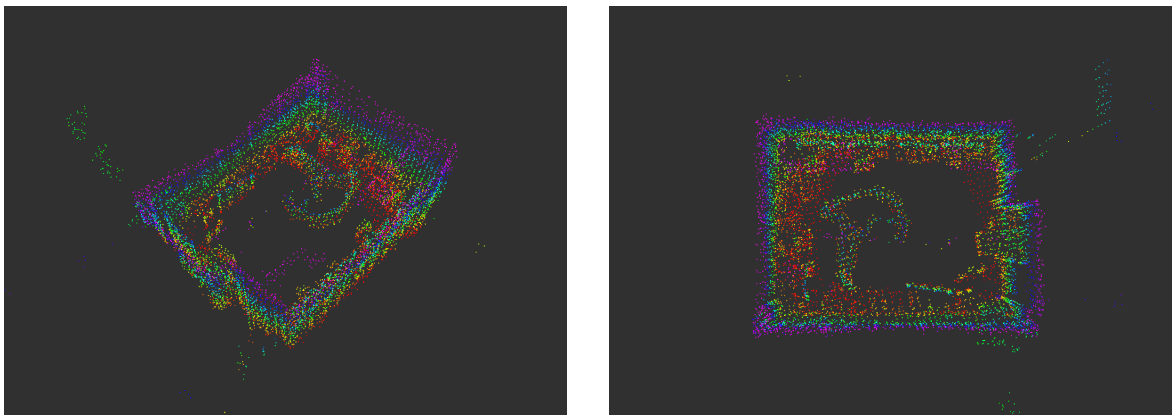
While the LiDAR odometry algorithm runs at a frequency of 10 Hz, the LiDAR mapping algorithm runs at a much lower frequency at 1 Hz, which is called only once at the end of each sweep. The motion of LiDAR from the beginning of $k+1^{th}$ sweep to the beginning of $k+1^{th}$ sweep was obtained from the LiDAR odometry algorithm as $\boldsymbol{T}_{k+1}^{L}$. This transform vector will be fused with the accumulated motion of the LiDAR up to k^{th} sweep expressed in world frame W, denoted as $\boldsymbol{T}_{k}^{W}$ to get the updated transform $\boldsymbol{T}_{k+1}^{W}$. The undistorted point cloud output from LiDAR odometry P_{k+1} is projected into the world frame W to get P_{k+1}^{W}.

Next, the algorithm will match newly obtained point cloud P_{k+1}^{W} with accumulated point cloud up to k^{th} sweep ($P_{1:k}^{W}$). The feature points are extracted with the same method mentioned earlier but the number of maximum feature points is 10 times higher. The correspondence of each feature points in the map was matched and the distance to their correspondence is calculated. The distance is minimized through the Levenberg-Marquardt optimization method again. Then, the new point cloud P_{k+1}^{W} is registered onto the map to get the map with updated point cloud $P_{1:k+1}^{W}$. Both LiDAR odometry and LiDAR mapping algorithm were called simultaneously, albeit at different frequencies, to continue keeping the motion or transformation of the LiDAR and the point cloud in the map updated in real-time.

LOAM is implemented on digit using LiDAR point cloud measurements to perform SLAM as described earlier in this section. The 3D map output of an indoor room generated ny LOAM is shown in Figure 3.6.

3.3 Summary

In this chapter, we talked about 3D reconstruction of the environment, and the 3D mapping method with the presence of an external source of odometry was introduced. We used the OctoMap mapping method to probabilistically register new point cloud measurements in the tree-like structure called octree. Other than visualization purposes, the occupancy characteristics of the volumetric cube or voxel are projected to lower dimensions such as occupancy grid maps which can be utilized in other applications like path planning.



(a) Acute view (b) Top view

Figure 3.6: LOAM constructs the 3D amp of a room within indoor environment using point cloud.

Finally, we presented the simultaneous localization and mapping method using LiDAR to generate a 3D map and localize the agent onto the map at the same time. The method used in this task is called LOAM which is one of the state of the art method that achieved relatively high performance. Using only LiDAR sensor measurement, we can estimate the odometry of the robot, build a 3D map of the environment and localize the robot in the map through two algorithms running in parallel.

In the next section, we will discuss an equally important topic in the control perspective which is state estimation of the robot through dynamic models and measurement. Although we can obtain the odometry of the robot using LiDAR SLAM, the velocity estimate is not robust because the SLAM method used in this research uses scan matching of point cloud while disregarding the effect of acceleration. Therefore, in the next section, we will introduce the use of Kalman filters and their variations which are widely used in robotic applications.

CHAPTER 4

STATE ESTIMATION

State estimation is one of the most important aspects of the robotic and autonomous behavior fields. Estimating accurate states of an agent such as its position and orientation when the agent is moving through the world is crucial for designing robust control and planning modules. Usually, robots or autonomous vehicles rely on noisy sensor measurements such as inertial measurement units (IMU), cameras, or encoders to estimate the motion and the state of the robot over time. For the purpose of estimating the state of a dynamic system, filters were introduced, of which the goal is to combine the evolution of the dynamic model over time and sensor measurements which collects partial information of the state of the agent and provides the best estimate of the state. However, the evolution model can be inaccurate in practical applications and sensor measurements are generally noisy, so these uncertainties have to be taken into the account in the estimation as well. Various types of filters were introduced in the past few decades such as Kalman filter, Particle filter, and optimization-based filter. Many filters achieve high performance and can provide robust state estimates, one such example is particle filters. However, due to high computational cost, it is not appropriate to be used in real-time operations where the computation speed needs to keep up with the changes in a dynamic environment. Therefore, the Kalman filter, which can estimate the state of the agent such as position and trajectory in real-time remains one of the most popular choices in navigation, control, and guidance applications even though the idea was first introduced in 1960.

4.1 Extended Kalman Filtering

The main idea of Kalman Filtering is to take a series of noisy measurements along with a dynamic model of the system and compute the best estimate of the state of the robot in real-time. Extended Kalman Filter is one of the most important tools in various estimation applications [40, 20]. It is a variation of Kalman Filtering, where the state evolution and observation model is nonlinear as opposed to the linear models in classical Kalman Filter.

The state of a dynamic system in discrete time is expressed as

$$\boldsymbol{X}_t = f(\boldsymbol{X}_{t-1}, u_t, w_t) \ , \tag{4.1}$$

where

- $\boldsymbol{X}_t$ is the vector containing the states of the system at time t,
- $\boldsymbol{X}_{t-1}$ is the vector containing the states of the system at previous time step $t-1$,
- $f(\cdot)$ is the function describing the evolution of system,
- u_t is the control input at time step t,
- w_t is the Gaussian noise associated with model covariance matrix $\boldsymbol{Q}_t$.

The observation or measurement is expressed as

$$\boldsymbol{Y}_t = h(\boldsymbol{X}_t) + V_t \ , \tag{4.2}$$

where

- $\boldsymbol{Y}_t$ is the vector containing measurements at time step t,
- $h(\cdot)$ is the observation function of system,
- V_t is the measurement noise.

Extended Kalman Filter (EKF) will approximate the state $\hat{\boldsymbol{X}}_{t|t}$ along with the estimated uncertainty matrix $\boldsymbol{P}_{t|t}$ given measurements up to time t, $\boldsymbol{Y}_{1:t}$. There are two steps in the EKF, which are propagation step using evolution dynamic model and update step where measurements are taking into account in the state estimate.

4.1.1 Propagation step

In this step, EKF will predict the state $\hat{\boldsymbol{X}}_{t|t-1}$ using previously estimated state $\hat{\boldsymbol{X}}_{t-1|t-1}$ which was obtained after observation $\boldsymbol{Y}_{t-1}$. The state is propagated as in (4.1) with current control input u_t to get a deterministic estimate of the state (noise parameter w_t is zero).

$$\hat{\boldsymbol{X}}_{t|t-1} = f(\hat{\boldsymbol{X}}_{t-1|t-1}, u_t, 0) \ , \tag{4.3}$$

Then, the error in estimation are defined as

$$e_{t-1|t-1} = \boldsymbol{X}_{t-1} - \hat{\boldsymbol{X}}_{t-1|t-1} \tag{4.4}$$

$$e_{t|t-1} = \boldsymbol{X}_{t} - \hat{\boldsymbol{X}}_{t|t-1} \ , \tag{4.5}$$

where

- $e_{t-1|t-1}, e_{t|t-1}$ are the estimation error given measurements up to time $t-1$ at time step t and $t-1$ respectively,
- $\boldsymbol{X}_{t-1}, \boldsymbol{X}_t$ are the true state of the system at time step $t-1$ and t respectively,
- $\boldsymbol{X}_{t-1|t-1}, \boldsymbol{X}_{t|t-1}$ are the estimated state of the system given measurements up to time $t-1$ at time step t and $t-1$ respectively.

The core idea of EKF is to linearize the error system using first-order Taylor expansion on the non-linear functions $f(\cdot)$, $h(\cdot)$ and noise parameter as stated in (4.1) and (4.2) at the

estimated state $\hat{\boldsymbol{X}}_{t-1|t-1}$ to obtain Jacobians as follows

$$\boldsymbol{F}_t = \frac{\partial f}{\partial \boldsymbol{X}}(\hat{\boldsymbol{X}}_{t-1|t-1}, u_t, 0) \tag{4.6}$$

$$\boldsymbol{H}_t = \frac{\partial h}{\partial \boldsymbol{X}}(\hat{\boldsymbol{X}}_{t-1|t-1}, u_t, 0) \tag{4.7}$$

$$\boldsymbol{G}_t = \frac{\partial f}{\partial w}(\hat{\boldsymbol{X}}_{t|t-1}) \ , \tag{4.8}$$

These Jacobians, with higher order terms ignored, are used to formulate the error dynamic model as follows

$$e_{t|t-1} = \boldsymbol{F}_t e_{t-1|t-1} + \boldsymbol{G}_t w_t \tag{4.9}$$

$$\boldsymbol{Y}_t - h(\hat{\boldsymbol{X}}_{t|t-1}) = \boldsymbol{H}_t e_{t|t-1} + V_t \ , \tag{4.10}$$

Then, we propagate the uncertainty matrix from $\boldsymbol{P}_{t-1|t-1}$ to $\boldsymbol{P}_{t|t-1}$ as

$$\boldsymbol{P}_{t|t-1} = \boldsymbol{F}_t \boldsymbol{P}_{t-1|t-1} \boldsymbol{F}_t^T + \boldsymbol{G}_t \boldsymbol{Q}_t \boldsymbol{G}_t^T \ . \tag{4.11}$$

where $\boldsymbol{Q}_t = \mathrm{cov}(w_t)$ is the covariance matrix associated to noise parameter w_t at time step t.

4.1.2 Update step

Next step is to account for the measurement $\boldsymbol{Y}_t$ obtained at time step t. We define the innovation term z_t as

$$z_t = \boldsymbol{Y}_t - h(\hat{\boldsymbol{X}}_{t|t-1}) \ . \tag{4.12}$$

We can compute Kalman gain as

$$\boldsymbol{S}_t = \boldsymbol{H}_t \boldsymbol{P}_{t|t-1} \boldsymbol{H}_t^T + \boldsymbol{R}_t \ , \tag{4.13}$$

where S_t is the innovation covariance and R_t is the covariance matrix associated with measurement noise V_n.

$$\boldsymbol{K}_t = \boldsymbol{P}_{t|t-1}\boldsymbol{H}_t^T\boldsymbol{S}_t^{-1} \ . \tag{4.14}$$

Finally, we can update the current estimated state $\hat{\boldsymbol{X}}_{t|t}$ and uncertainty matrix $\boldsymbol{P}_{t|t}$ as

$$\hat{\boldsymbol{X}}_{t|t} = \hat{\boldsymbol{X}}_{t|t-1} + \boldsymbol{K}_t z_n \tag{4.15}$$

$$\boldsymbol{P}_{t|t} = [\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{H}_t]\boldsymbol{P}_{t|t-1} \ , \tag{4.16}$$

The overview of the Extended Kalman Filter algorithm is shown as in Algorithm 2 and Figure 4.1. However, due to the linearization of the non-linear functions f and h through first-order Taylor expansion, EKF may sometimes be inconsistent and sometimes leads toward divergence in practical applications. Therefore, the idea of incorporating geometry in Kalman filter was introduced in [6] where matrix Lie groups are used to describe the states, in which we will explore the details in the following sections.

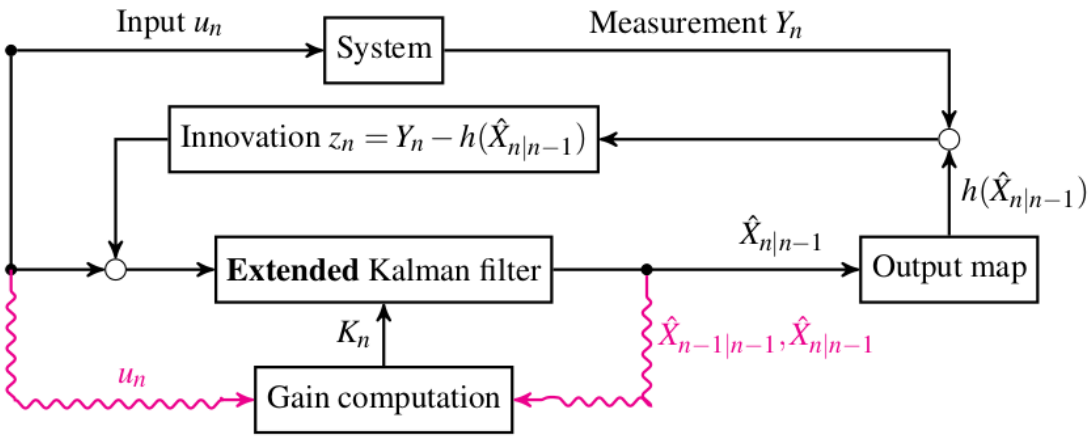


Figure 4.1: Architecture of EKF depicted in [4]

Algorithm 2 Extended Kalman Filter

Initialize state $\boldsymbol{X}_{0|0}$ and uncertainty matrix $\boldsymbol{P}_{0|0}$
while (true) **do**

Calculate Jacobians $\boldsymbol{F}_t, \boldsymbol{H}_t, \boldsymbol{G}_t$

Propagation step
$\hat{\boldsymbol{X}}_{t|t-1} = f(\hat{\boldsymbol{X}}_{t-1|t-1}, u_t, 0)$ ▷ State Propagation
$\boldsymbol{P}_{t|t-1} = \boldsymbol{F}_t \boldsymbol{P}_{t-1|t-1} \boldsymbol{F}_t^T + \boldsymbol{G}_t \boldsymbol{Q}_t \boldsymbol{G}_t^T$ ▷ Uncertainty matrix propagation

Measurements
$z_t = \boldsymbol{Y}_t - h(\hat{\boldsymbol{X}}_{t|t-1})$ ▷ Innovation
$\boldsymbol{S}_t = \boldsymbol{H}_t \boldsymbol{P}_{t|t-1} \boldsymbol{H}_t^T + \boldsymbol{R}_t$ ▷ Innovation covariances
$\boldsymbol{K}_t = \boldsymbol{P}_{t|t-1} \boldsymbol{H}_t^T \boldsymbol{S}_t^{-1}$ ▷ Kalman gain

Update step
$\hat{\boldsymbol{X}}_{t|t} = \hat{\boldsymbol{X}}_{t|t-1} + \boldsymbol{K}_t z_n$ ▷ State Update
$\boldsymbol{P}_{t|t} = [\boldsymbol{I} - \boldsymbol{K}_t \boldsymbol{H}_t] \boldsymbol{P}_{t|t-1}$ ▷ Uncertainty matrix update

end while

4.2 Lie groups theory introduction

4.2.1 Matrix Lie Groups

Lie group is a differentiable manifold which is widely used in mathematics and robotics. A matrix Lie group ($\mathcal{G}$) is a subset of square invertible matrices with the following properties

$$\begin{cases} I_N \in \mathcal{G} \\ \forall \mathcal{X} \in \mathcal{G}, \mathcal{X}^{-1} \in \mathcal{G} \\ \forall \mathcal{X}_1, \mathcal{X}_2 \in \mathcal{G}, \mathcal{X}_1 \mathcal{X}_2 \in \mathcal{G} \end{cases}, \tag{4.17}$$

where I_n is the identity matrix or $\mathbb{R}^N$ and $\mathcal{X}$ is a point on curved space $\mathcal{G}$.

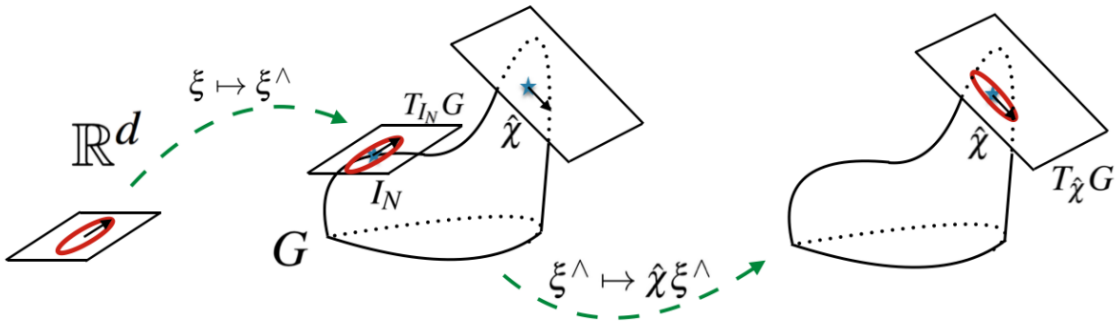


Figure 4.2: Mapping between spaces in matrix Lie groups depicted in [4]

For each point $\mathcal{X} \in \mathcal{G}$, we can associate the point to a tangent space $T_\mathcal{X}\mathcal{G}$ and the elements in this space are called tangent vectors. The tangent space at the identity of Lie group I_N is specifically called "Lie algebra" denoted as $\mathfrak{g}$ (see Figure 4.2). Lie algebra's dimension will define the dimension of the Lie group $\mathcal{G}$ itself. We will denote a vector in euclidean space as $\xi \in \mathbb{R}^d$ and the corresponding element in Lie algebra as $\xi^\mathfrak{g} \in \mathfrak{g}$. There exist an invertible linear map ($\mathcal{L}_\mathfrak{g} : \mathbb{R}^d \to \mathfrak{g}$) from euclidean space to Lie algebra or ($\mathcal{L}_\mathfrak{g}(\xi) = \xi^\mathfrak{g}$), for example skewed symmetric matrix ($\omega \to [\omega]_\times$).

There are two methods to identify $T_\mathcal{X}\mathcal{G}$ at any $\mathcal{X} \in \mathcal{G}$ for vector $\xi \in \mathbb{R}^d$ through left and right multiplication, depending on how the vector ξ is defined. For example, if the vector ξ is defined in fixed frame S as ξ_s, the corresponding tangent vector in tangent space $T_\mathcal{X}\mathcal{G}$ at $\mathcal{X}$ is $(\xi^\mathfrak{g})\mathcal{X}$. However, if the vector is defined in body frame B as ξ_b, the tangent vector on tangent space became $\mathcal{X}(\xi^\mathfrak{g})$. Although both vectors are in the same tangent space $T_\mathcal{X}\mathcal{G}$, they are different due to the non-commutative properties of matrix. This is the basis for left or right observation models.

Then, we will define the matrix exponential map from Lie algebra to Lie group ($\mathfrak{g} \to \mathcal{G}$) which is a bijection of the neighborhood of origin of Lie algebra ($\mathfrak{g}$) to the neighborhood of identity in Lie group ($\mathcal{G}$) as

$$\exp_m : \mathfrak{g} \to \mathcal{G} \ . \tag{4.18}$$

We will define Lie group exponential map from Euclidean space to Lie group ($\mathbb{R}^d \rightarrow \mathcal{G}$) which is a bijection map from the neighborhood of origin of Euclidean space to the neighborhood of $\mathcal{X}$ in Lie group as

$$\exp(\xi) = \exp_m(\mathcal{L}_{\mathfrak{g}}(\xi)) = \exp_m(\xi^{\mathfrak{g}}) \ , \tag{4.19}$$

4.2.2 Uncertainty and error definition

For random variable defined in Euclidean space ($\mathbb{R}^d$), we can represent the variable as

$$\boldsymbol{X} = \bar{\boldsymbol{X}} + \boldsymbol{e}, \ \boldsymbol{e} \sim \mathcal{N}(0, \boldsymbol{P}) \ , \tag{4.20}$$

where

- $\bar{\boldsymbol{X}}$ is the mean of random variable $\boldsymbol{X}$.
- $\boldsymbol{P}$ is the covariance of $\boldsymbol{e}$.
- $\boldsymbol{e}$ is the Gaussian noise parameters where $\boldsymbol{e} \sim \mathcal{N}(0, \boldsymbol{P})$.

However, in Lie groups, we cannot use additive noise as in Euclidean space. We define the distribution of random variable $\mathcal{X} \in \mathcal{G}$ as $\mathcal{X} \sim \mathcal{N}(\bar{\mathcal{X}}, \boldsymbol{P})$ and it is defined depending left or right multiplication by

$$\begin{matrix} \mathcal{X} = \bar{\mathcal{X}}\exp(\xi), \text{ for left multiplication} \\ \mathcal{X} = \exp(\xi)\bar{\mathcal{X}}, \text{ for right multiplication} \end{matrix} \ , \tag{4.21}$$

where

- $\bar{\mathcal{X}}$ is the mean of random variable $\mathcal{X}$,
- $\boldsymbol{P}$ is the covariance associated with error ξ,
- ξ is the Gaussian noise parameters where $\xi \sim \mathcal{N}(0, \boldsymbol{P})$.

To represent the error terms for matrix Lie group, we will use non-linear error terms: left-invariant error (η^L)and right-invariant error (η^R). For the error between states in Lie group at time t (actual state $\mathcal{X}_t$ and estimated state $\hat{\mathcal{X}}_t$), it is defined according to whether left or right multiplication are used.

$$\begin{aligned} \eta^L &= \mathcal{X}_t^{-1}\hat{\mathcal{X}}_t = (L\hat{\mathcal{X}}_t)(L\mathcal{X}_t)^{-1}, \text{ left-invariant} \\ \eta^R &= \hat{\mathcal{X}}_t\mathcal{X}_t^{-1} = (\hat{\mathcal{X}}_t L)(\mathcal{X}_t L)^{-1}, \text{ right-invariant} \end{aligned} , \tag{4.22}$$

where L is arbitrary element in Lie group ($\mathcal{G}$).

4.2.3 Important theorems for Invariant EKF

There are two key theorem which is the core of Invariant Extended Kalman Filter (InEKF).

Given a function of control input $f_{u_t}(\cdot)$, the evolution of the state $\mathcal{X}$ in Lie group can be described as

$$\frac{d}{dt}\mathcal{X}_t = f_{u_t}(\mathcal{X}_t) \ . \tag{4.23}$$

Theorem 1. According to [4], a system is group affine if the system dynamics $f_{u_t}(\cdot)$ satisfy

$$\forall t \geq 0 \quad f_{u_t}(\mathcal{X}_1\mathcal{X}_2) = f_{u_t}(\mathcal{X}_1)\mathcal{X}_2 + \mathcal{X}_1 f_{u_t}(\mathcal{X}_2) - \mathcal{X}_1 f_{u_t}(I_d)\mathcal{X}_2 \ , \tag{4.24}$$

where

- $f_{u_t}(\cdot)$ is the dynamic function of control input u_t at time t of the system,
- $\mathcal{X}_1, \mathcal{X}_2$ are elements defined in Lie group ($\mathcal{G}$),
- I_d is the identity matrix of dimension d defined in in Lie group ($\mathcal{G}$).

If the condition is satisfied, we can formulate error dynamics as

$$\begin{aligned} \frac{d}{dt}\eta_t^L &= f_{u_t}(\eta_t^L) - \eta_t^L f_{u_t}(I_d) = g_{u_t}(\eta_t^L) \\ \frac{d}{dt}\eta_t^R &= f_{u_t}(\eta_t^R) - f_{u_t}(I_d)\eta_t^R = g_{u_t}(\eta_t^R) \end{aligned} . \tag{4.25}$$

Let $A_t \in \mathbb{R}^{dxd}$ be a square matrix with same dimension as Lie algebra ($\mathfrak{g}$) and A_t satisfy the following equation;

$$g_{u_t}(\exp(\xi)) \triangleq \mathcal{L}_{\mathfrak{g}}(A_t) + \mathcal{O}(||\xi^2||) \ . \tag{4.26}$$

We will define ξ_t as the solution of the following linear differential equation

$$\frac{d}{dt}\xi_t = A_t\xi_t \ . \tag{4.27}$$

Theorem 2. Log-linear property of error described in [4] states that for initial error $\xi_0 \in \mathbb{R}^d$, if $\eta_0 = \exp(\xi_0)$, for $t \geq 0$, the non-linear error at time t can be described by

$$\eta_t = \exp(\xi_t) \ . \tag{4.28}$$

Theorem 2 suggests that the non-linear error estimation between two states at time t can be fully recovered from the solution ξ_t of (4.27). Therefore, the left-invariant and right-invariant is independent of the previous states. This property is used in the propagation step of an EKF to propagate exact covariance of the system.

4.3 Invariant Extended Kalman Filtering

Invariant Extended Kalman Filtering (InEKF) was first introduced in [6] where the system is continuous and it is further explained as discrete time system in [[3], [4]]. The state space in InEKF is represented as matrix Lie group and taking advantage of the properties presented in previous section, the filter is non-linear observer with local convergence properties.

4.3.1 System and observation models

Consider a noisy system with the following dynamic model

$$\frac{d}{dt}\mathcal{X}_t = f_{u_t}(\mathcal{X}_t) + \mathcal{X}_t w_t \ , \tag{4.29}$$

where $f_{u_t}(\cdot)$ satisfy (4.24) and $w_t \in \mathfrak{g}$ is the white noise whose covariance matrix is denoted as Q_t For the measurement, we have two kinds of observations. The first type is left-invariant observation where observations are in the following form

$$Y_t^i = \mathcal{X}_t(d^i + B^i) + V^i, \ i = 1, 2, ..., k \ , \tag{4.30}$$

where

- Y_t^i is the i^{th} measurement at time t
- d^i are known vectors that associate state space with measurement
- k is the number of measurements
- B^i, V^i are noise parameter with known characteristics

As for right-invariant observer design, we have measurements in the following forms

$$Y_t^i = \mathcal{X}_t^{-1}(d^i + B^i) + V^i, \ i = 1, 2, ..., k \ . \tag{4.31}$$

4.3.2 Error definition

From Theorem 2, the non-linear error is defined in (4.28). In the propagation step for time step $t_{n-1} \leq t \leq t_n$, according to Theorem 1, we have the noisy non-linear error model for left-invariant and right-invariant as

$$\begin{aligned} \frac{d}{dt}\eta_t^L &= g_{u_t}^L(\eta_t^L) - w_t\eta_t^L \\ \frac{d}{dt}\eta_t^R &= g_{u_t}^R(\eta_t^R) - (\hat{\mathcal{X}}_t w_t \hat{\mathcal{X}}_t^- 1)\eta_t^R \ , \end{aligned} \tag{4.32}$$

where $\hat{w}_t \in \mathbb{R}^d$ is the noise parameters in Euclidean space and we can map it to Lie algebra by $\mathcal{L}_{\mathfrak{g}}(\hat{w}_t) = -w_t$ for left-invariant error and $\mathcal{L}_{\mathfrak{g}}(\hat{w}_t) = -\hat{\mathcal{X}}_t w_t \hat{\mathcal{X}}_t^{-1}$ for right-invariant error as shown in (4.32). Then, we can linearized the error equation in $\mathbb{R}^d$ similar to (4.27) but with added noise parameter as follows

$$\frac{d}{dt}\xi_t = A^i_{u_t}\xi_t + \hat{w}_t \ , \tag{4.33}$$

where $A^i_{u_t}$ is defined as $g^i_{u_t}(\exp(\xi)) \triangleq \mathcal{L}_{\mathfrak{g}}(A^i_{u_t}) + \mathcal{O}(||\xi_t^2||)$ of which the higher order term $\mathcal{O}(||\xi_t^2||)$ is ignored.

For simplicity, we will use left-invariant EKF (LIEKF) to explain the error update procedure when some noisy measurements are obtained as follows

$$\begin{aligned}(\eta^L_{t_n})^+ &= \mathcal{X}^{-1}_{t_n}\hat{\mathcal{X}}^+_{t_n} \\ &= \eta^L_{t_n}\exp\left[L_n\left((\eta^L_{t_n})^{-1}d^i - d^i + \hat{V}^i_n + (\eta^L_{t_n})^{-1}B^i_n\right)\right],\ i = 1, 2, ..., k \ ,\end{aligned} \tag{4.34}$$

where L_n is the Kalman gain which will be calculated later.

We can linearize the innovation terms in the bracket using Taylor expansion of matrix exponential map as

$$\begin{aligned}&(\eta^L_{t_n})^{-1}d^i - d^i + \hat{V}^i_n + (\eta^L_{t_n})^{-1}B^i_n \\ &= (\eta^L_{t_n})^{-1}(d_i + B^i_n) - d^i + \hat{V}^i_n \\ &= \exp_m\left(\mathcal{L}_{\mathfrak{g}}(\xi_{t_n})\right)^{-1}(d_i + B^i_n) - d^i + \hat{V}^i_n \\ &= (I_d - \mathcal{L}_{\mathfrak{g}}(\xi_{t_n}))(d_i + B^i_n) - d^i + \hat{V}^i_n + \mathcal{O}(||\xi_{t_n}||^2) \\ &= -\mathcal{L}_{\mathfrak{g}}(\xi_{t_n})d^i + \hat{V}^i_n + B^i_n + \mathcal{O}(||\xi_{t_n}||^2) + \mathcal{O}(||\xi_{t_n}||||B^i_n||),\ i = 1, 2, ..., k \ .\end{aligned} \tag{4.35}$$

Similarly, from (4.34) and (4.35), we have

$$I_d + \mathcal{L}_{\mathfrak{g}}(\xi^+_{t_n}) = Id + \mathcal{L}_{\mathfrak{g}}(Ln(-\mathcal{L}_{\mathfrak{g}}(\xi_{t_n})d^i) + \hat{V}^i_n + B^i_n) + T,\ i = 1, 2, ..., k \ , \tag{4.36}$$

where T consists of higher order terms which are neglected and thus, we have the linearized

error equation expressed in Euclidean space ($\mathbb{R}^d$) as follows

$$\xi_{t_n}^{+} = \xi_{t_n} + L_n(H\xi_{t_n} + \hat{V}_n + B_n) \ , \tag{4.37}$$

where $H\xi, \hat{V}_n, B_n$ are defined as follows

$$H\xi = \begin{pmatrix} -\mathcal{L}_{\mathfrak{g}}(\xi_{t_n})d^1 \\ ... \\ -\mathcal{L}_{\mathfrak{g}}(\xi_{t_n})d^k \end{pmatrix} \tag{4.38}$$

$$\hat{V}_n = \begin{pmatrix} \hat{V}_n^1 \\ ... \\ \hat{V}_n^k \end{pmatrix} \tag{4.39}$$

$$B_n = \begin{pmatrix} B_n^1 \\ ... \\ B_n^k \end{pmatrix} \ . \tag{4.40}$$

4.3.3 Uncertainty evaluation

Let $\hat{Q}_t$ be the covariance of noise parameter $\hat{w}_t$ and let $\hat{N}_n$ be the covariance of the modified noise parameters $\hat{V}_n + B_n$. According to Kalman filter design, we have state dynamic as $\frac{d}{dt}\mathcal{X}_t = A_{u_t}\mathcal{X}_t + \hat{w}_t$ and discrete measurement model $Y_n = H\mathcal{X}_{t_n} + \hat{V}_n + B_n$. We have the dynamic model of uncertainty matrix P_t as

$$\frac{d}{dt}P_t = A_{u_t}P_t + P_tA_{u_t}^T + \hat{Q}_t \ . \tag{4.41}$$

Then, we can calculate the Kalman gain (L_n) as follows

$$\begin{aligned} S_n &= H P_{t_n} H^T + \hat{N}_n \\ L_n &= P_{t_n} H^T S^{-1} \ . \end{aligned} \tag{4.42}$$

Finally, the uncertainty update model can be formulated as

$$P_{t_n}^+ = (I - L_n H) P_{t_n} \ . \tag{4.43}$$

4.3.4 Summary

To summarize, consider the propagation step of a noisy system. The dynamic model of state and uncertainty matrix are defined as follows

$$\begin{aligned} \frac{d}{dt}\mathcal{X}_t &= f_{u_t}(\mathcal{X}_t) + \mathcal{X}_t w_t \\ &= A_{u_t}\mathcal{X}_t + \hat{w}_t \\ \frac{d}{dt}P_t &= A_{u_t} P_t + P_t A_{u_t}^T + \hat{Q}_t \ . \end{aligned} \tag{4.44}$$

Next, we can formulate dynamic model of non-linear error similar to (4.32)

$$\begin{aligned} &\frac{d}{dt}\eta_t = g_{u_t}(\eta_t) + \mathcal{L}_{\mathfrak{g}}(\hat{w}_t)\eta_t \\ &\frac{d}{dt}\eta_t^L = g_{u_t}^L(\eta_t^L) - w_t\eta_t^L \qquad \text{for left-invariant} \\ &\frac{d}{dt}\eta_t^R = g_{u_t}^R(\eta_t^R) - (\hat{\mathcal{X}}_t w_t \hat{\mathcal{X}}_t^{-} 1)\eta_t^R \quad \text{for right-invariant} \ . \end{aligned} \tag{4.45}$$

According to (4.26), we can linearize the error to get the dynamic model of the linearized error as shown in (4.33) : $\frac{d}{dt}\xi_t = A_{u_t}^i \xi_t + \hat{w}_t$.

Then, we obtain some noisy measurements

$$\begin{aligned} &Y_t^i = \mathcal{X}_t(d^i + B^i) + V^i, \ i = 1, 2, ..., k \quad \text{for left-invariant} \\ &Y_t^i = \mathcal{X}_t^{-1}(d^i + B^i) + V^i, \ i = 1, 2, ..., k \quad \text{for right-invariant} \ . \end{aligned} \tag{4.46}$$

The error update model are formulated as follows

$$\begin{aligned}
(\eta_{t_n}^L)^+ &= \mathcal{X}_{t_n}^{-1}\hat{\mathcal{X}}_{t_n}^+ \text{ for left-invariant} \\
&= \eta_{t_n}^L \exp\left[L_n\left((\eta_{t_n}^L)^{-1}d^i - d^i + \hat{V}_n^i + (\eta_{t_n}^L)^{-1}B_n^i\right)\right],\ i = 1, 2, ..., k \\
(\eta_{t_n}^R)^+ &= \hat{\mathcal{X}}_{t_n}^+\mathcal{X}_{t_n}^{-1} \text{ for right-invariant} \\
&= \exp\left[L_n\left((\eta_{t_n}^R)d^i - d^i + \hat{V}_n^i + (\eta_{t_n}^R)B_n^i\right)\right]\eta_{t_n}^R,\ i = 1, 2, ..., k\ .
\end{aligned} \tag{4.47}$$

From (4.47), we can linearize the non-linear error update using Taylor expansion of the exponential map as in (4.37) : $\xi_{t_n}^+ = \xi_{t_n} + L_n(H\xi_{t_n} + \hat{V}_n + B_n)$ where $H\xi_{t_n}, \hat{V}_n$ and B_n are defined in (4.38), (4.39) and (4.40) respectively.

We can calculate Kalman gain L_n as in (4.42)

$$\begin{aligned}
S_n &= HP_{t_n}H^T + \hat{N}_n \\
L_n &= P_{t_n}H^TS^{-1}\ .
\end{aligned} \tag{4.48}$$

Finally, in the update step of the filter, we have state and uncertainty matrix update model as follows

$$\begin{aligned}
(\mathcal{X}_{t_n})^+ &= \hat{\mathcal{X}}_{t_n}\exp\left[L_n\left((\mathcal{X}_{t_n})^{-1}d^i - d^i + \hat{V}_n^i + (\mathcal{X}_{t_n})^{-1}B_n^i\right)\right],\ i = 1, 2, ..., k \text{ (LIEKF)} \\
(\mathcal{X}_{t_n})^+ &= \exp\left[L_n\left((\mathcal{X}_{t_n})d^i - d^i + \hat{V}_n^i + (\mathcal{X}_{t_n})B_n^i\right)\right]\hat{\mathcal{X}}_{t_n} \quad ,i = 1, 2, ..., k \text{ (RIEKF)} \\
P_{t_n}^+ &= (I - L_nH)P_{t_n}\ .
\end{aligned} \tag{4.49}$$

The overview of Invariant Kalman Filter architecture is shown in Figure 4.3.

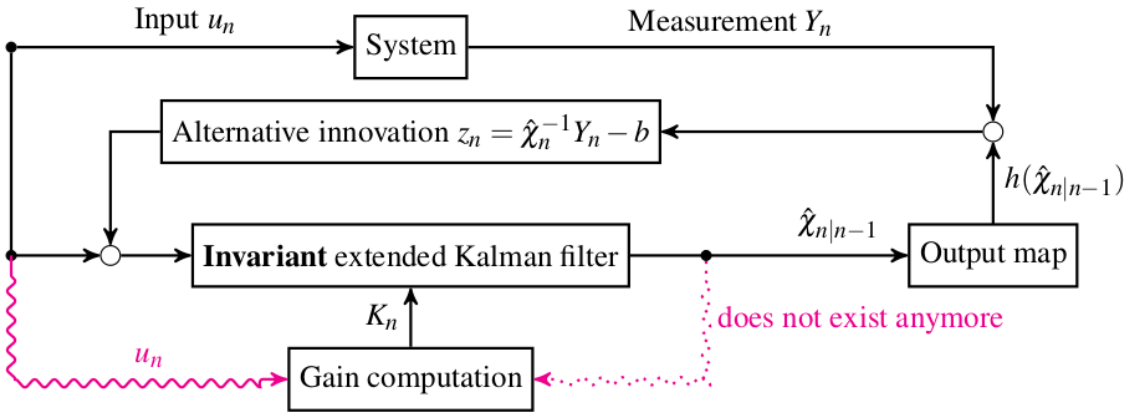


Figure 4.3: Invariant EKF architecture depicted in [4]

4.4 Contact measurement implementation of InEKF

The most common method to estimate the state of the robot with a nonlinear observer is to use EKF with IMU measurements including linear acceleration and angular velocity as a control input, propagating the system, and some sensor measurements to update the state prediction. For legged robots, the odometry of the leg joints from kinematics measurement is usually used as the measurement to the filters due to the high update frequency of the joint encoder and IMU. However, due to the linearization in EKF, the system does not have globally convergence properties and might diverge in practical implementation. Therefore, an approach utilizing the properties of right invariant observer design along with contact assumption for the bipedal robot, Cassie, was proposed in [16]. The contact point at the foot is assumed to have zero velocity but the measurement is designed to be corrupted with additive noises to account for the possible slippage. The position of the robot can be obtained from a forward kinematic calculation using the joint angles obtained from encoders. The system has strong convergence result and achieves better results than quaternion-based EKF which is commonly used in the system where orientation is expressed in quaternion. Due to similarities between Cassie and Digit, we will implement the same system to estimate the

state of Digit. The IMU frame and contact frame for Digit is roughly depicted in Figure 4.4. The problem formulation in [16] are presented as follows.

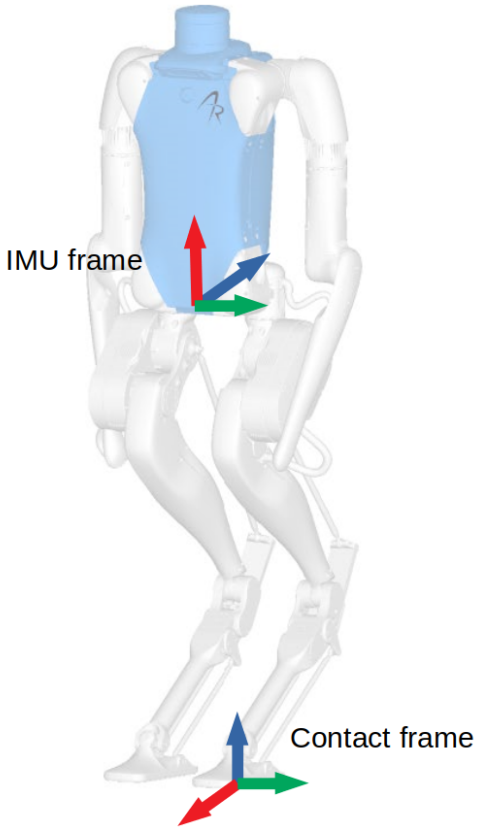


Figure 4.4: IMU frame and contact frame positions on Digit

4.4.1 State space definition

We will estimate the orientation, velocity, and position of the IMU expressed in world frame W. Additionally, the position of the current contact point is incorporated into the state as well. For simplicity, we will assume that there is only one contact point at a time indicated by binary contact measurement. For example, if the force measurement from the contact sensor for the right foot exceeds a certain threshold, Digit will switch to right support mode

indicating that the contact point at the right foot is stationary in the world frame and the position of the right foot contact point is used in the state space, vice versa. The state of the system in Lie group $\mathcal{X}_t \in \mathcal{G}$ can be represented as

$$\mathcal{X}_t = \begin{bmatrix} \boldsymbol{R}_t & \boldsymbol{v}_t & \boldsymbol{p}_t & \boldsymbol{b}_t \\ \mathbf{0}_{1,3} & 1 & 0 & 0 \\ \mathbf{0}_{1,3} & 0 & 1 & 0 \\ \mathbf{0}_{1,3} & 0 & 0 & 1 \end{bmatrix} , \tag{4.50}$$

where

- $\boldsymbol{R}_t \in SO(3)$ is the rotation matrix representing the orientation of IMU in world frame W,
- $\boldsymbol{v}_t \in \mathbb{R}^{3\times 1}$ is the velocity vector of IMU in world frame W,
- $\boldsymbol{p}_t \in \mathbb{R}^{3\times 1}$ is the position vector of IMU in world frame W,
- $\boldsymbol{b}_t \in \mathbb{R}^{3\times 1}$ is the contact point position in world frame W.

4.4.2 System dynamics

IMU measurements are treated as control input of the system and the measurements from IMU including angular velocity ($\tilde{\omega}_t$) and linear acceleration ($\tilde{\alpha}_t$) are modeled with additive Gaussian noise as follows

$$\begin{aligned} \tilde{\omega}_t &= \omega_t + w_t^g, \qquad & w_t^g &\sim \mathcal{N}(\mathbf{0}_{3,1}, \sigma^g) \\ \tilde{\alpha}_t &= \alpha_t + w_t^a, \qquad & w_t^a &\sim \mathcal{N}(\mathbf{0}_{3,1}, \sigma^a) \ , \end{aligned} \tag{4.51}$$

where

- ω_t and α_t are true angular velocity and linear acceleration respectively,
- w_t^g, w_t^a are Gaussian noise from gyroscope and accelerometer respectively,

- σ^g, σ^a are covariances associated with sensor noises from gyroscope and accelerometer respectively.

From the assumption that at contact, the measured velocity of contact point is zero but the measurement is corrupted with additive Gaussian noise, the velocity of contact point is

$$\tilde{v}_{C,t} = \mathbf{0}_{3,1} = v_{C,t} + w_t^v, \qquad w_t^v \sim \mathcal{N}(\mathbf{0}_{3,1}, \sigma^v) \ , \tag{4.52}$$

where

- $\tilde{v}_{C,t}$ is the measured velocity of contact point which is assumed to be zero,
- $v_{C,t}$ is the true velocity of the contact point,
- w_t^v is the Gaussian noise of the measurement,
- σ^v is covariances associated with slippage at contact point.

The system dynamics can be formulated using IMU strap down model and contact measurements as the following equations

$$\begin{aligned}
\frac{d}{dt}\boldsymbol{R}_t &= \boldsymbol{R}_t(\tilde{\omega}_t - w_t^g)_\times \\
\frac{d}{dt}\boldsymbol{v}_t &= \boldsymbol{R}_t(\tilde{\alpha}_t - w_t^a) + \boldsymbol{g} \\
\frac{d}{dt}\boldsymbol{p}_t &= \boldsymbol{v}_t \\
\frac{d}{dt}\boldsymbol{b}_t &= \boldsymbol{R}_t \boldsymbol{h}_R(\tilde{q}_t)(-w_t^v) \ ,
\end{aligned} \tag{4.53}$$

where

- $(\cdot)_\times$ denotes the skewed-symmetric form,
- $\boldsymbol{g}$ is the gravity vector,
- $\boldsymbol{h}_R(\tilde{q}_t)$ is the orientation of the contact frame C with respect to IMU frame which is obtained from forward kinematics calculation using joint parameters $\tilde{q}_t$.

The system in (4.53) can be written in matrix form corresponding to the state space. The system consisting of deterministic part and associated noise part. The deterministic part can be written in the form of function $f_{u_t}(\mathcal{X}_t)$ as

$$f_{u_t}(\mathcal{X}_t) = \begin{bmatrix} \boldsymbol{R}_t(\tilde{\omega}_t)_\times & \boldsymbol{R}_t\tilde{\alpha}_t + \boldsymbol{g} & \boldsymbol{v}_t & \mathbf{0}_{3,1} \\ \mathbf{0}_{1,3} & 0 & 0 & 0 \\ \mathbf{0}_{1,3} & 0 & 0 & 0 \\ \mathbf{0}_{1,3} & 0 & 0 & 0 \end{bmatrix} . \tag{4.54}$$

The function $f_{u_t}(\cdot)$ is proven to satisfy the group affine property described in (4.24), so according to Theorem 1, we can conclude that the right-invariant error dynamic can evolve in time while being independent from the system state.

Next, the noise parameters part can be represented in matrix form as

$$\mathcal{X}_t\mathcal{L}_{\mathfrak{g}}(\boldsymbol{w}_t) = \begin{bmatrix} \boldsymbol{R}_t & \boldsymbol{v}_t & \boldsymbol{p}_t & \boldsymbol{b}_t \\ \mathbf{0}_{1,3} & 1 & 0 & 0 \\ \mathbf{0}_{1,3} & 0 & 1 & 0 \\ \mathbf{0}_{1,3} & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} (w_t^g)_\times & w_t^a & \mathbf{0}_{3,1} & \boldsymbol{h}_R(\tilde{q}_t)w_t^v \\ \mathbf{0}_{1,3} & 0 & 0 & 0 \\ \mathbf{0}_{1,3} & 0 & 0 & 0 \\ \mathbf{0}_{1,3} & 0 & 0 & 0 \end{bmatrix} . \tag{4.55}$$

Finally, we have the full dynamic model of the system as

$$\frac{d}{dt}\mathcal{X}_t = f_{u_t}(\mathcal{X}_t) - \mathcal{X}_t\mathcal{L}_{\mathfrak{g}}(\hat{\boldsymbol{w}}_t) . \tag{4.56}$$

4.4.3 Error formulation

With function $f_{u_t}(\cdot)$ satisfied the group affine property, from Theorem 1, we have the dynamic model of nonlinear right-invariant error as follows

$$\begin{aligned} \frac{d}{dt}\eta_t^R &= f_{u_t}(\eta_t^R) - \eta_t^R f_{u_t}(I_d) + \hat{\mathcal{X}}_t\boldsymbol{w}_t\hat{\mathcal{X}}_t^{-1}\eta_t^R \\ &= g_{u_t}(\eta_t^R) + \mathcal{L}_{\mathfrak{g}}(\hat{\boldsymbol{w}}_t)\eta_t^R . \end{aligned} \tag{4.57}$$

From Theorem 2, if matrix A_t is defined by

$$g_{u_t}(\exp(\xi)) \triangleq \mathcal{L}_{\mathfrak{g}}(A_t\xi) + \mathcal{O}(||\xi_{t_n}||^2) \ . \tag{4.58}$$

Then, the linearized error or the log of invariant error ($\xi_t \in \mathbb{R}^d$) satisfies the following

$$\begin{aligned} \frac{d}{dt}\xi_t &= A_t\xi_t + \hat{w}_t \\ &= A_t\xi_t + Ad_{\hat{\mathcal{X}}_t} w_t \\ \text{and} \qquad \eta_t^R &= \exp(\xi_t) \ , \end{aligned} \tag{4.59}$$

where $Ad_{\mathcal{X}_t}$ is the adjoint operator for the state defined as

$$Ad_{\mathcal{X}_t} = \begin{bmatrix} \boldsymbol{R}_t & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} \\ (\boldsymbol{v}_t)_\times \boldsymbol{R}_t & \boldsymbol{R}_t & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} \\ (\boldsymbol{p}_t)_\times \boldsymbol{R}_t & \mathbf{0}_{3,3} & \boldsymbol{R}_t & \mathbf{0}_{3,3} \\ (\boldsymbol{b}_t)_\times \boldsymbol{R}_t & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \boldsymbol{R}_t \end{bmatrix} \ . \tag{4.60}$$

We can compute the matrix A_t by linearizing the invariant error term using first-order Taylor approximation $\eta_t^R = \exp(\xi_t) \approx I_d + \mathcal{L}_{\mathfrak{g}}(\xi_t)$, so we have the linearized dynamic function $g_{u_t}(\cdot)$

as

$$
\begin{aligned}
g_{u_t}(\eta_t^R) &= g_{u_t}(I_d + \mathcal{L}_{\mathfrak{g}}(\xi_t)) \\
&= \begin{bmatrix} (I + (\xi_t^{\boldsymbol{R}})_\times)(\tilde{\omega}_t)_\times & (I + (\xi_t^{\boldsymbol{R}})_\times)\tilde{\alpha}_t + \boldsymbol{g} & \xi_t^{\boldsymbol{v}} & \mathbf{0}_{3,1} \\ \mathbf{0}_{1,3} & 0 & 0 & 0 \\ \mathbf{0}_{1,3} & 0 & 0 & 0 \\ \mathbf{0}_{1,3} & 0 & 0 & 0 \end{bmatrix} \\
&- \begin{bmatrix} I + (\xi_t^{\boldsymbol{R}})_\times & \xi_t^{\boldsymbol{v}} & \xi_t^{\boldsymbol{p}} & \xi_t^{\boldsymbol{b}} \\ \mathbf{0}_{1,3} & 1 & 0 & 0 \\ \mathbf{0}_{1,3} & 0 & 1 & 0 \\ \mathbf{0}_{1,3} & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} (\tilde{\omega}_t)_\times & \tilde{\alpha}_t + \boldsymbol{g} & \mathbf{0}_{3,1} & \mathbf{0}_{3,1} \\ \mathbf{0}_{1,3} & 0 & 0 & 0 \\ \mathbf{0}_{1,3} & 0 & 0 & 0 \\ \mathbf{0}_{1,3} & 0 & 0 & 0 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{0}_{3,3} & (\boldsymbol{g})_\times \xi_t^{\boldsymbol{R}} & \xi_t^{\boldsymbol{v}} & \mathbf{0}_{3,1} \\ \mathbf{0}_{1,3} & 0 & 0 & 0 \\ \mathbf{0}_{1,3} & 0 & 0 & 0 \\ \mathbf{0}_{1,3} & 0 & 0 & 0 \end{bmatrix} .
\end{aligned} \tag{4.61}
$$

From (4.61), we can get A_t matrix as

$$
A_t = \begin{bmatrix} \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} \\ (\boldsymbol{g})_\times & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} \\ \mathbf{0}_{3,3} & I_3 & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} \\ \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & \mathbf{0}_{3,3} \end{bmatrix} . \tag{4.62}
$$

Finally, the uncertainty matrix can be propagated using the following evolution model from Kalman filtering design

$$
\frac{d}{dt} P_t = A_t P_t + P_t A_t^T + \hat{Q}_t \ , \tag{4.63}
$$

where $\hat{Q}_t = Ad_{\hat{\mathcal{X}}_t} \mathrm{Cov}(\boldsymbol{w}_t) Ad_{\hat{\mathcal{X}}_t}^T$ is the covariance matrix associated with noise $\boldsymbol{w}_t$.

4.4.4 Measurement and update model

The relative position of the contact point relative to the body frame is used as the measurement in the RIEKF filter, which is obtained from a forward kinematic calculation using joint position or angle. Similar to other measurements, the measurement of the joint parameters is also modeled to have additive Gaussian noise as follows

$$\tilde{q}_t = q_t + w_t^q, \qquad w_t^q \sim \mathcal{N}(\mathbf{0}_{M,1}, \sigma^q) \ , \tag{4.64}$$

where M is the number of corresponding joints in the forward kinematic calculation The contact point position relative to the IMU is calculated as

$$\begin{aligned} p_{C,t} &= \boldsymbol{h}_p(\tilde{q}_t - w_t^q) \\ \boldsymbol{R}_t^T(\boldsymbol{b}_t - \boldsymbol{p}_t) &= \boldsymbol{h}_p(\tilde{q}_t) - \boldsymbol{J}_v(\tilde{q}_t) w_t^q \\ \boldsymbol{h}_p(\tilde{q}_t) &= \boldsymbol{R}_t^T(\boldsymbol{b}_t - \boldsymbol{p}_t) + \boldsymbol{J}_v(\tilde{q}_t) w_t^q \ , \end{aligned} \tag{4.65}$$

where $\boldsymbol{h}_p(\tilde{q}_t)$ is the relative position of the contact point from forward kinematic calculation and $\boldsymbol{J}_v$ denotes the linear velocity Jacobian From (4.65), the measurements can be written in matrix form corresponding to the right invariant observer design as follows

$$\begin{aligned} \boldsymbol{Y}_t &= \mathcal{X}_t^{-1} d + V_t \\ \begin{bmatrix} \boldsymbol{h}_p(\tilde{q}_t) \\ 0 \\ 1 \\ -1 \end{bmatrix} &= \begin{bmatrix} \boldsymbol{R}_t^T & -\boldsymbol{R}_t^T \boldsymbol{v}_t & -\boldsymbol{R}_t^T \boldsymbol{p}_t & -\boldsymbol{R}_t^T \boldsymbol{b}_t \\ \mathbf{0}_{1,3} & 1 & 0 & 0 \\ \mathbf{0}_{1,3} & 0 & 1 & 0 \\ \mathbf{0}_{1,3} & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{0}_{1,3} \\ 0 \\ 1 \\ -1 \end{bmatrix} + \begin{bmatrix} \boldsymbol{J}_v(\tilde{q}_t) w_t^q \\ 0 \\ 0 \\ 0 \end{bmatrix} . \end{aligned} \tag{4.66}$$

4.4.5 Kalman gain calculation

From (4.47), we have the nonlinear error update model as

$$(\eta_{t_n}^R)^+ = \exp\left[L_t \left((\eta_t^R) d - d + \hat{\mathcal{X}}_t V_t \right) \right] \eta_{t_n}^R \ . \tag{4.67}$$

For simplicity, the reduced Kalman gain (K_t) and auxiliary matrix ($\Pi \triangleq \begin{bmatrix} I_3 & \mathbf{0}_{3,3} \end{bmatrix}$) is used so that

$$L_t(\hat{\mathcal{X}}_t \boldsymbol{Y}_t - d) = K_t \Pi (\hat{\mathcal{X}}_t \boldsymbol{Y}_t) \ . \tag{4.68}$$

Then, we can rewrite (4.67) with reduced Kalman gain as

$$(\eta_{t_n}^R)^+ = \exp\left[K_t \Pi \left((\eta_t^R) d + \hat{\mathcal{X}}_t V_t\right)\right] \eta_{t_n}^R \tag{4.69}$$

(4.69) can be linearized using $\eta_t^R = \exp(\xi_t) \approx I_d + \mathcal{L}_{\mathfrak{g}}(\xi_t)$ so we have

$$(\eta_{t_n}^R)^+ \approx I_d + \mathcal{L}_{\mathfrak{g}}(\xi_t) + \mathcal{L}_{\mathfrak{g}}\left(K_t \Pi \left((I_d + \mathcal{L}_{\mathfrak{g}}(\xi_t)) d + \hat{\mathcal{X}}_t V_t\right)\right)$$

$$I_d + \mathcal{L}_{\mathfrak{g}}(\xi_t^+) \approx I_d + \mathcal{L}_{\mathfrak{g}}(\xi_t) + \mathcal{L}_{\mathfrak{g}}\left[K_t \Pi \left(\begin{bmatrix} I + (\xi_t^{\boldsymbol{R}})_\times & \xi_t^{\boldsymbol{v}} & \xi_t^{\boldsymbol{p}} & \xi_t^{\boldsymbol{b}} \\ \mathbf{0}_{1,3} & 1 & 0 & 0 \\ \mathbf{0}_{1,3} & 0 & 1 & 0 \\ \mathbf{0}_{1,3} & 0 & 0 & 1 \end{bmatrix} + \hat{\mathcal{X}}_t \begin{bmatrix} \boldsymbol{J}_v(\tilde{q}_t) w_t^q \\ 0 \\ 0 \\ 0 \end{bmatrix}\right)\right]$$

$$\mathcal{L}_{\mathfrak{g}}(\xi_t^+) \approx \mathcal{L}_{\mathfrak{g}}(\xi_t) + \mathcal{L}_{\mathfrak{g}}\left[K_t \Pi \left(\begin{bmatrix} \xi_t^{\boldsymbol{p}} - \xi_t^{\boldsymbol{b}} \\ 0 \\ 1 \\ -1 \end{bmatrix} + \hat{\mathcal{X}}_t \begin{bmatrix} \boldsymbol{J}_v(\tilde{q}_t) w_t^q \\ 0 \\ 0 \\ 0 \end{bmatrix}\right)\right] \ . \tag{4.70}$$

Taking $\mathcal{L}_{\mathfrak{g}}^{-1}$ for both side in (4.70), we have

$$\begin{aligned} \xi_t^+ &= \xi_t - K_t \left(\begin{bmatrix} \mathbf{0}_{3,3} & \mathbf{0}_{3,3} & -I_3 & I_3 \end{bmatrix} \xi_t - \hat{\boldsymbol{R}}_t (\boldsymbol{J}_v(\tilde{q}_t) w_t^q)\right) \\ &= \xi_t - K_t \left(H_t \xi_t - \hat{\boldsymbol{R}}_t (\boldsymbol{J}_v(\tilde{q}_t) w_t^q)\right) \ , \end{aligned} \tag{4.71}$$

where H_t is the observation matrix

We can compute reduced dimensional Kalman gain K_t as

$$\begin{aligned} S_t &= H_t P_t H_t^T + \hat{N}_t \\ K_t &= P_t H_t^T S_t^{-1} \ , \end{aligned} \tag{4.72}$$

where $\hat{N}_t = \hat{\boldsymbol{R}}_t \boldsymbol{J}_v(\tilde{q}_t) \mathrm{Cov}(w_t^q) \boldsymbol{J}_v(\tilde{q}_t)^T \hat{\boldsymbol{R}}_t^T$ and H_t is obtained from (4.71)

Finally the update models for state and uncertainty model can be expressed as

$$\begin{aligned} \hat{\mathcal{X}}_t^+ &= \exp\left(K_t \Pi(\hat{\mathcal{X}}_t \boldsymbol{Y}_t)\right) \hat{\mathcal{X}}_t \\ P_t^+ &= (I - K_t H_t) P_t \ . \end{aligned} \tag{4.73}$$

4.5 Experiment on Digit

4.5.1 Experiment setup

Right-observer Invariant EKF was implemented on Digit to estimate the state of the robot along with LiDAR SLAM (LOAM) to obtain the odometry simultaneously in an indoor environment. The estimate is compared with the output from the default controller provided by Agility Robotics. Furthermore, in the absence of a GNSS signal, the motion capture system is used to get an absolute position as the ground truth reference of the robot. The motion capture system consists of 8 motion capture cameras and a set of retro-reflective markers attached to Digit.

Digit is controlled to follow a rectangular path in an indoor closed space as shown in Figure 4.5. The starting point and endpoint of the path are approximately the same positions to monitor the accuracy of the estimation. The kinematic, inertial, and LiDAR measurements were recorded with robot operating system (ROS) and the state estimation along with processing and filtering were done offline. Even though the system is formulated in 3D settings, we will focus on 2D results for simplicity.

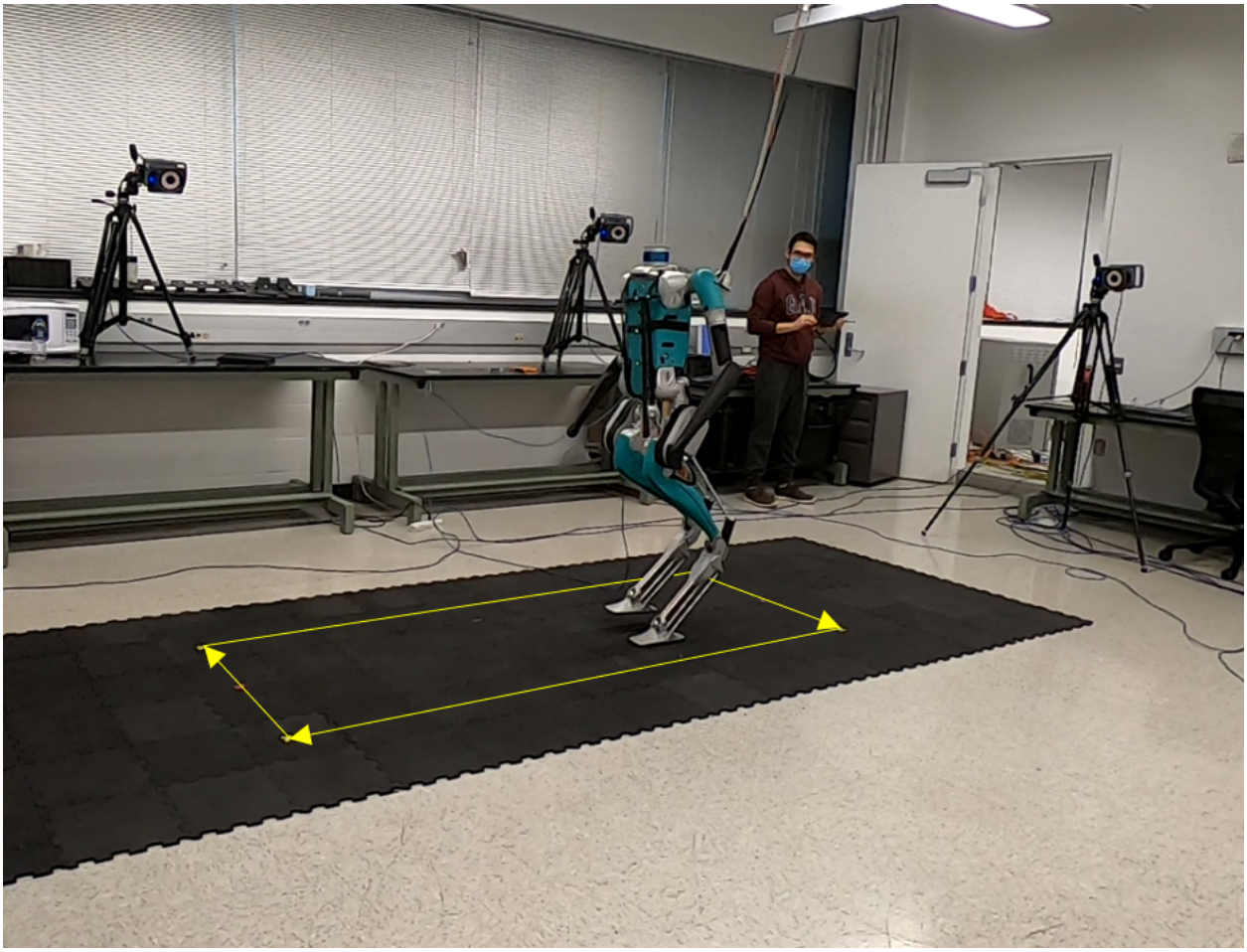


Figure 4.5: Experiment setup with motion capture system

4.5.2 Experimental results

The position estimate in x and y direction from different sources including default state estimator provided by the robot, Invariant EKF estimate using contact point position and kinematics, odometry generated from LiDAR odometry and mapping algorithm (LOAM) were measured and compared with the ground truth trajectory measured using external motion capture system. The results are plotted as follows.

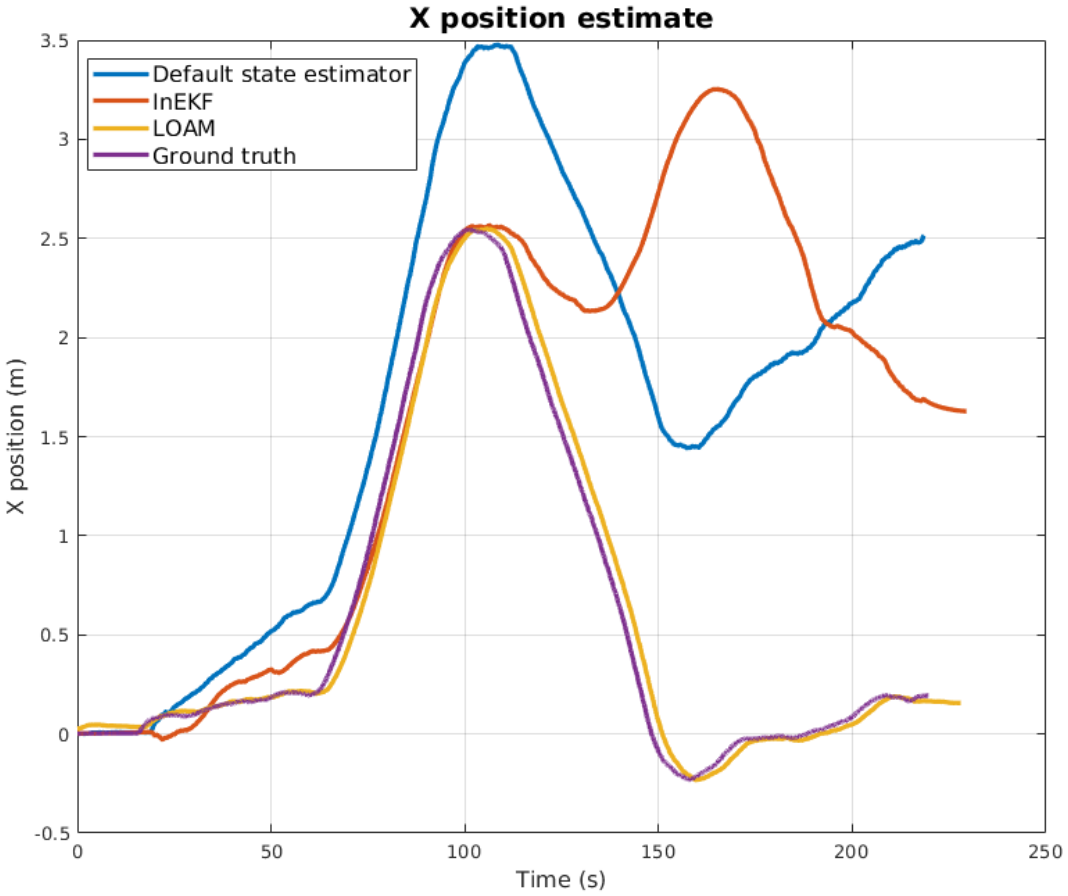


Figure 4.6: Estimate of x position over time from different sources compared to ground truth

From Figure 4.6, the position estimate in the x-direction from LOAM (shown in yellow) is significantly more accurate than other sources compared to the ground truth trajectory (shown in purple). The mean square error for LOAM is at 0.0648 m^2. For the x-position estimate from the Invariant EKF (shown in orange), the estimate is reasonably accurate in the first 100 seconds of the experiment, but the estimate diverges from the actual trajectory when the robot starts to walk backward. The mean square error of the first 100 seconds for Invariant EKF is approximately 1.6405 m^2. Finally, for the default state estimator from the robot itself, even though the trends of the graph look similar to other sources, the estimation

error or drift is significantly large and the highest estimation error is more than 2 meters.

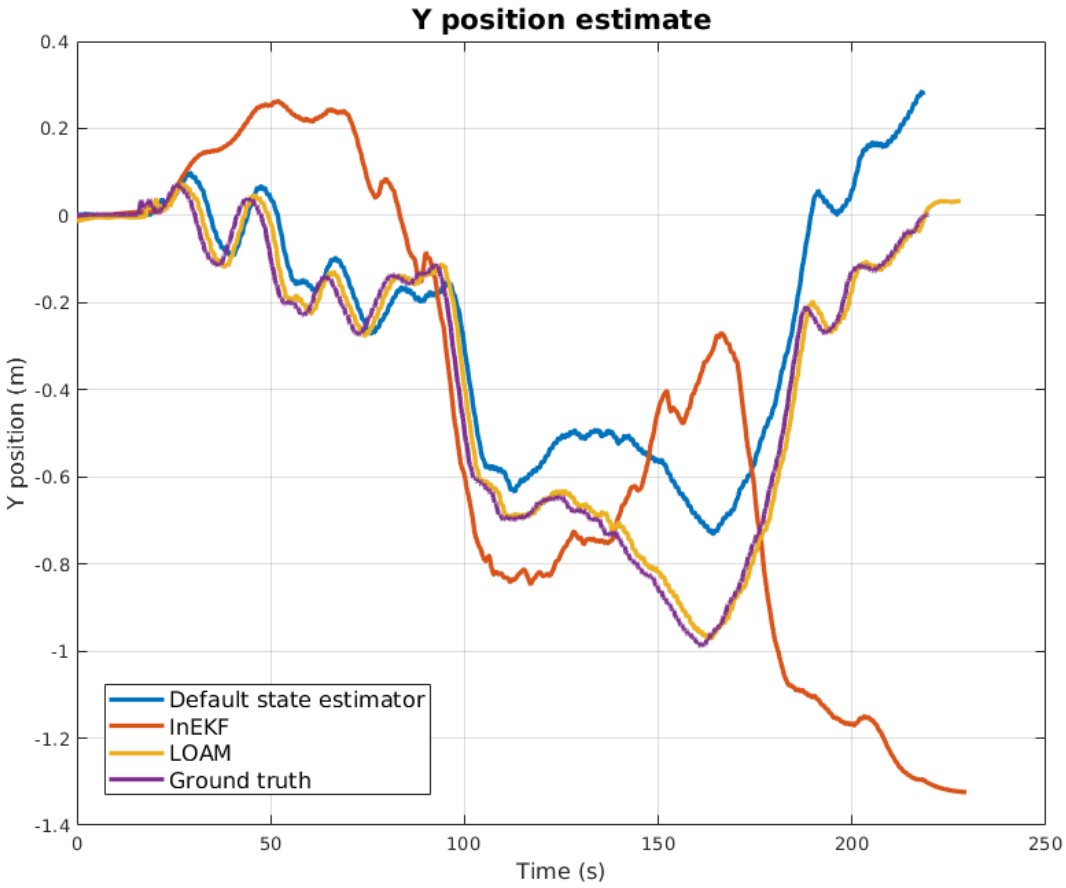


Figure 4.7: Estimate of y position over time from different sources compared to ground truth

Figure 4.7 shows the position estimation in the y-direction, which represents the ability to predict the lateral movement of the robot. Similar to results in Figure 4.6, the results from LOAM still demonstrates highest accuracy with mean square error at 0.0309 m^2. However, in contrast to the previous figure, the y-position estimate from Invariant EKF shows a large error due to divergence early in the trajectory. Even though the estimation recovers when

the robot is sidestepping, the rest of the trajectory has a large estimation error. The default state estimator has good estimation performance in the earlier part of the trajectory at an approximate mean square error of 0.0320 m^2 but starts to diverge from true trajectory after the robot walks backward for a short period of time.

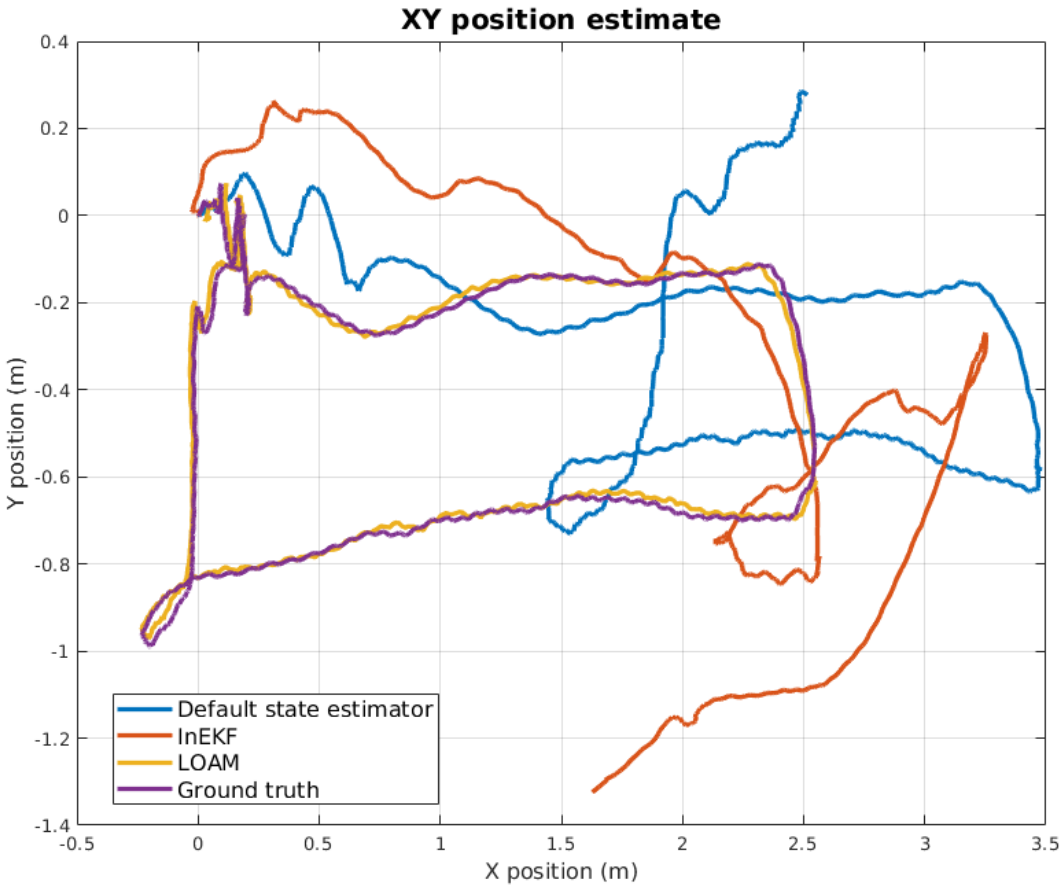


Figure 4.8: Estimate of X-Y position from different sources compared to ground truth

In Figure 4.8, the estimated position in the X-Y plane is shown to give a better understanding of the spatial movement characteristic of the robot in the experimental space. The

robot starts at the origin or (0.0, 0.0) position shown in the graph. The initial movement of the robot is aggressive, resulting in large noise in IMU measurements, which would consecutively affect the estimation from both the default state estimator and the Invariant EKF filter where IMU measurement plays a big role in propagating the system. Since the LOAM algorithm does not depend on IMU measurements, the noises in IMU did not affect the odometry estimation. So, the position estimation results are significantly better for LiDAR SLAM.

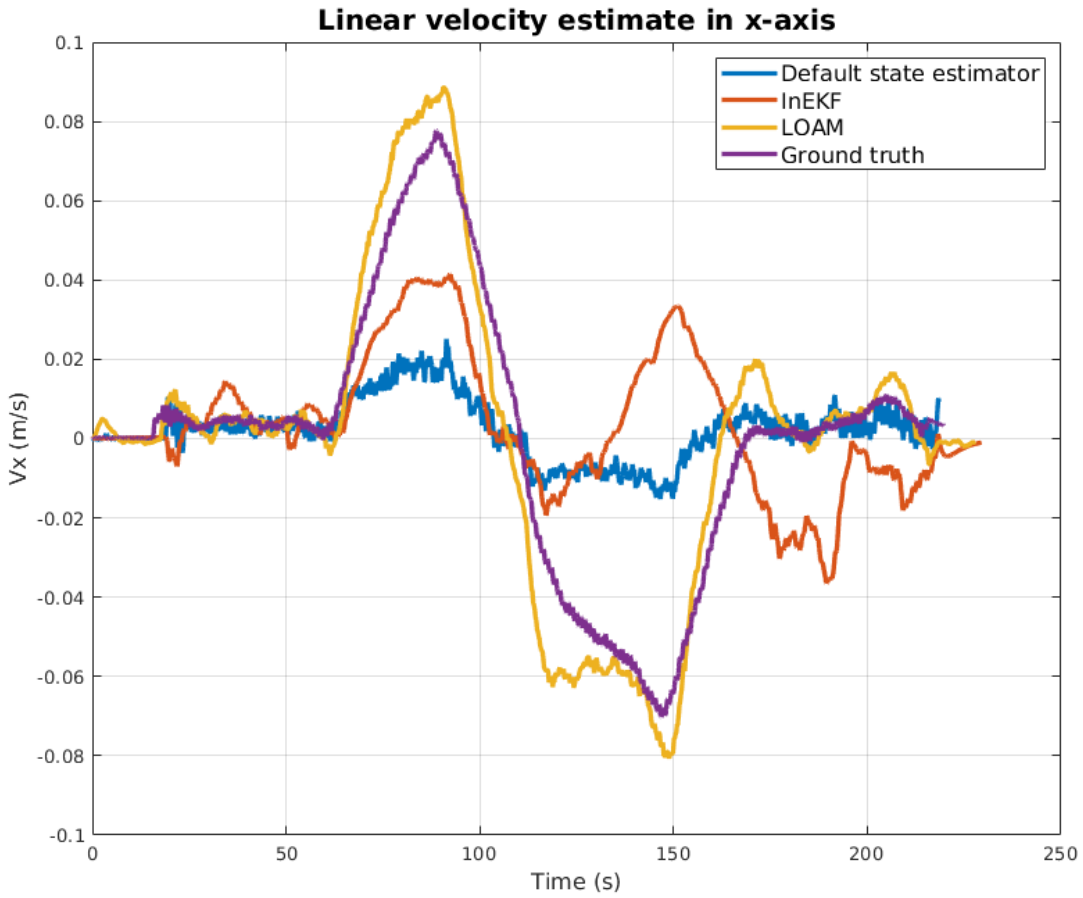


Figure 4.9: Estimate of linear velocity in x-direction from different sources

From Figure 4.9, although the velocity estimate from Invariant EKF and default state estimator seems less noisy, compared to the ground truth, the estimation underestimates the actual velocity which means that it did not accurately depict the velocity in the x-direction. As the Invariant EKF diverges, the estimation became unreliable and since the implemented Invariant Kalman filter only relies on proprioceptive sensors, there is no external reference to recover the trajectory. Although showing some estimation error, the velocity estimation of LOAM resembles the actual velocity the most out of the sources of prediction we have.

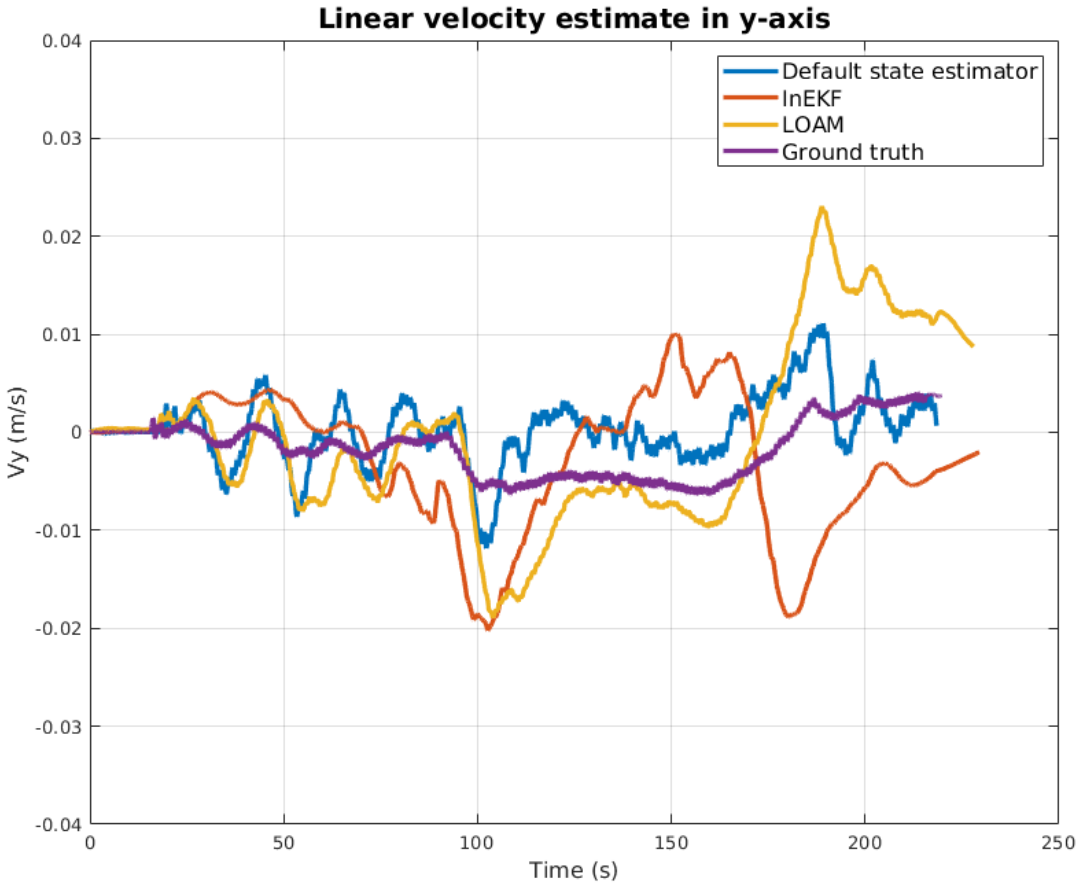


Figure 4.10: Estimate of linear velocity in y-direction from different sources

Due to the general movement of bipedal robots, the lateral velocity is usually noisy, and is difficult to obtain a robust estimate. The characteristics of each source of prediction seem noticeably different, but they still follow a certain trend. LOAM shows accumulated drift at the later stage of the trajectory and Invariant EKF also shows a large error in prediction in the same period as well. Default state estimator demonstrates great results in lateral prediction based on the ground truth trajectory.

4.6 Summary

In this chapter, we introduced the state-of-the-art method on state estimation, the Extended Kalman Filter which is very popular in robotics applications. However, due to the linearization through first-order Taylor expansion which leads to divergence in some practical operations. Therefore, the idea of incorporating geometry into the filter is introduced. Lie group became one of the focus of the Kalman filter due to the group-affine property resulting in invariant error dynamic being independent of the previously estimated state. We explained the details and theory of the matrix Lie group and how the Invariant Extended Kalman filter takes advantage of the property to achieve highly convergence characteristics.

Then, we showed the derivation of the system's dynamic model and how to formulate update model equations as an implementation of Right Invariant Extended Kalman Filter introduced in [16] on Digit. The position of the contact point relative to the body frame of the robot is used as a measurement to update the predicted state from the deterministic propagation model. Finally, we perform an experiment to verify the performance of the filter as well as the accuracy of the odometry generated from LOAM compared to the ground truth from the motion capture system.

CHAPTER 5

CONCLUSION

5.1 Conclusion

We demonstrated various methods to process sensor measurements and convert them into useful information of the environment for perception purposes. First, we obtained raw data from sensors and processed them in a meaningful manner to conform with the objective of the task. Down-sampling and voxel filtering techniques were used to reduce the number of the point cloud in a 3D space. The amount of point cloud filtered depends on the leaf size which is the minimum size that a voxel can take. In our implementation, the leaf size is set to be 0.02 meters per side of a cubic voxel which is the optimal trade-off between the amount of data retained and the amount of data removed. We also used the RANSAC algorithm to separate the ground region of the point cloud from the obstacle point cloud, so we can define the task space for the robot while avoiding obstacles. Finally, we fused all the point clouds from various sensors together to obtain a complete field of perception that the robot is capable of sensing.

Then, we used the processed data to generate and reconstruct the three-dimensional scene of the environment. Given the odometry of the robot from a fixed frame, we can use OctoMap to probabilistically register the obtained point cloud onto the map and store them in an octree structure. The advantage of OctoMap is the ability to update the map in a dynamic environment through ray-casting techniques and probabilistic inferences. In the scenario where an external source of odometry is not available and we want to obtain

a 3D map while localizing our robot or agent onto the newly created map, we can use the Simultaneous Localization and Mapping or SLAM approach. We used a LiDAR sensor to perform SLAM due to the accuracy of the depth measurement and the independence of lighting conditions. The LiDAR SLAM we implemented is introduced in [45] which is called LiDAR Odometry And Mapping or LOAM. It can provide a low drift and robust 3D map of the environment as well as perform an accurate localization of the map is created. However, the disadvantage of LOAM is that the environment is assumed to be static, so dynamic objects were registered onto the map as well. This causes noises in the map and sometimes an error during the point cloud registration step. Nonetheless, the LiDAR SLAM shows a high accuracy position estimate in the indoor environment as well as a robust 3D map output.

Finally, we implemented Invariant Extended Kalman Filter (InEKF) on Digit to estimate the state of the robot. The position estimate was evaluated in the experiment with ground truth trajectory from the motion capture system. As seen from the results, Invariant EKF's initial position estimate can be noisy, but it eventually recovers to the actual value after some time. However, the position estimation from Invariant EKF, as well as the default state estimator, diverges from true values in the latter part of the trajectory. This estimation error is assumed to have been caused by the noisy inertial measurement from the high update frequency of IMU. Even though, theoretically, Invariant EKF has log-linear error property which makes the error dynamic independent of the previous state as long as the group affine property is satisfied, practically, there might be some violation of the assumption which will result in an error being not fully invariant. So, along with asynchronous initial orientation of frames and hardware implementation, the result from Invariant EKF did not shows the expected results and showed a large error when divergence occurred. Overall, for position estimation, LiDAR SLAM demonstrated better results since LiDAR is an exteroceptive sensor that continuously collects external information of the environment, so the position estimate in fixed world frame is more accurate than estimating the state of the robot from IMU and kinematic measurements which are all considered internal measurements.

5.2 Future work

5.2.1 Verification of state estimation assumption and parameters

The Invariant EKF is implemented on Digit using an open-source package from [16] which is originally implemented on Cassie. Therefore, there are some minor differences in contact position assumption which is varied depending on the control policy on walking gait. In the current implementation, the contact point is defined at the heels of Digit's feet. These differences might violate the zero velocity assumption at the contact point and leads to accumulated error and inaccuracies in state estimations. So, the details of the parameters need to be intensively revised.

5.2.2 Experimental Implementations

Due to time limitations, the Invariant EKF implementation is not tested in multiple scenarios including the various movement of the robot and different paths to verify its performance. This can provide us with more understanding of the limitation and issues of the estimation filter in a different scenario. Additionally, the velocity and orientation estimation, which are crucial information in control modules, were not investigated in this thesis due to a lack of time limitation and proper ground truth references.

5.2.3 Integration of LiDAR odometry to Invariant EKF filter

From the experiment, the LOAM implementation of LiDAR SLAM gives good results in position estimate, but the velocity estimate is not very accurate due to the relatively low update frequency (10 Hz) compared to the high-frequency update of the IMU and joint encoders. Therefore, the output from SLAM can be used to formulate a left-invariant observer which can be expressed in Lie group format and integrated into the current state estimation module as an extra measurement update layer to provide some global information to the filters, increasing the consistency in fixed frame estimation.

BIBLIOGRAPHY

[1] Martin Barczyk and Alan F Lynch. "Invariant extended Kalman filter design for a magnetometer-plus-GPS aided inertial navigation system". In: *2011 50th IEEE Conference on Decision and Control and European Control Conference*. IEEE. 2011, pp. 5389–5394.

[2] Martin Barczyk and Alan F Lynch. "Invariant observer design for a helicopter UAV aided inertial navigation system". In: *IEEE Transactions on Control Systems Technology* 21.3 (2012), pp. 791–806.

[3] Axel Barrau and Silvère Bonnabel. "The invariant extended Kalman filter as a stable observer". In: *IEEE Transactions on Automatic Control* 62.4 (2016), pp. 1797–1812.

[4] Axel Barrau and Silvere Bonnabel. "Invariant kalman filtering". In: *Annual Review of Control, Robotics, and Autonomous Systems* 1 (2018), pp. 237–257.

[5] Mehdi Benallegue and Florent Lamiraux. "Estimation and stabilization of humanoid flexibility deformation using only inertial measurement units and contact information". In: *International Journal of Humanoid Robotics* 12.03 (2015), p. 1550025.

[6] Silvere Bonnabel. "Left-invariant extended Kalman filter and attitude estimation". In: *2007 46th IEEE Conference on Decision and Control*. IEEE. 2007, pp. 1027–1032.

[7] Silvère Bonnable, Philippe Martin, and Erwan Salaün. "Invariant extended Kalman filter: theory and application to a velocity-aided attitude estimation problem". In: *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. IEEE. 2009, pp. 1297–1304.

[8] Gregory S Chirikjian. *Stochastic models, information theory, and Lie groups, volume 2: Analytic methods and modern applications*. Vol. 2. Springer Science & Business Media, 2011.

[9] Andrew J Davison et al. "MonoSLAM: Real-time single camera SLAM". In: *IEEE transactions on pattern analysis and machine intelligence* 29.6 (2007), pp. 1052–1067.

[10] Jorhabib Eljaik, Naveen Kuppuswamy, and Francesco Nori. "Multimodal sensor fusion for foot state estimation in bipedal robots using the extended kalman filter". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 2698–2704.

[11] Martin A Fischler and Robert C Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* 24.6 (1981), pp. 381–395.

[12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[13] Kristen Grauman and Bastian Leibe. "Visual object recognition". In: *Synthesis lectures on artificial intelligence and machine learning* 5.2 (2011), pp. 1–181.

[14] Giorgio Grisetti et al. "A tutorial on graph-based SLAM". In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), pp. 31–43.

[15] Gaël Guennebaud, Benoit Jacob, et al. "Eigen". In: *URl: http://eigen. tuxfamily. org* 3 (2010).

[16] Ross Hartley et al. "Contact-aided invariant extended Kalman filtering for robot state estimation". In: *The International Journal of Robotics Research* 39.4 (2020), pp. 402–430.

[17] Wolfgang Hess et al. "Real-time loop closure in 2D LIDAR SLAM". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 1271–1278.

[18] Dirk Holz et al. “Registration with the point cloud library: A modular framework for aligning in 3-D”. In: *IEEE Robotics & Automation Magazine* 22.4 (2015), pp. 110–124.

[19] Armin Hornung et al. “OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees”. In: *Autonomous Robots* (2013). Software available at `https://octomap.github.io`. DOI: `10.1007/s10514-012-9321-0`. URL: `https://octomap.github.io`.

[20] Arthur J Krener. “The convergence of the extended Kalman filter”. In: *Directions in mathematical systems theory and optimization*. Springer, 2003, pp. 173–182.

[21] In-So Kweon et al. “Terrain mapping for a roving planetary explorer”. In: *IEEE International Conference on Robotics and Automation*. IEEE. 1989, pp. 997–1002.

[22] Krystof Litomisky and Bir Bhanu. “Removing moving objects from point cloud scenes”. In: *International Workshop on Depth Image Analysis and Applications*. Springer. 2012, pp. 50–58.

[23] Minjie Liu et al. “A convex optimization based approach for pose SLAM problems”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 1898–1903.

[24] Ying Liu and Ruofei Zhong. “Buildings and terrain of urban area point cloud segmentation based on PCL”. In: *IOP Conference Series: Earth and Environmental Science*. Vol. 17. 1. IOP Publishing. 2014, p. 012238.

[25] Philippe Martin and Erwan Salaün. “Generalized multiplicative extended kalman filter for aided attitude and heading reference system”. In: *AIAA Guidance, Navigation, and Control Conference*. 2010, p. 8300.

[26] Marius Miknis et al. “Near real-time point cloud processing using the PCL”. In: *2015 International Conference on Systems, Signals and Image Processing (IWSSIP)*. IEEE. 2015, pp. 153–156.

[27] Hans Moravec and Alberto Elfes. "High resolution maps from wide angle sonar". In: *Proceedings. 1985 IEEE international conference on robotics and automation*. Vol. 2. IEEE. 1985, pp. 116–121.

[28] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. "ORB-SLAM: a versatile and accurate monocular SLAM system". In: *IEEE transactions on robotics* 31.5 (2015), pp. 1147–1163.

[29] Richard M Murray, Zexiang Li, and S Shankar Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 2017.

[30] Paul Newman, David Cole, and Kin Ho. "Outdoor SLAM using visual appearance and laser ranging". In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE. 2006, pp. 1180–1187.

[31] Manuel González Ocando et al. "Autonomous 2D SLAM and 3D mapping of an environment using a single 2D LIDAR and ROS". In: *2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*. IEEE. 2017, pp. 1–6.

[32] GF Page. "MULTIPLE VIEW GEOMETRY IN COMPUTER VISION, by Richard Hartley and Andrew Zisserman, CUP, Cambridge, UK, 2003, vi+ 560 pp., ISBN 0-521-54051-8.(Paperback£ 44.95)". In: *Robotica* 23.2 (2005), pp. 271–271.

[33] Young-Sik Park et al. "Navigation system of UUV using multi-sensor fusion-based EKF". In: *Journal of institute of control, robotics and systems* 22.7 (2016), pp. 562–569.

[34] Anandakumar M Ramiya, Rama Rao Nidamanuri, and Ramakrishan Krishnan. "Segmentation based building detection approach from LiDAR point cloud". In: *The Egyptian Journal of Remote Sensing and Space Science* 20.1 (2017), pp. 71–77.

[35] Nicholas Rotella et al. "State estimation for a humanoid robot". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 952–958.

[36] Radu Bogdan Rusu and Steve Cousins. "3d is here: Point cloud library (pcl)". In: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 1–4.

[37] Angelo M Sabatini. "Quaternion-based extended Kalman filter for determining orientation by inertial and magnetic sensing". In: *IEEE transactions on Biomedical Engineering* 53.7 (2006), pp. 1346–1356.

[38] Tixiao Shan and Brendan Englot. "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 4758–4765.

[39] Tixiao Shan et al. "Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 5135–5142.

[40] Yongkyu Song and Jessy W Grizzle. "The extended Kalman filter as a local asymptotic observer for nonlinear discrete-time systems". In: *1992 American control conference*. IEEE. 1992, pp. 3365–3369.

[41] Kai Storjohann et al. "Visual obstacle detection for automatically guided vehicles". In: *Proceedings., IEEE International Conference on Robotics and Automation*. IEEE. 1990, pp. 761–766.

[42] Nikolas Trawny and Stergios I Roumeliotis. "Indirect Kalman filter for 3D attitude estimation". In: *University of Minnesota, Dept. of Comp. Sci. & Eng., Tech. Rep* 2 (2005), p. 2005.

[43] Rudolph Triebel, Patrick Pfaff, and Wolfram Burgard. "Multi-level surface maps for outdoor terrain mapping and loop closing". In: *2006 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2006, pp. 2276–2282.

[44] Manuel Yguel, Olivier Aycard, and Christian Laugier. "Update Policy of Dense Maps: Efficient Algorithms and Sparse Representation". In: *Springer Tracts in Advanced Robotics Field and Service Robotics* (), pp. 23–33. DOI: 10.1007/978-3-540-75404-6_3.

[45] Ji Zhang and Sanjiv Singh. "LOAM: Lidar Odometry and Mapping in Real-time." In: *Robotics: Science and Systems*. Vol. 2. 9. 2014.

[46] Runyang Zou, Xueshi Ge, and Geng Wang. "Applications of RGB-D data for 3D reconstruction in the indoor environment". In: *2016 IEEE Chinese Guidance, Navigation and Control Conference (CGNCC)*. IEEE. 2016, pp. 375–378.