ABSTRACT

JUNEJA, ABHAYJEET SINGH. A Comparative Study of Slam Algorithms for Indoor Navigation of Autonomous Wheelchairs. (Under the direction of Edward Grant).

Following the rapid research and development in the field of autonomous cars followed by the brilliant research in biped and quadruped robots, it is not incorrect to state that the dawn of automation is upon us and robots are being developed to be a man's best assistant. Striving to make life easier for the patients with Motor Neuron Diseases like Amyotrophic lateral sclerosis (ALS), we attempt to develop an autonomous wheelchair which will help such patients move from point A to B. Robots follow a common protocol which is inspired by humans, and that can be explained with the three words - perception, thinking, and reaction. A simple rule-based robot will perceive the environment, and react according to the predefined rules. Things change when rules are defined on-the-go by the means of machine learning or deep learning.

SLAM or Simultaneous Localization and Mapping is an essential problem, where the goal is to make the robot aware of its position in the environment while building the map of the environment. Many solutions for SLAM have been developed in past two decades, and in this thesis we study five of the algorithms viz., Gmapping, Hector, RTAB-Map, VINS-Mono, and RGBD-SLAM. For the test bench, an electric powered wheelchair was modified to incorporate the desired sensors, e.g., LiDAR, Kinect camera, IMU, and wheel encoders. The wheelchair was made functional for testing and comparing the defined metrics for localization, mapping accuracy and its usability for patients with differing medical conditions. RTAB-Map was found to be the most scalable algorithm, one that works with different combinations of sensors. This algorithm produces accurate maps while accurately estimating robot location consistently, for a range of speeds.

A Comparative Study of Slam Algorithms for Indoor Navigation of Autonomous Wheelchairs

by
Abhayjeet Singh Juneja

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Electrical Engineering

Raleigh, North Carolina

2019

APPROVED BY:

Andrew Rindos	John Muth

Edward Grant
Chair of Advisory Committee

DEDICATION

To my parents and sister who always supported me and believed in my dreams.

To my professor and advisor Dr. Grant who provided me with this unparalleled opportunity to work on something that may one day bring a smile on someone's face.

To Heather and Sanket who have been there as a constant support and inspiration to bring this work to completion.

Thank You

BIOGRAPHY

The author was born in New Delhi, the capital of India. Both his parents and sister are doctors and practice medicine in India. He completed his Bachelors in Computer Science and Engineering from Vellore Institute of Technology in Chennai, India. During his Bachelors, he took up and completed a number of Robotics projects which were primarily related to Embedded Systems and Robotics Control. His interest in Electronics and Embedded Systems while pursuing a degree in Computer Science helped him expand his knowledge base along multiple disciplines. He was a part of BAJA-SAE team during his undergrad and helped design a data acquisition system for the All Terrain Vehicle. Currently, his interest lies in the field of Perception and Machine Learning for robots.

ACKNOWLEDGEMENTS

Firstly, I would like to acknowledge the opportunity that my professor and advisor Dr. Edward Grant provided me with. His constant faith and dedication towards the project is what kept me from not giving up. The time spent over the period of two semesters was full of learning and having fun. Dr. Grant has always emphasized upon the fact that work is important but having fun while working is more important. This is something I will try to live by. I would like to thank Hamed Mohammedbagherpoor for guiding me to the right path at times when I was stuck. Working in CRIM lab was a journey I will always cherish. I would also like to extend my sincere gratitude to Dr. John Muth and Dr. Andrew Rindos for accepting to be a part of the advising committee. My colleague and friend, Lakshay Bhandari, was a constant support for bringing this thesis to completion. We have spent hours together to make the wheelchair functional. The worked we did in collaboration in Spring and Summer of 2018 has been used and mentioned as a precursor in this thesis.

There are many people outside my academic circle who have been a part of my journey and believed in me. Heather, Sanket Goutam, Shivam Luthra, Payal Singal, Pritha Singh are a few special people who will always be ready to extend a helping hand. Honestly, I cannot ask for better friends. Lastly, I would like to add that I'm blessed to have such parents who have supported me through thick and thin. Nevertheless, they have encouraged me to follow my dreams and supported my decision to travel across oceans to pursue a master's degree. A sincere thanks to everyone for being a part of this adventurous journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER

1

INTRODUCTION

The role of assistive technology in lives of people with disabilities is not to compensate or to adapt for missing or delaying functions; it is also used to support for everyday living in targeted performance areas [Tro89]. Wheelchairs act as one such assistive technology which has evolved over centuries. Electric Powered wheelchairs are available which help the patients with mobility impairment to travel from point A to point B using a joystick control [Dic10]. However, there are often medical conditions when patients become incapable of moving even a single finger to control a joystick. For such patients assistive technologies have been devised which help them regain their strength if possible. A post traumatic impairment could be temporary or permanent. Therapy works well in case of

temporary impairment since the patient shows signs of internal recovery. But, in cases of permanent impairment where there is no sign of recovery, life becomes stagnant for such patients. An example would be of a patient fighting the ALS (Amyotrophic Lateral Sclerosis). In order to improve the quality of life of these patients and to make them capable of performing ADLs (Activities of Daily Living) to a certain extent, there is a need for "smart" autonomous wheelchairs.

Recently, smart and intelligent wheelchairs have been conceptualized by embedding a range of sensors for patient monitoring and control [Pos14], [Pos12]. Such systems assist the care-takers, nurses, and doctors to keep track of patient health, but with little regard to patient's improvement in performing ADLs. Autonomous wheelchairs are an attempt at making the patient independent in terms of making decisions for themselves, removing the need for them to continually rely on other people for transportation. Autonomous wheelchairs may not help the patient regain the motor neuron activity, but will definitely improve their quality of living.

Research in the field of autonomous robots is directly applicable to a wheelchair, because a wheelchair is a two wheel differential drive robot chassis with additional idler wheels. Developing an autonomous wheelchair is similar to developing any other autonomous robot but the challenge is to choose the most suitable control algorithm from a large pool of available algorithms. One such algorithm is called SLAM - Simultaneous Localization and Mapping. SLAM is essential to autonomous robots as it facilitates autonomous navigation by building a map of the environment and estimating the position of the robot within the said map [RB04].

1.1 Thesis Statement

Various SLAM algorithms have been proposed to efficiently map and localize a robot in an observed and observable environment, but not all algorithms are applicable in every situation. In this project a comparative study is conducted to determine which SLAM algorithm is best suited to successfully map and localize an autonomous wheelchair, given its motion and computation constraints.

1.2 Organization

The remainder of the thesis is structured as follows.
Chapter 2 provides the reader with required background into research related to autonomous robots, and of related research work that is currently being pursued.
Chapter 3 gives an overview of the test bed, i.e., the wheelchair used to test the algorithms. Chapter 4 discusses in range of the various LiDAR based and camera based SLAM algorithms tested an compared. Chapter 5 discusses the metrics used and provides the experimental setup and results for each tested algorithm. Finally, the thesis in concluded in Chapter 6 with a summary of results.

CHAPTER

2

BACKGROUND AND RELATED WORK

2.1 Background

2.1.1 Autonomous Systems

When we hear the phrase "Autonomous System", we think of industry automation, UAVs (Unmanned Aerial Vehicles), self-driving cars and the list goes on. These are all modern technologies with ancient roots. The idea of autonomous systems can be dated back to centuries ago with the *"airscrew"*, conceptualized by Leonardo Da Vinci in the 15th century [FOL76], which is now more commonly known as Da Vinci's Helicopter. The modern

history of automation goes back to the World War II, when Alan Turing developed the Bombe, which helped decipher the German cipher machine, Enigma. He went on to invent the Turing Machine which he called the "a-machine" or "automatic machine" [Hod12]. The Turing Machine is the conceptual precursor of the modern computer, and helps explain the theory of automation. Every modern technology uses the theory of automation in some form. A simple *if-else* statement in a basic computer programming language is an example of automation. Modern applications of autonomous systems deal with solving modern problems, like an efficient transportation without human interference, e.g., self-driving cars, [Urm08], home automation with cleaning robots like Roomba by iRobot [FD06], factory automation in terms of precise manufacturing, warehouse organization [Kon00], and delivery using robots [Zho14].

1. **Self Driving Cars**: Self-driving cars (or autonomous cars, or driverless cars) is currently a hot research topic in the automotive industry as well as in academia. Many companies have put in thousands of hours of research into bringing the best technology in the market. Recently, the technology has also been nearly successfully commercialized by companies like Tesla, Uber, Aptiv, NuTonomy, etc. The beginning of the research can be dated back even before the DARPA Grand Challenge in 2004 where all the competing cars failed during the initial miles, but realized a platform to showcase their research in autonomous cars. The challenge was repeated in 2005 [Bue07] where the car *Stanley* by Stanford University bagged the first position [Thr06].

2. **Warehouse and Delivery Robots**: A solution, provided by the engineers to the supply-chain management and logistics problem, are the warehouse and delivery robots. [ROB16] states the need of robotics in logistics. These robots are capable of

navigating the warehouse by either learning the map or using a predefined map of the warehouse. Unlike self driving cars, GPS is not an option available since there is no GPS signal available indoors. The localization problem is solved by using beacons or visual/colored landmarks which the robots use to estimate their position. The robots can pick, drop, and sort packages within the warehouse. Since the map of the ware house is usually static with defined locations of shelves and stacks for picking and dropping, path planning is as easy as implementing one algorithm and updating the path dynamically to avoid any moving obstacles in the way.

2.1.2 Common Pipeline

Every autonomous robot developed follows a common control pipeline. The control is a closed loop system with feedback, so the autonomous robot needs sensors. Fig. 2.1 shows a common control pipeline which is at the core of every autonomous robot. Two types of sensors are commonly used:

- **Exteroceptive Sensors**: Those which help in sensing the environment, i.e., external to the robot. Example: LiDAR, Camera, Ultrasonic Sensor, Radar.
- **Proprioceptive Sensors**: Those which help in sensing/measuring the state of the "ego robot," i.e., the robot itself. Example: Encoders, IMU.

Perception, in simple words, is what the sensors see (perceive) from the environment. Sensors can see a person walking (dynamic object) or a painting on the wall (static object or Landmark). These observations are used to build a map of the environment. The static objects (or landmarks) are a part of the global map, where as dynamic objects are used by the local planner to plan a path around them by estimating the motion of the dynamic objects.

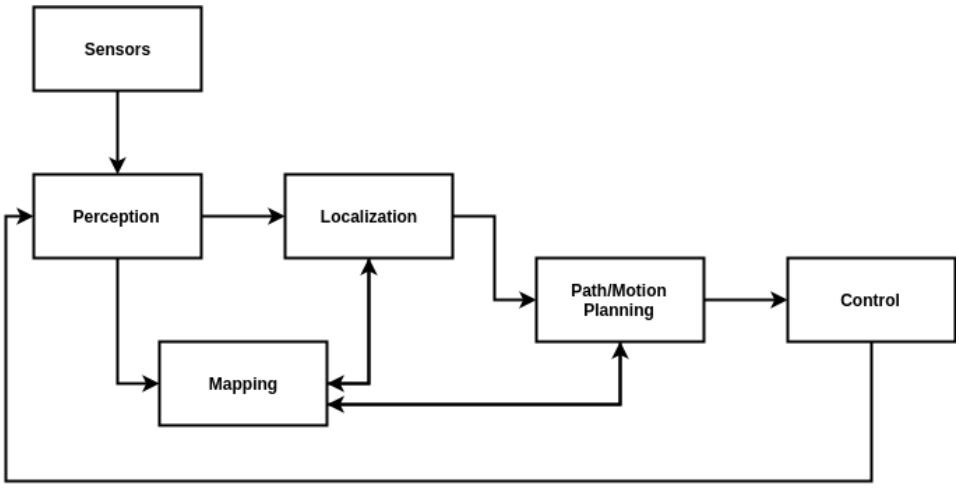


Figure 2.1 The common control pipeline of an autonomous robot

Another part of the pipeline is Localization. The landmarks generated by the perception algorithm are used to learn a map. These landmarks are also used to estimate the pose (position and orientation) of the robot in the generated map.

Unlike self-driving cars, where a major source of positional data is the GPS receiver, indoor robots used for warehouse navigation and vacuum cleaning robots cannot rely on the GPS for navigation and localization. Various algorithms have been proposed for localization that do not require GPS. These algorithms use sensors like LiDAR and camera to make sense of the environment and estimate the pose of the robot. [LT10] proposed a localization method that makes use of an offline grid map of probability distributions over environment reflectance to the laser rays from the LiDAR. Some methods use LiDAR data to build a map and camera data to estimate the location of the self-driving car relative to the generated map. [Xu17] proposed a localization method that matches stereo images to a 3D point-cloud map.

Once the map is generated and the robot is localized, the next step is the decision making which includes planning a path to the destination and also taking care of the Holo-

nomic constraints of the robot while doing so. Algorithms like Dijkstra's, A*, RRT, and RRT* are used for path planning.

The localization problem and the Mapping problem together construct the SLAM problem. The attempt is to squeeze the localization and mapping pipeline into one single SLAM algorithm. SLAM or Simultaneous Localization and Mapping has been the most talked of problem in the field of Robotics. The problem was first introduced by Durrant-Whyte and John J. Leanord in 1991 [LDW91]. The problem states that if a moving robot is kept in an unknown environment, how likely is it to build a consistent map of the observed environment while estimating the current location and navigating through the environment. Over the past two decades, many solutions to the SLAM problem have been provided and is now considered to be solved. Some solutions require a combination of sensors whereas some require only one sensor. Algorithms are designed to process the incoming data from these sensors and build a map through observing environment, learning landmarks, and simultaneously estimating the location of the moving platform in that map with respect to the landmarks.

Any uncertainty inherent in either mapping and/or localization is commonly caused by noisy sensor measurements. Uncertainty leads to the inaccurate perception of an environment. Separately, Localization and Mapping can be explained by the following equations:

$$Localization : p(x|z,u) \tag{2.1}$$

$$Mapping : p(m|z,u) \tag{2.2}$$

Where, m is the environment map, x is the current pose of the robot in the world frame, z is the observation, i.e., information from sensor measurements, and u is the control command given to the robot, or odometry from the robot. Together these form the SLAM

problem as follows.

Given,

$$Observations: \ z_{0:T} = z_0, ..., z_T, \ and \tag{2.3}$$

$$Odometry \ measurements: \ u_{1:T} = u_1, ..., u_T \tag{2.4}$$

Find,

$$Posterior: \ p(x_{0:T}, m | z_{0:T}, u_{1:T}) \tag{2.5}$$

Where, the SLAM problem is solved over time (0:T) and is expressed as a conditional probability of the path ($x_{0:T}$: pose over time) and map (m), given observations ($z_{0:T}$) and control and odometry ($u_{1:T}$) over time. This probability usually follows a Gaussian distribution which helps with better estimation of the unknown states over time.
Many LiDAR based SLAM algorithms use Kalman Filters or Particle Filters [Gri05] and [Gri07] (aka Sequential Monte Carlo) based on Monte Carlo algorithms, whereas Visual (camera based) SLAM incorporates Structure from Motion (SFM) techniques to estimate the camera pose and motion in the environment by processing multiple images of the same environment taken from different angles.

2.1.3 Sensors

A variety of sensors have been used in order to try to solve the SLAM problem. The type of exteroceptive sensors can be categorized based on the range of sensors, or based on the type of data. Based on the sensors' range there are short range sensors like ultrasonic sensor, and long range sensors LiDAR and Camera. Based on the type of data, there are sensors which provide color (RGB) images, Depth Images, 2D point cloud, 3D point cloud for the perception and mapping. For localization, proprioceptive sensors for mea-

suring the odometry of the moving platform include encoders, Inertial Measurement Unit (IMU), GPS. Received signal strength from beacons is also used to localize a robot in the map. The beacons act as landmarks used to estimate the pose of the robot.

2.1.4 Previous Work

To implement various SLAM solutions on the Center for Robotics and Intelligent Machines (CRIM) test bed, it was required to test the capabilities of the system. An overview of the system is provided in Chapter 3. The research was done in the CRIM lab, NCSU, in collaboration with Dr. Edward Grant, Lakshay Bhandari, and Hamed Mohammedbagherpoor.

Similar to building an autonomous car test bed, where the first task is to give the car the ability to be controlled-by-wire, that is, instead of manual human control the car needs to be capable of taking commands from a micro-controller or a computer. Here, the first task was to make the wheelchair capable of being controlled by a computer. The computer used is a Raspberry Pi which was interfaced with a motor driver. Unlike a car where the throttle/speed is controlled by the amount of gas going into the engine, the control system of a powered wheelchair is as simple as it can be. The wheelchair takes two 12 Volt batteries which power the 2 DC motors. The motors are interfaced with the Raspberry Pi through a 10 ampere motor driver which takes the PWM signals from the raspberry pi and that helps controlling the speed of the motor. The wheelchair follows the kinematics of a differential drive wheeled mobile robot.

A series of tests were performed to benchmark the wheelchair's system capabilities and to test the interfaces of the sensors installed. An algorithm was designed for proximity mapping with LiDAR which was later integrated with the wheelchair's Robot Operating

System (ROS). Similarly, to test the system control a simple obstacle detection and avoidance system was created using Kinect (RGB-D camera) as the input sensor. A MobileNet-SSD [How17] and [Liu16] was implemented using python-caffe for fast and efficient detection of objects in the Kinect's frame. MobileNet-SSDs have proven to be useful for traffic density estimation, tracking the number of vehicles in every input image frame, as demonstrated in [Bis18].

2.1.4.1 Obstacle Detection and Avoidance with Kinect

The RGB and Depth images provided by Kinect are with the help of a computer vision algorithm that detects objects in Wheelchair's vicinity in real-time. For the Kinect, a library compatible with ROS called Openni_Launch was used. This publishes RGB and depth images as ROS messages and can be used by various ROS nodes running simultaneously. A detailed sketch of ROS implementation of the node has been shown in Fig.2.2.

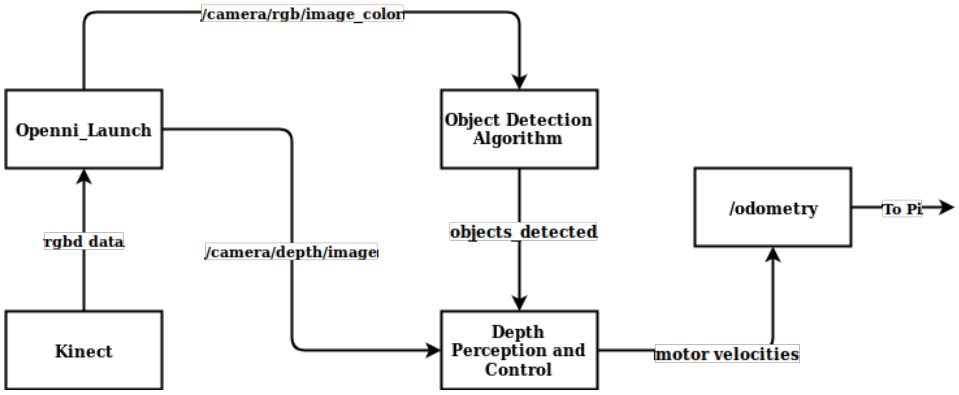


Figure 2.2 Structure of Kinect node in ROS

- **Object Detection**: A custom ROS node was built for the object detection algorithm. This node subscribed to the RGB images published as ROS messages to the the

object detection module. This module incorporates a pre-trained MobileNet-SSD Convolutional Neural Network model. The model detects objects in every incoming frame and generates bounding boxes around these objects with labels for each object detected. After identifying the objects, a custom ROS message published the information about these bounding boxes to a ROS topic.

- **Depth Perception Control**: A separate ROS node subscribed to the incoming message with information on bounding boxes which are then mapped to the depth image and their distance from the wheelchair is estimated in the pixel space. The obstacle avoidance algorithm was executed as a function of this distance between the detected object and wheelchair. The closest object is dealt with for obstacle avoidance. A snippet of the object detection algorithm and its corresponding mapping on the depth image is shown in Fig. 2.3. The safe zone is represented by the grey box. If the obstacle is relatively towards left in the safe zone, the error is calculated as the distance (in pixels) between left edge of safe zone and right edge of bounding box. Similarly, the error is calculated for an obstacle if it is relatively to right side of the safe zone.

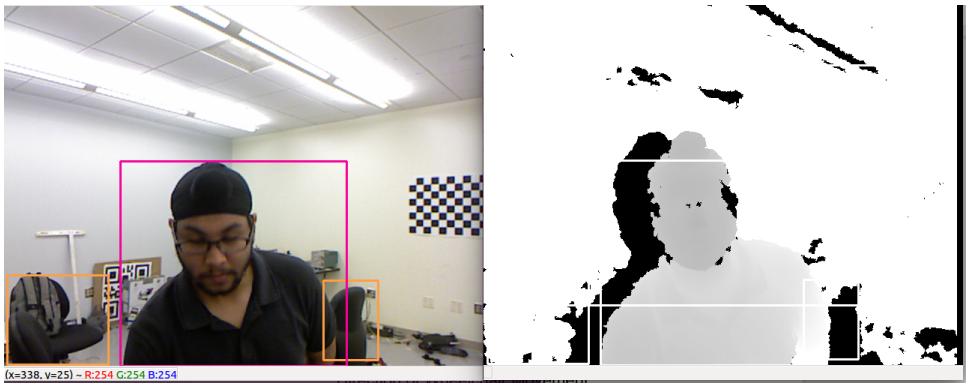


Figure 2.3 Object Detection in RGB image and corresponding Depth image

- **Proportional Controller**: The aim of the obstacle avoidance is to remove the closest object from a specific region (called as safe zone) of the camera frame, which eventually leads to planning the path around the detected object. The safe zone is defined as a region in pixel space where no object should be available in order for safe passage of the wheelchair. The obstacles are prioritized based on the closeness to the wheelchair.

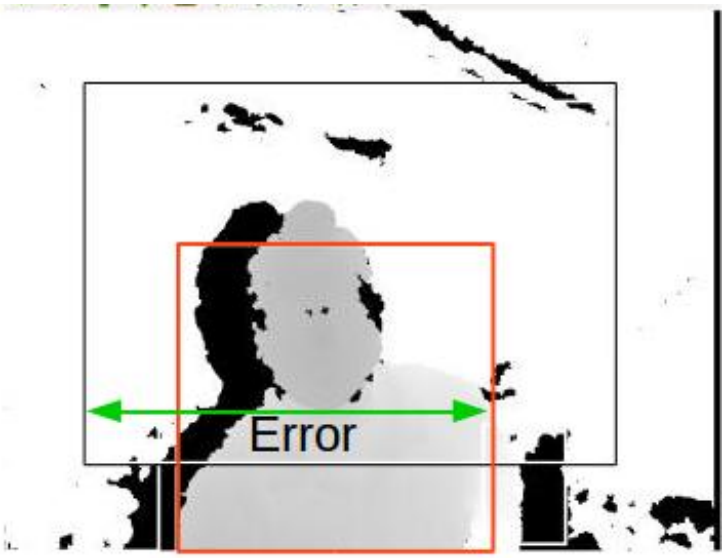


Figure 2.4 Error calculation for Depth Precision Control

The algorithm is based on Proportional Control with error determined by how much the obstacle is inside the safe zone. The obstacles in safe zone, in a depth image, are shown in Fig.2.4. The center of the obstacle is calculated. If the obstacle is relatively in the left side of the safe zone then the error will be positive as shown in (eq 2.6a). Similarly, (eq 2.6b) shows the calculation of error if the obstacle is relatively in the right side of the safe zone, which gives a negative error value. (eq

2.7a) and (eq 2.7b) present the velocity calculation for the right and left motors based on the calculated error.

$$e = obstacle\_right_{edge} - safe\_zone\_left_{edge} \tag{2.6a}$$

$$e = obstacle\_left_{edge} - safe\_zone\_right_{edge} \tag{2.6b}$$

$$v_r = (v_{ref} - k_p * e) * vel_{flag} \tag{2.7a}$$

$$v_l = (v_{ref} + k_p * e) * vel_{flag} \tag{2.7b}$$

where k_p is the proportional gain, e is the error calculated (in pixels) from depth image, v_{ref} is the set speed for wheelchair, v_r and v_l are the speeds calculated for right and left motors respectively, and vel_{flag} is the fail safe variable to bring the wheelchair to a complete halt. A positive error e (or obstacle on left) makes the wheelchair turn right and a negative error e (or obstacle on right) makes the wheelchair turn left.

2.1.4.2 Proximity Mapping with LiDAR

A 2D LiDAR and the rplidar_ros package was used to interpret and communicate LiDAR scan data between different ROS nodes. The package comes with a node for LiDAR which converts the raw LiDAR scan data to ROS messages. This ROS message consists of an array of values of distances between objects and LiDAR with a resolution of 1° in a 2D plane. Based on the orientation, the field of view for this application is between 0° and 180°. A point cloud is generated at every sampling instant of only the objects which are

closer than the threshold distance from the wheelchair, i.e, 2.5m.

- **Pre-processing of LiDAR Scan Data**: Using the rplidar ROS package, a laser scan is received with values of distance between objects and LiDAR in the range 0° to 360°. After applying a median filter to the raw scan data, an array of 360 accurate distance values, d (in meters) between wheelchair and any obstacle in the vicinity is generated. These values are used in the algorithm.

$$d(k,t) = median(d(k-i, t-j)) \tag{2.8}$$

  where t is the sampling instant, $k \in [0°,360°]$ is the current angle in scan at t. i, j are integers where $i \in [-2,2]$, $j \in [0,4]$. Here, scan at each angle k at sampling instant t is obtained after by considering the scan data at previous few sampling instants $(t-1, t-2, ...)$ and scan values at t at neighboring angles of k $(..., k-2, k-1, k+1, k+2, ...)$. The median of these values is taken and averaged over a few previous sampling instants to give out the filtered scan data.

- **Proximity mapping using LiDAR** LiDAR is mounted at a height on the wheelchair so that it can scan moving objects such as humans, and the other static entities in a enclosed space such as walls of a corridor. The height was chosen such that the LiDAR became the highest sensor on the wheelchair, so that it does not detect the patient, or the structure of the wheelchair itself. At each sampling instant, the filtered scan data is transformed into a 2D point cloud. This point cloud carries the information of the obstacles in the vicinity which are at a distance of 2.5m or less from the wheelchair in any direction. The point cloud is then used to determine the clearance between every two consecutive objects, as in [PN05]. If the computed

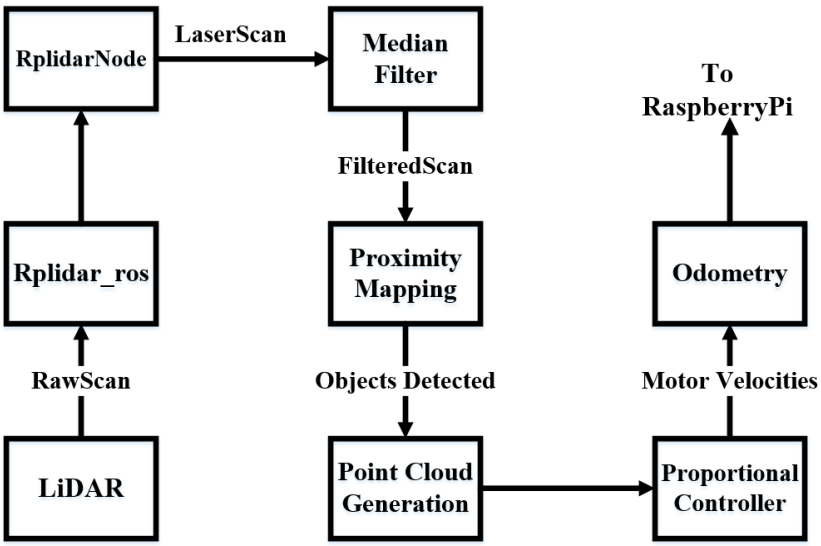


Figure 2.5 ROS node for LiDAR and Proximity Mapping algorithm

clearance is enough for the wheelchair to pass, a driving angle is calculated and appropriate velocity commands are generated for left and right wheels. In the case of multiple viable directions, i.e., when more than one clearance is detected, the algorithm chooses the angle which proves cheaper, i.e., where the difference between the current direction and new direction is least. If the clearance is not enough, the wheelchair is halted immediately. Variable "vel_{flag}" is used for this purpose which is set to 1 if clearance is good enough, otherwise 0. This flag provides an extra fail-safe feature to the wheelchair. A simplified blueprint of the LiDAR node structure and the proximity mapping algorithm is shown in Fig.2.5. If Θ is the list of plausible directions for the wheelchair and Θ_{curr} is the current direction of wheelchair motion, then an updated direction angle for wheelchair can be expressed as:

$$\Theta_{new} = \frac{\Theta_i - \Theta_{i+1}}{2}, \Theta_{new} \in \Theta : \Delta\Theta_i = min(\Delta\Theta) \tag{2.9}$$

where Θ_{i+1}, Θ_i are boundary angles of the clearance zone i (w.r.t. zero reference of LiDAR), $\Delta\Theta = \Theta - [\Theta_{curr}]$. Once $\Delta\Theta_i$ is calculated, the error for proportional controller can be calculated as described in (5).

$$e = \frac{R * \Delta\Theta_i}{W} \tag{2.10}$$

where R is the radius of wheels; W is the distance between wheel bases. The calculation of velocities can be written as (6a) and (6b).

$$v_r = (k_p * e + v_r ef) * vel_{flag} \tag{2.11a}$$

$$v_l = (k_p * e - v_r ef) * vel_{flag} \tag{2.11b}$$

where k_p, e, $v_r ef$, v_r, v_l and vel_{flag} retain their usual meanings. Since 90° of LiDAR scan aligns with direction of motion of the wheelchair and according to the zero reference chosen for LiDAR, any Θ beyond 90° represents turning left, while any Θ less than 90° represents turning right.

2.2 Related Work

Following many solutions provided for the SLAM problem, studies have been conducted to compare the same. [Kas18] provides a performance analysis and benchmarking of two popular visual SLAM Algorithms: RGBD-SLAM and RTABMap. The evaluation metrics used for the comparison are Absolute Trajectory Error (ATE) and Relative Pose Error (RPE). The analysis involves comparing the Root Mean Square Error (RMSE) of the two metrics and the processing time for each algorithm. The time taken by RTABMap to

process the sequence is less than the time taken by RGB-D SLAM since RTABMap fails to recover from the tracking problem and does not process the remaining sequence thus resulting in a shorter processing time. [Hes16] compares a custom LiDAR based scan-to-scan real time loop closure algorithm with already existing algorithms like Cartographer and Gmapping in terms of loop closure precision, and absolute and relative error in mapping and localization.
Recently, [Fil17] defined the metrics to evaluate five 2D SLAM algorithms. The metrics defined were:

- The proportion of occupied and free cells in the walls of the built map.
- The amount of corners in a map, which measures the precision of a map. A low quality map will have more corners than an accurate one. This is caused by overlapping of some parts of the map.
- The amount of enclosed areas, which is based over a similar idea that is if there are overlapping parts this metric can be used to detect them.

Further, [Kam] presents a performance analysis of two open-source, laser scanner-based Simultaneous Localization and Mapping (SLAM) techniques (i.e., Gmapping and Hector SLAM) using a Microsoft Kinect to replace the laser sensor. The results indicate that certain modifications to the default laser scanner-based parameters were able to improve the map accuracy. However, the limited field of view and range of the Kinect's depth sensor can causes map inaccuracies, especially in featureless areas, therefore the Kinect sensor is not a direct replacement for a LiDAR laser scanner. LiDAR offers a feasible alternative for 2D SLAM tasks only. Chapter 5 discusses more about the metrics used for evaluation of the various SLAM algorithms being compared in this thesis.

CHAPTER

3

SYSTEM OVERVIEW

This chapter will help the reader understand the architecture of the developed system. Here, we talk about the sensors used, and their location on the wheelchair. We also talk about the hardware and the software stack of the system, and how ROS (Robot Operating System) was used to build a sophisticated system with synchronized end-to-end communication between sensors and actuators. Further, the chapter also explains in detail how the wheelchair follows the kinematics of a differential drive wheeled mobile robot.

3.1 The Test Bed - Wheelchair

Figure 3.1 Jazzy select Wheelchair - modified

The wheelchair is a Jazzy Select series electric powered wheelchair as shown in Fig. 3.1. The wheelchair came pre-equipped with two brushed-DC motors, an ECU (Electronic Control Unit), and a joystick. The manufacturer-installed ECU was replaced by a Raspberry Pi. A motor driver was interfaced with the Raspberry Pi to control the speed of the motors using PWM signals. Additionally, an RGBD camera (Kinect), and a LiDAR were added to facilitate the perception of the wheelchair's environment. Having two drive

wheels, i.e., one on either side, the wheelchair's kinematics are that of a differential drive Wheeled Mobile Robot (WMR). Following, we will discuss each sensor and component individually.

3.1.1 Kinect

The Kinect is an RGB-D camera, where the 'D' stands for Depth. It is a composite device which has become an important 3D measuring device [cite 1]. Developed by Microsoft as an add-on for their gaming platform console XBox-360, used for recording hand gestures of the user (gamer), the device has gained popularity in robotics research as a readily available depth camera.

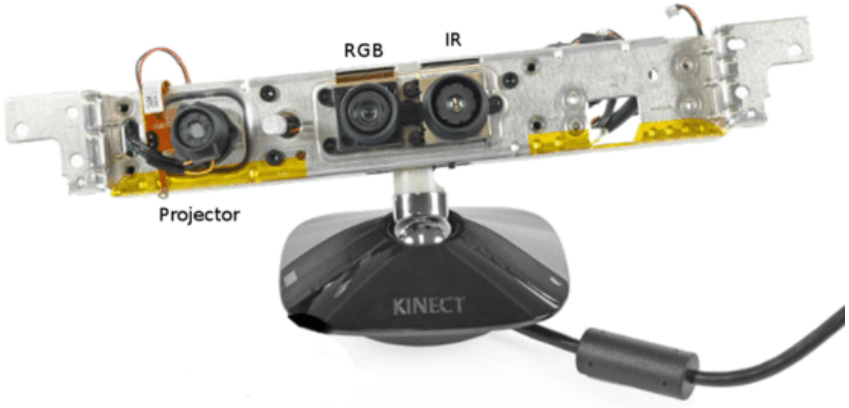


Figure 3.2 Kinect camera with IR Projector and sensor pair, and a regular color camera

As shown in Fig. 3.2, the device comprises of a regular color (RGB) camera, and an infra-red (IR) camera paired with an IR projector which is used for measuring depth. Some of the intrinsic properties of the RGB camera are mentioned below:
Focal Length: 2.9 mm.

Field of View (FOV): 63x50 (Horizontal x Vertical) degrees.

Pixel Size: 2.8 μm.

Max Resolution: 1280x1024 pixels.

3.1.2 LiDAR

A LiDAR (Light Detection And Ranging) is a device which, similar to a SONAR or RADAR, is used to map surroundings. The only difference is that instead of Sound or Radio waves, Light (Laser) is used. The LiDAR used in this research was a SLAMTEC RPLIDAR-A3 which is a 360° Laser Range Scanner, shown in Fig. 3.3. It is a unidirectional Laser (projector and sensor pair) rotating about a vertical axis giving an omnidirectional sense of measurement. The ability to rotate helps map the complete surrounding in range of the sensor. The sensor parameters are as follows:

Range: 25 m.

Samples per second: 16000.

The sensor provides a polar coordinates (angle, distance) of the measured points. These polar coordinates are converted into Cartesian (x, y) coordinates to be used for mapping, localization, and path planning.

3.1.3 Motors and Motor Driver

The motors used are the stock motors installed by the manufacturer of the Jazzy Select wheelchair. The motors are DC-brushed motors with gearbox attached. The gear ratio is 18:1 (motor to wheel). Both (left and right) motors operate at 24 volts, 10 amps (max).

The motor driver used is a Cytron MDD10A, which is a dual channel motor driver. It uses a H-Bridge circuit to drive 2 motors with a single power source of 24 volts. The motor

Figure 3.3 SLAMTEC RPLIDAR-A3

driver takes PWM inputs ranging from 0 to 100. The two channels have dedicated PWM inputs which control each motor individually.

3.1.4 Encoders

The encoders used are 2-phase rotary encoders with 600 pulses per revolution. Single phase encoders are useful to measure the rotational speed of the motor, while 2-phase encoders also tell the direction of rotation. The two channels, A and B, generate pulses in coordination such that the direction is determined by whether pulse A is leading pulse B, or vice versa. Having a sense of direction of rotation helps us determine the direction of wheelchair's motion.

The encoders are interfaced with an Arduino using interrupts. For every pulse generated by the encoders, an interrupt is raised and a pulse count is increased in the interrupt service routine. The number of pulses, along with the direction, is used to calculate the

angular velocity of the motor and hence the wheel which is further translated to linear velocity. This will be derived further in the Kinematics section of this chapter.

3.1.5 Inertial Measurement Unit (IMU)

IMU or Inertial Measurement Unit have become an essential part of the robotics industry. The aviation industry has been using IMUs for many years for measuring the orientation (roll, pitch, yaw) of airplanes, the linear acceleration, and direction wrt. EarthâĂŹs magnetic North. This sensor is used for inertial navigation technology in both airplanes and also in recently in some high end ground vehicles. An Inertial Navigation System (INS) provides the dead reckoning of a vehicle's position, velocity, and orientation. An IMU is used for vehicle motion control when its GPS satellite communication is lost for a certain period of time. For robot hobbyists, IMU is available in many variants including 6-DOF, 9-DOF, 10-DOF, and 11-DOF categories. Generally, IMU consists of 3 DOFs for accelerometer (x, y, and z axes), 3 DOFs for gyroscope which measures orientation wrt. axes x, y, z - pitch, roll, and yaw respectively. Further, 3 DOFs are for magnetometer, which is essentially a compass measuring the Earth's magnetic field and calculating the heading of the sensor (or the vehicle). Many manufacturers also provide barometer or pressure sensor, and a temperature sensor as extra degrees of freedom. The barometer is used to calculate the altitude, from sea level. The IMU we used is a 9-DOF Bosch BNO055 sensor. Following are the properties of the sensor taken from the datasheet available at [Sen]:

Absolute Orientation: Euler Vector, 100Hz

Absolute Orientation: Quaternion, 100Hz

Angular Velocity Vector: 100Hz, 3-axis rad/s

Acceleration Vector: 100Hz, 3-axis in m/s^2

Magnetic Field Strength Vector: 20Hz, micro Tesla (μT)

Linear Acceleration Vector: 100Hz, 3-axis m/s^2

Gravity Vector: 100Hz, 3-axis in m/s^2

3.1.6 Raspberry-Pi

Raspberry Pi is a mini computer which can run full Linux-based Operating Systems. Raspbian is an operating system, based on Debian (a Linux distribution), specifically designed for the Raspberry Pi. The Raspberry Pi was first introduced in 2011 and has since been through many revisions. We use the latest Raspberry Pi 3 model B+ which has a 1.4 GHz processor, BCM2837B0 - a 64-bit ARM SOC (system-on-chip) designed by Broadcom. The Raspberry Pi also comes with a 1GB SDRAM, and a micro-SD slot where the bootable micro-SD goes.
The Raspberry Pi is an excellent tool for systems development since it runs Linux and has: (1) 4 USB 2.0 ports, (2) an Ethernet jack, (4) a HDMI port, and (5) audio jack, Wi-Fi, and Bluetooth. The Raspberry Pi here was used to run ROS (Robot Operating System), which is a middleware with sophisticated message passing protocols. ROS will be discussed in more detail in a later section in this chapter.

3.2 Sensors Location and Coordinate Frames

The location and placement of sensors on the wheelchair is essential since it determines the accuracy in both perception and measurement. The location is in turn determined by the type of measurement, and range of the sensor. In the following subsections we will discuss the placement of each of the 4 sensors: Kinect, LiDAR, encoders, and IMU

along with their transform frames with respect to the wheelchair base. Associating a coordinate with every link of a robot is a conventional way to determine the position and orientation of the link. Two or more links are connected via a joint. Once the coordinate frames for the system are defined, the motion of each link with respect to the robot base (or base_link) can be determined easily with the help of transformation matrices. A transformation matrix is a useful tool that is composed of rotation matrix and translation matrix combinations. Below is an example of a transformation matrix defined as a transformation from an arbitrary frame A to an arbitrary frame B,

$$ {}^{A}T_{B} = \left(\begin{array}{ccc|c} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right) $$

The Fig. 3.4 shows the robot model with five coordinate frames superimposed on it, as an example. The Fig. 3.5 shows the coordinate frames with their respective names without the robot model. The robot model and the coordinate frames were generated using ROS URDF (Unified Robot Description Format) package to match the actual wheelchair. The five coordinate frames in the figure are, from top to bottom, laser, camera_link, right_-front_wheel, base_link, and left_front_wheel. More than these five frames were used in the actual processing and development of the system. These coordinate frames are used to define the transformations between various parts of the robot, which in turn helps determining the motion of individual links (parts) with respect to the robot base frame and also the world frame.

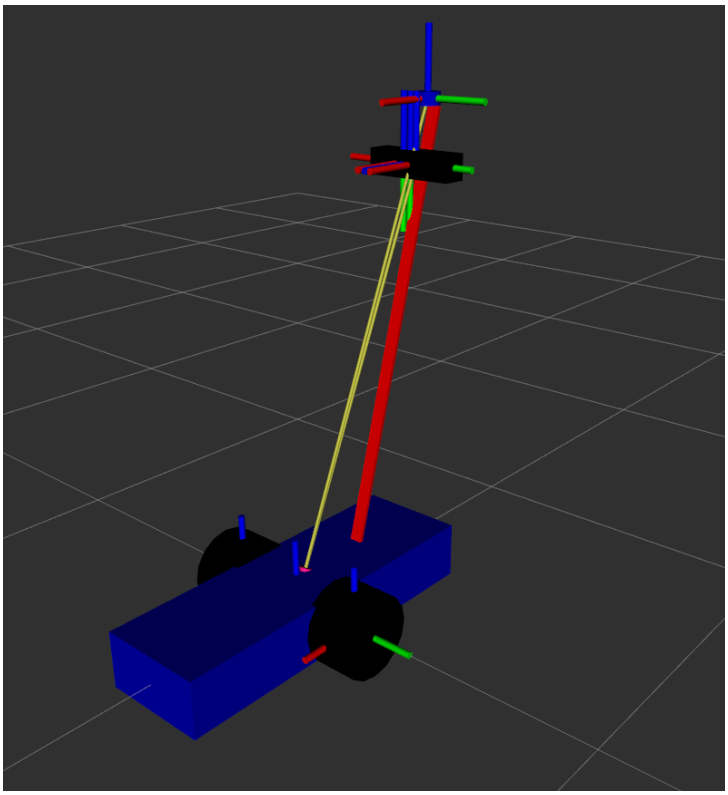


Figure 3.4 Wheelchair model developed in ROS

3.2.1 LiDAR (laser frame)

The LiDAR, in this project, is being used to obtain a 360Âř 2D map of the surrounding of the wheelchair, with particular attention being given to walls of the room or a corridor. Since the scanning requirement is 360Âř, the LiDAR could not be placed at the bottom of the wheelchair. Similarly, the LiDAR could not be placed too high on the wheelchair, otherwise it would fail to detect any close obstacles. The trade-off determined that the LiDAR should be placed at 160 cm (5 ft 3 inches) above ground level, which is just below the average height of an adult woman in the USA. The average height of an adult male in the USA is 175.3 cm (5 ft 9 inches), which will be detected by the LiDAR placed at 160 cm.

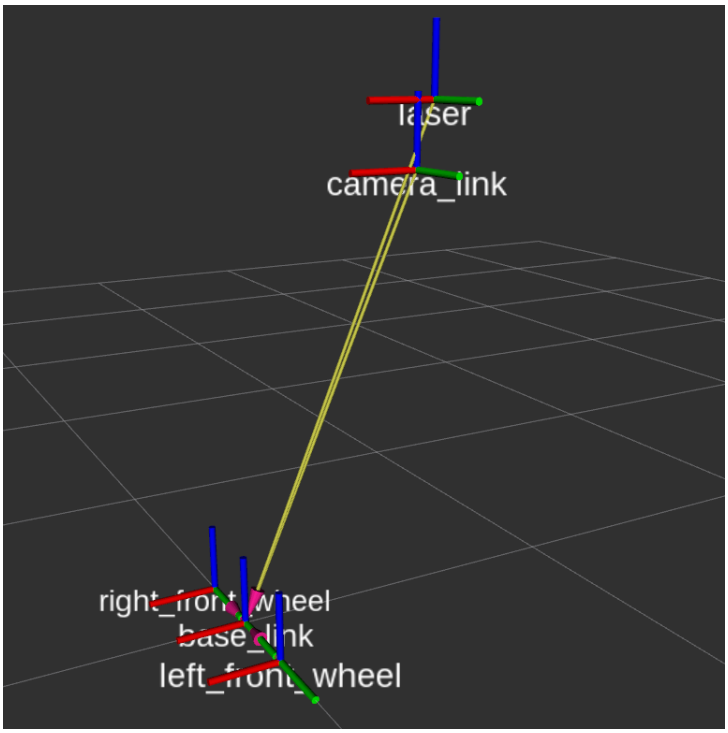


Figure 3.5 Coordinate frames of the Wheelchair

$$ {}^{base\_link}T_{laser} = \left(\begin{array}{ccc|c} 1 & 0 & 0 & -0.6096 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1.473 \\ \hline 0 & 0 & 0 & 1 \end{array}\right) $$

Where, the rotation matrix is simply and identity matrix which implies no rotation. There is translation along negative x-axis and negative z-axis (unit in meters).

3.2.2 Kinect (camera_link)

The Kinect camera, in this project is being used for visual SLAM implementation. For the same reason, it needs to be facing forward to capture the surroundings as the wheelchair

moves. Intuitively, the Kinect became the primary vision sensor for the wheelchair, it was eventually placed at the average human eye level, which is about 137.2 cm (4.5 ft) above the ground. The placement also need to take into account the average sitting height of a person so that the head of the wheelchair user does not obstruct the KinectâĂŹs view. The corresponding transformation matrix is:

$$ {}^{base\_link}T_{camera_link} = \left(\begin{array}{ccc|c} 1 & 0 & 0 & -0.5334 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1.2446 \\ \hline 0 & 0 & 0 & 1 \end{array}\right) $$

Where, the rotation matrix is simply and identity matrix which implies no rotation. There is translation along negative x-axis and negative z-axis (units in meters).

3.2.3 Encoders (wheel frames)

The encoders used to measure the rotational rate (rpm) of each drive wheel have their coordinate frames coinciding with the wheel axes. There were two possible locations for the encoders to be placed at: (1) at the wheel that is after the gear reduction, and (2) at the motor shaft before the gear reduction. Given the gear ratio of 18:1 (motor to wheel), and 600 pulses per rotation for the encoder, one full wheel rotation will generate 10,800 (= 600*18) pulses. Since, we know the wheel diameter to be 10 inches, for one full rotation the wheel covers approximately 0.8 meters (= 10*0.0254*3.14). Further, this information can be used to calculate the resolution of the encoders to be equal to 7.4 x 10-5 which is very small and thus provides better measurement precision.

3.2.4 Inertial Measurement Unit (IMU)

The Bosch BNO055 055 IMU was placed midway between the two drive wheels. The intuition behind this specific placement is that the IMU needs to capture linear velocity, which is the resultant vector of the individual velocities of the right and the left wheel. The IMU must also capture the orientation of the wheelchair in the space, so placing it midway between the right and left drive wheels coincides with the x-axis of the IMU and the base_link. In doing this the y and z axes of the IMU are parallel to the y and z axes of the robot base. The transformation matrix for the same is:

$$ {}^{base\_link}T_{imu\_link} = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0.3302 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right) $$

3.3 ROS - Robot Operating System

ROS or Robot Operating System, unlike the name suggests is not an operating system. ROS is a middleware which acts as a bridge between the operating system and application level programs. ROS was developed to sophisticate the hardware and software architecture of the robot. ROS provides a simple yet elegant message packing and transportation service which helps to pass around sensor data to more than any one processes that require that data for robot control. Following are a few terminologies that are required to understand ROS:

- **ROS Node**: It is an executable code or program that runs in the background. A ROS

node can contain a publisher, or a subscriber, or both.

- **ROS Topic**: A ROS Topic can be thought of as a newspaper column with a single dedicated editor (Publisher) and many readers (Subscriber).
- **Publisher**: A ROS Node may Publish a ROS message to a ROS Topic.
- **Subscriber**: A Subscriber subscribes to a ROS Topic and stays up-to-date with the incoming ROS messages.
- **ROS Message**: A ROS message is broadcasted by the Publisher to a ROS Topic. It can be composed of sensor readings, or control parameters, or any data in any format.

Fig. 3.6 shows how the hardware components are connected and interacting with each other. The direction of arrows represent the data flow.

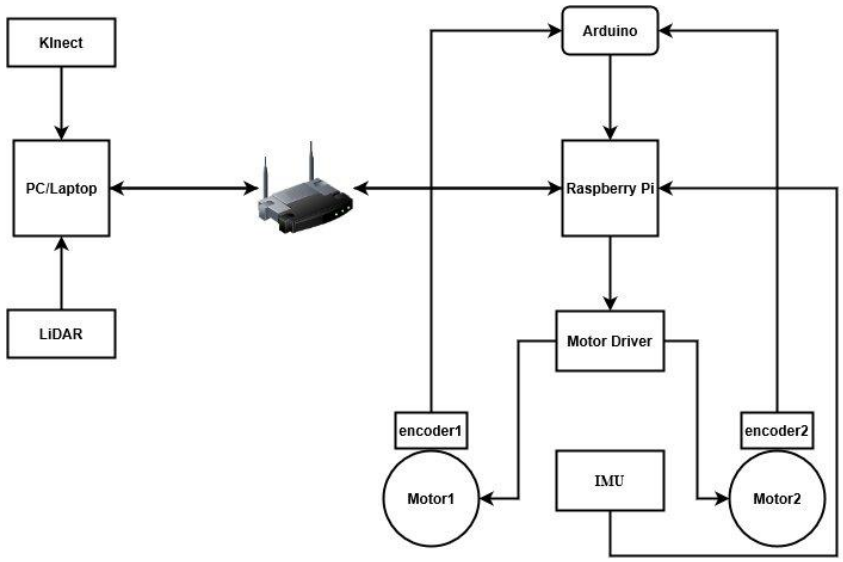


Figure 3.6 Hardware architecture and data flow

ROS follows a standard called URDF for generating robot models. URDF stands for Unified Robot Description Format which uses static transformations between frames and continuously publishes the same. Using robot model made shown in Fig. 3.4, ROS generates a transformation tree also called *tf_tree* which shows the links and latest transformations (if any) published between these frames. Fig. 3.7 shows such a tree generated.

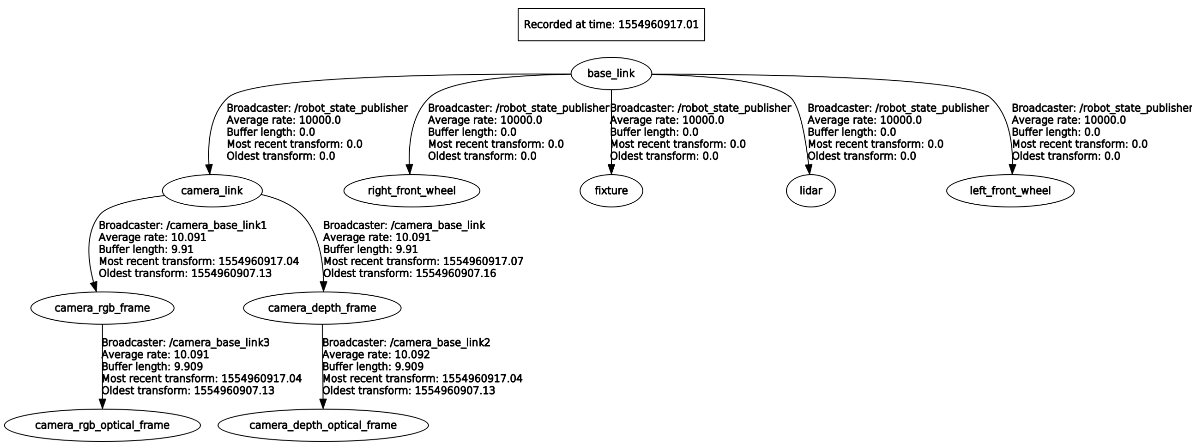


Figure 3.7 TF-tree generated using robot model in ROS

The next chapter will discuss how all these components, sensors and actuators, interact with each other with the help of ROS for SLAM implementation. Further, we will discuss each SLAM algorithm in detail.

CHAPTER

4

SLAM

For SLAM to work, data from the environment (exteroceptive) as well as data from the robot (proprioceptive) must be present. Various solutions make use of exteroceptive sensors such as LiDARs, cameras, sonars to generate meaningful data from the environment. Further, proprioceptive data is often generated using wheel encoders or a sensor fusion of acceleration and orientation data from an IMU. It is important to make sure that the SLAM algorithm is receiving and processing all of this data. This project focuses on comparing SLAM algorithms that make use of a LiDAR, camera, IMU, and wheel encoders to provide exteroceptive and proprioceptive data. The exteroceptive sensors are used for finding landmarks in the environment which is later used to build a map, by learn-

ing, and also used to localize within the map that was built. Once new landmarks are detected and processed, the SLAM algorithm attempts to associate these new landmarks with any previously observed landmarks. Landmarks can act as obstacles and obstacles can act as landmarks. There are times when the SLAM algorithm is localizing and mapping landmarks in the real-time path planning of the robot. In such a planning activity, obstacles and landmarks are synonymous.

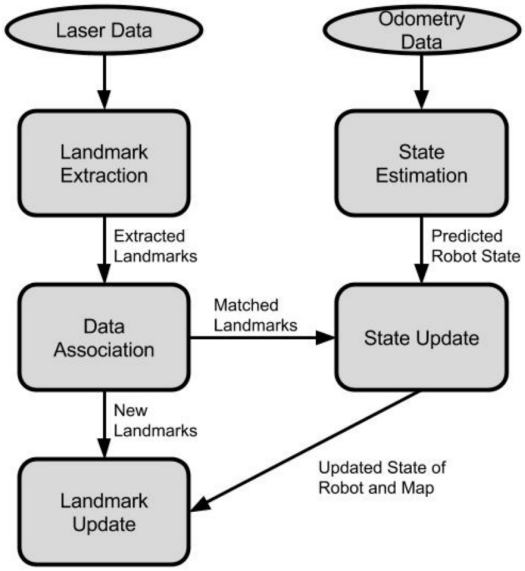


Figure 4.1 Flowchart of a basic LiDAR based SLAM

As the robot continues to explore and scan it's environment, the SLAM algorithm uses odometry data to estimate the robot's current position within the map. This estimation is compared to readings of the environment, obtained from sensors, and to facilitate corrections if these reading are proved inaccurate. The process is continued until a complete map of the environment is created. A basic LiDAR based SLAM algorithm can be divided into five main parts: landmark extraction, data association, state estimation,

state update, and landmark update [RB04]. A description of this process is visually represented in Fig. 4.1.

In this chapter we will discuss in brief five SLAM algorithms namely GMapping, Hector SLAM, RTAB-MAP, RGB-D SLAM, and VINS Mono of which the latter three are camera-based visual SLAM algorithms.

4.1 LiDAR based SLAM

4.1.1 Gmapping

OpenSLAM's Gmapping is the most used LiDAR based SLAM algorithm which follows an efficient implementation of Rao-Blackwellized Particle Filter (RBPF) [Gri05] and [Gri07]. Each particle is a sample of history of robot poses and posterior over maps given the sample pose history, i.e., each particle carries an individual map of the environment based on the recent observations. The primary objective is to reduce the number of particles for which an adaptive technique is implemented. Instead of completely relying on the movement of the robot (history of robot poses), recent observations are used to correct the generated map.

Previously, [Mur00] compared two algorithms viz., online EM (Expectation-Maximization) and Bayesian inference. In the online EM, the map was considered to be a fixed parameter whereas, in the Bayesian inference, the map was considered as a matrix of random variables. It was shown that the online EM algorithm gets easily stuck in a local minima, which causes the robot to be considered as lost. On the other hand, Bayesian inference, where multiple hypotheses are maintained, proved to be more robust, and a Rao-Blackwellized particle filter method is introduced to approximate the map. [Gri05]

continue the work on the same to make the approximation even faster by reducing the number of particle and keeping accuracy intact.
The Algorithm is summarized as follows in [Gri07]:

1. $x_t'^{(i)} = x_{t-1}^{(i)} \oplus u_{t-1}$ is the initial guess of the robot's pose represented by the particle i. This is calculated from the previous pose $x_{t-1}^{(i)}$ of that particle and the odometry measurements u_{t-1} collected since the last filter update. Here, the operator $\oplus$ is the standard pose compounding operator

2. Starting from the initial guess $x_t'^{(i)}$, a scan-matching algorithm is executed based on the map $m_{t1}^{(i)}$. The search is bounded to a limited region around $x_t'^{(i)}$. In case an failure is reported, the pose and the weights are calculated using the motion model and steps 3 and 4 are ignored.

3. In an interval around the pose $\hat{x}_t^{(i)}$ reported by the scan-matcher, a set of sampling points is selected. Based on these points, the mean and the covariance matrix of the proposal are calculated by evaluating the target distribution $p(z_t m_{t1}^{(i)}, x_j) p(x_j x_{t1}^{(i)}, u_{t1})$ in the sampled positions x_j. During this phase, the weighting factor $\eta^{(i)}$ is calculated.

4. The Gaussian approximation $N(\mu_t^{(i)}, \Sigma_t^{(i)})$ of the improved proposal distribution is calculated and the new pose $x_t^{(i)}$ of the particle i is drawn.

5. The importance weights are updated.

6. According to the drawn pose $x_t^{(i)}$ and the observation z_t, the map $m(i)$ of particle i is updated .

This SLAM algorithm is available as a black-box package in the ROS environment. The algorithm is tested with and without the provision of odometry. The setup and results of testing are provided in the next chapter.

4.1.2 Hector SLAM

Hector SLAM is LiDAR based 2D mapping and pose estimation technique available as an open source implementation as proposed in [Koh11]. Unlike Gmapping, Hector SLAM is not based on Rao-Blackwellized Particle filter for mapping and localization and further, it does not require wheel odometry information. Hector SLAM uses a Guass-Newton approach for the scan matching as proposed in [LK81]. The 2D pose is estimated using the scan matching process alone.

The algorithm tries to find an optimum alignment of the laser scan's endpoints with the already constructed map by finding the rigid-body transformation $\xi^* = (p_x, p_y, \psi)^T$ and minimize

$$\xi^* = \arg\min_{\xi} \sum_{i=1}^{n} [1 - M(S_i(\xi))]^2 \tag{4.1}$$

where, the function $M(S_i(\xi))$ is the map value at $S_i(\xi)$ i.e. the world coordinate of the endpoint of the laser scan. Given a prior estimate of ξ, the step transformation $\Delta\xi$ is estimated by optimizing the error such that

$$\sum_{i=1}^{n} [1 - M(S_i(\xi + \Delta\xi))]^2 \to 0 \tag{4.2}$$

Fig. 4.2 provides an overview of the Hector algorithm stack. The preprocessed data from the LiDAR is forwarded to a scan matching algorithm which in this case follows a Gauss-Newton approach to optimize the alignment using eq. 4.1. The Hector SLAM is also

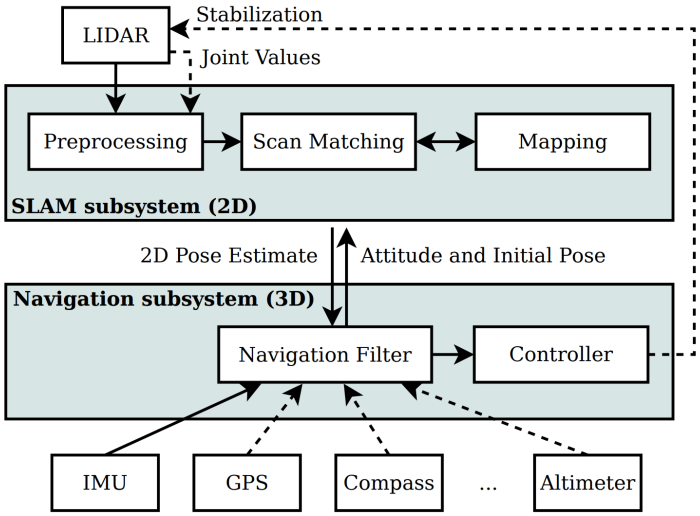


Figure 4.2 Overview of the Hector Mapping and Navigation stack

capable of 6 DOF (Degrees Of Freedom) pose (Attitude) estimation using an Extended Kalman Filter, since it was initially designed for search and rescue UGVs. In this thesis we only deal with 2D pose estimation and mapping.

4.2 Visual SLAM

4.2.1 RTAB MAP

RTAB-Map (or Real-Time Appearance-Based Mapping) is an RGB-D, Stereo and Lidar Graph-Based SLAM approach [LM19]. This approach is based on an incremental appearance-based loop closure detector which uses a bag-of-words approach and tries to estimate the likelihood that a new image comes from a previous location or a new location. Whenever a loop closure hypothesis is accepted, a new constraint is added to the map's graph, which is the map insertion step in a basic SLAM algorithm, as explained in detail at beginning of this chapter. A graph optimizer tries to minimize the errors in the

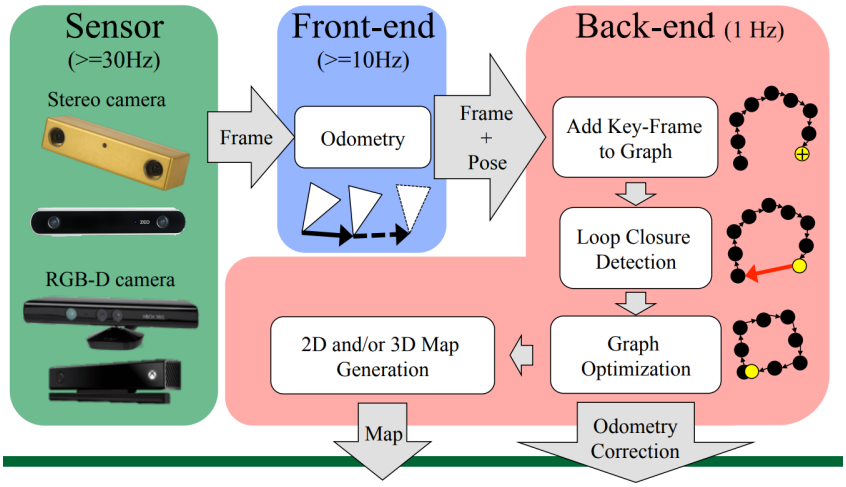


Figure 4.3 Flow of the RTAB Map

map. RTAB-MAP also accommodates a memory management approach [LM11] which is used to limit the number of locations used for the loop closure and graph optimization. This helps make the algorithm run faster on less powerful machines respecting the real-time constraints.

4.2.2 RGB-D SLAM

RGB-D SLAM uses both color (RGB) and depth images to produce map. The map generation and optimization follows four basic steps. First, visual features like SURF, SIFT, and ORB [Kar17] are extracted from the color image. Then these features are compared (or matched) with the features from previous frames. The location of the features in the color image is mapped to the corresponding location in depth image which gives a set of point-wise 3D correspondences between any two frames. The pose of the camera is obtained by evaluating these correspondences to get a relative transformation between the current and the previous frame using RANSAC [Hu12]. The Fig. 4.4 shows a flowchart

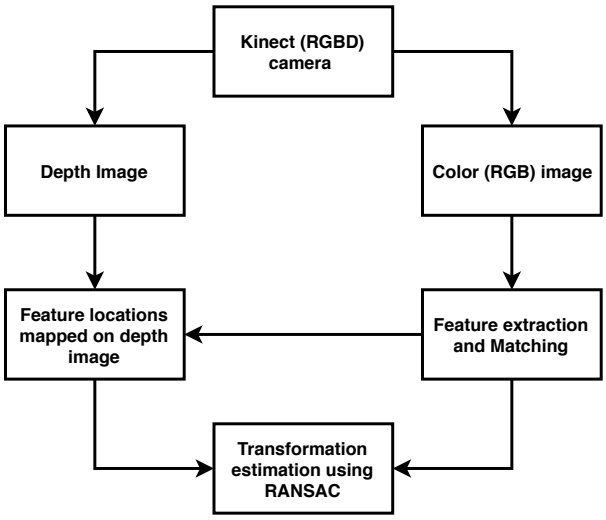


Figure 4.4 Flowchart for the RGBD Mapping

for the process.

4.2.3 VINS-MONO

VINS-Mono (or Visual Inertial Navigation System - Monocular) SLAM algorithm is based on Visual Inertial Odometry. Visual-inertial odometry (VIO) is extensively applied to state estimation problems in a variety of domains, e.g., autonomous vehicles, virtual and augmented reality, and UAVs. The research has reached a level of maturity that many commercial products now use proprietary VIO algorithms. Additionally, there are many open-source software packages available which provide off-the-shelf visual pipelines deployable on an end-user's system.

A simple yet efficient way to deal with visual and inertial measurements is a loosely-coupled sensor fusion [Wei12], [Lyn13], where data from IMU is treated as an independent module to assist pose estimation based on visual structure from motion. An Extended Kalman Filter is employed, where IMU is used for state propagation and the

vision-only pose is used for the update. Further, tightly-coupled visual-inertial algorithms are either based on the EKF [LM13] or graph optimization [She15], where camera and IMU measurements are jointly optimized from the raw measurement level.

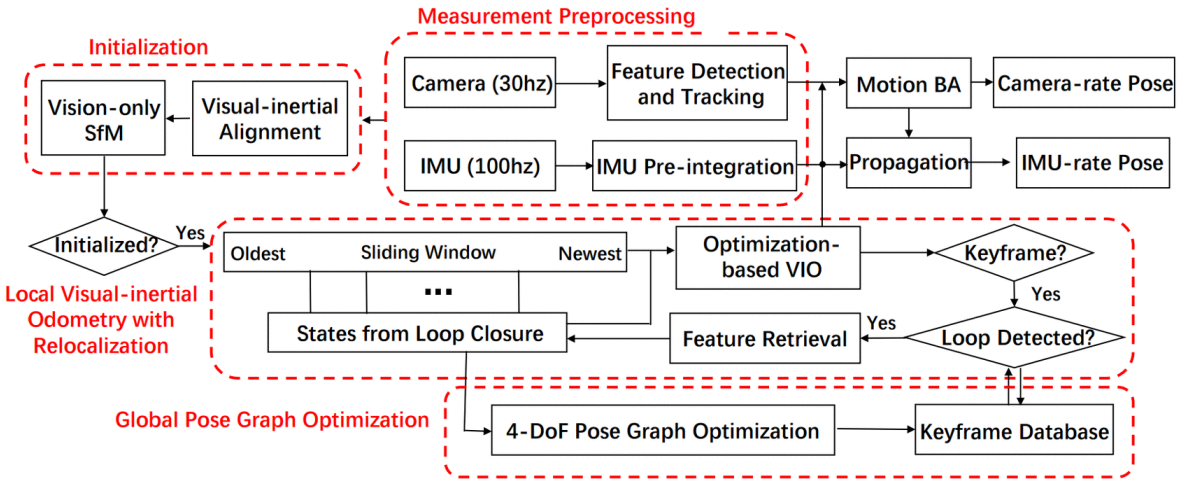


Figure 4.5 VINS Mono SLAM flow of control

VINS-Mono is a non-linear optimization-based sliding window estimator which tracks robust corner features [ST93]. Additionally, VINS-Mono introduces several new features to this class of visual-only pose estimation framework. The acceleration and orientation data from IMU is pre-integrated before being used in the optimization, and a tightly-coupled procedure for relocalization is implemented. VINS-Mono additionally features modules to perform 4DoF pose graph optimization and loop closure since it was originally designed for UAVs. An overview of the VINS-mono architecture is shown in the Fig. 4.5.

Following chapter discusses the experiments, and results for each of these SLAM algorithms.

CHAPTER

5

EXPERIMENTAL SETUP AND RESULTS

In this Chapter, we will walk-through the setup and testing of different SLAM Algorithms discussed in Chapter 4. A standard test set has been designed for the testing, and metrics have been defined which will compare the results from each SLAM. The test set includes

1. Mapping the the one chosen room, comparing the mapping accuracy based on the number of features learned and included in the map, and saving the map.
2. Retrieving the saved map and mapping the corridor outside the room to test and compare scan matching and map insertion.
3. Compare the generated trajectory, given a ground truth, when the wheelchair is

made to follow the ground truth trajectory.

An architectural plan of the NCSU Engineering Building II was acquired which is shown in Fig. 5.1. Additionally, the research inspects how the SLAM algorithms perform with and without the availability of odometry data since certain algorithms claim to work without the availability of proprioceptive data. A final result will be reported in the form of Table, at the end of the chapter, comparing different SLAM algorithms based on the metrics.

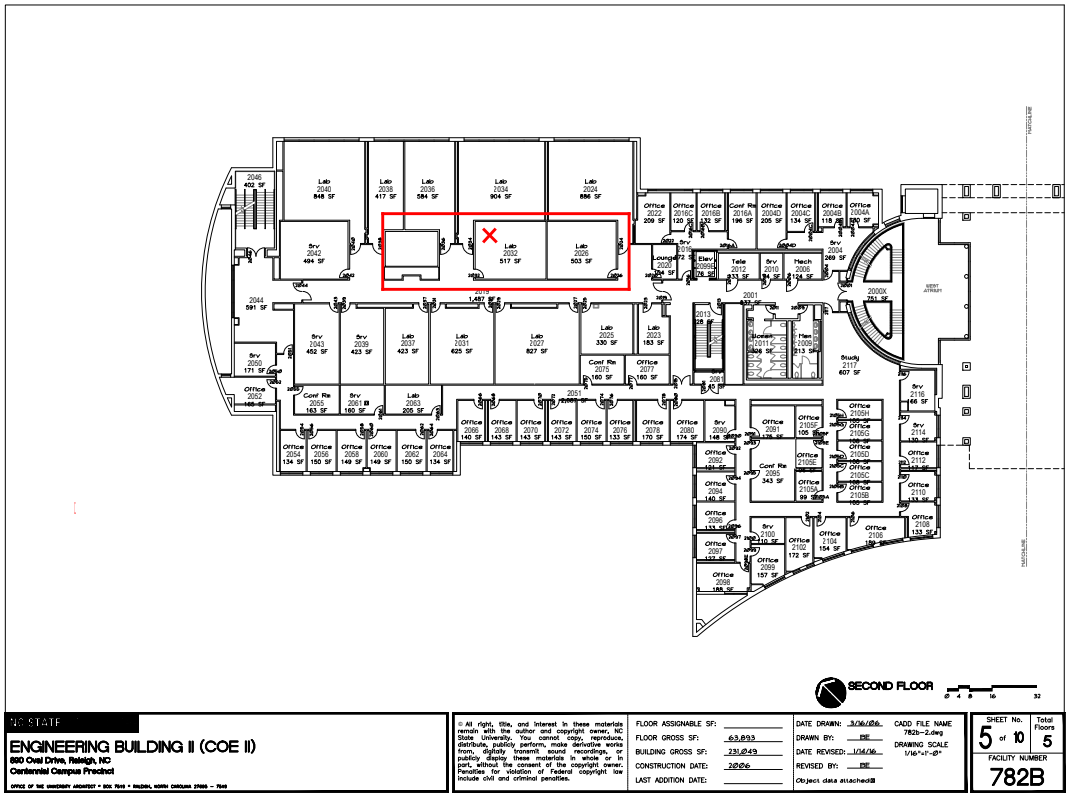


Figure 5.1

5.1 Metrics Used

5.1.1 Absolute Trajectory error

The global consistency of the estimated trajectory is an important quantity for a visual or LiDAR based SLAM system. The Absolute Trajectory Error (ATE) is evaluated by comparing the absolute distances between the estimated and the ground truth trajectory. A trajectory in this case is the history of robot's poses. Since, both estimated and ground truth trajectories can be specified in arbitrary coordinate frames, they first need to be aligned. This is done by finding a rigid transformation S corresponding to the least-squares solution that maps the estimated trajectory $P_{1:n}$ onto the ground truth trajectory $Q_{1:n}$. The absolute trajectory error at time step i can be calculated as

$$F_i = Q_i^{-1} S P_i \tag{5.1}$$

Evaluating the root mean squared error (RMSE) over all time steps of the translation components, we get,

$$RMSE(F_{1:n}) = (\frac{1}{n}\sum_{i=1}^{n} ||trans(F_i)||^2)^{1/2} \tag{5.2}$$

5.1.2 Relative Pose Error

The Relative Pose Error measures the local accuracy of the trajectory over a fixed time interval Δ. Thus, the Relative Pose Error is essentially drift of the robot's trajectory which is useful for the evaluation of SLAM systems. We define the relative pose error at time step i as

$$E_i = (Q_i^{-1} Q_{i+\Delta})^{-1} (P_i^{-1} P_{i+\Delta})^{-1} \tag{5.3}$$

From a sequence of n poses, we get m $(= n - \Delta)$ individual RPEs along the sequence. The Root Mean Squared Error (RMSE) was computed over all the time indices of the translation component as:

$$RMSE(E_{1:n}) = (\frac{1}{m} \sum_{i=1}^{m} ||trans(E_i)||^2)^{1/2} \tag{5.4}$$

where $trans(E_i)$ is the translation components of the Relative Pose Error E_i. For visual odometry systems that match consecutive frame, the time parameter Δ is usually set to be equal to 1 which is intuitive. $RMSE(E_{1:n})$ then gives the drift per frame. For systems that use more than one previous frame, larger values of Δ may be appropriate. On a frame sequence recorded at 20 Hz, a Δ of 20 gives the drift per second. Also, a common choice is to set $\Delta = n$ which means that the start point is directly compared to the end point. This metric can be misleading as it penalizes rotational errors in the beginning of a trajectory more than towards the end [Kum09] and must not be used. It therefore makes sense to average over all possible time intervals Δ, i.e., to calculate

$$RMSE(E_{1:n}) = \frac{1}{n} \sum_{\Delta=1}^{n} RMSE(E_{1:n}, \Delta) \tag{5.5}$$

Note that the computational complexity of this expression is quadratic in the trajectory length. Therefore, it is approximated by calculating it from a fixed number of relative pose samples. The RPE can be used to evaluate the global error of a trajectory by averaging over all possible time intervals. The RPE assesses both translational and rotational errors, while the ATE only assesses the translational errors. Therefore, the RPE is always

greater than the ATE (or equal if there is no rotational error). Thus, the RPE metric gives us a way to combine rotational and translational errors into a single measure. However, rotational errors are also indirectly captured by the ATE as it manifest itself in wrong translations. From a practical perspective, the ATE has an intuitive visualization which facilitates visual inspection. Nevertheless, the two metrics are strongly correlated.

5.2 Tests

A total of three tests are performed that provide the mapping accuracy, precision, and localization accuracy which is also the accuracy of the 2D pose estimation. To keep the test fair with minimal contamination, the result for each algorithm is compared to a ground truth. For test 1 and test 2, which deal with mapping, ground truth is available in the form of the architectural plan of the floor of the building. For test 3 ground truth is generated by maneuvering the wheelchair through a set track once and recording the data which is replayed for each of the SLAM algorithms.

5.2.1 Test 1 - Mapping the lab room

The first test focuses on testing the mapping accuracy. The algorithms were made to follow a set trajectory in a room such that the wheelchair moves through a certain environment only once and the data from LiDAR, camera, IMU and encoders was recorded. This data was replayed for each SLAM algorithm and the result was compared with the ground truth. Since the LiDAR used is a 2D LiDAR, it generates scan points in a horizontal plane which is at the height of the LiDAR. The LiDAR based algorithms generate a 2D map, whereas visual SLAM algorithms generate a 3D map of the environment. For fair comparison, a 2D projection of the generated 3D maps were taken into consideration.

Fig. 5.2 shows the ground truth map taken from the architectural plan of the building. The room with the red cross is the room that each SLAM algorithms is expected to learn and map.

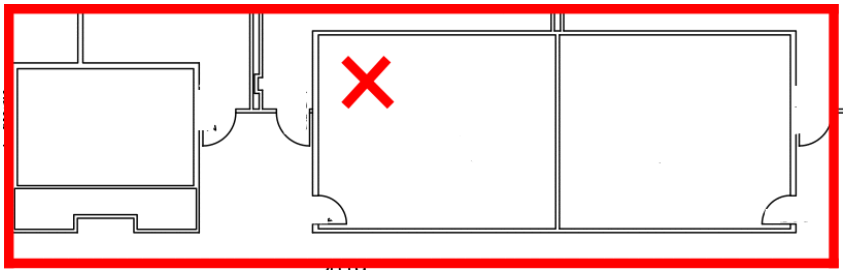


Figure 5.2

5.2.1.1 Gmapping

Gmapping is able to map the walls of the room accurately. Fig. 5.3 shows the output of the Gmapping when the wheelchair has been around the room exactly once. The dark grey is the unmapped area, while the light grey area is the free space where the wheelchair can move, according to the algorithm. The black area/points represent the features (or landmarks) learned from the environment.
Note that there are discontinuous blacks points in the middle of the room. These are the points from the desks in the room and should have formed a straight continuous line. Gmapping failed to register the desks even though they were in the observable range of the LiDAR's location. The red line marks the trajectory estimated by the algorithm as the wheelchair moves through the environment. The $base\_link$ in the figure is the current estimated location of the wheelchair in the map, and the map point represents the origin of the map i.e., the point where first LiDAR scan was registered.

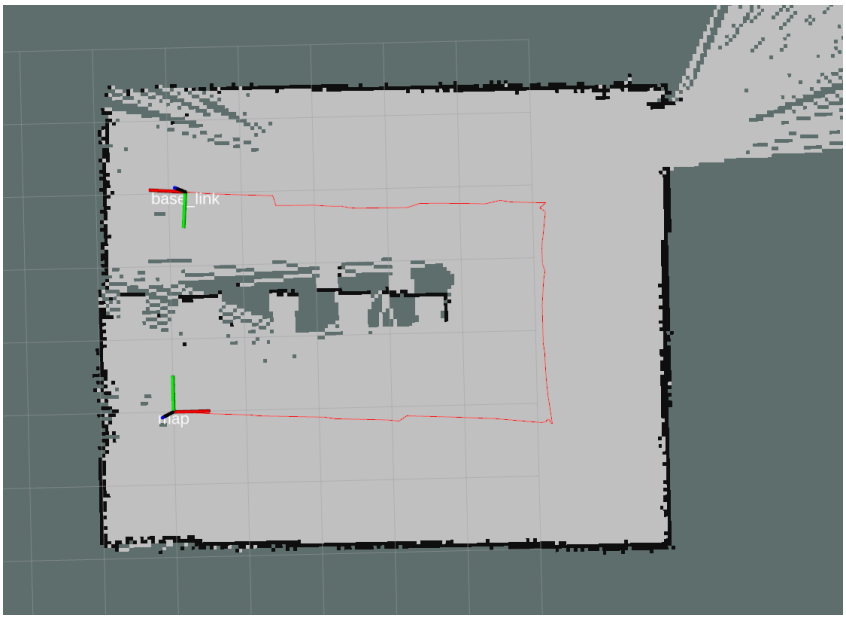


Figure 5.3 Mapping the lab room with Gmapping

5.2.1.2 Hector SLAM

The test was repeated with Hector SLAM and the result is shown in figure 5.4 Since we know Hector SLAM relies on scan matching for learning new landmarks and localization (2D pose estimation), we would expect the map insertion to fail in an environment where the landmarks are non-unique. This will be more evident in test-2.
For test 1, Hector was able to map the lab room accurately. We can see some rotational drift in Fig. 5.4 where the algorithm performed an incorrect map insertion at some angle relative to the actual map. Hector was able to correct the map as soon as it processed the subsequent scan matching.

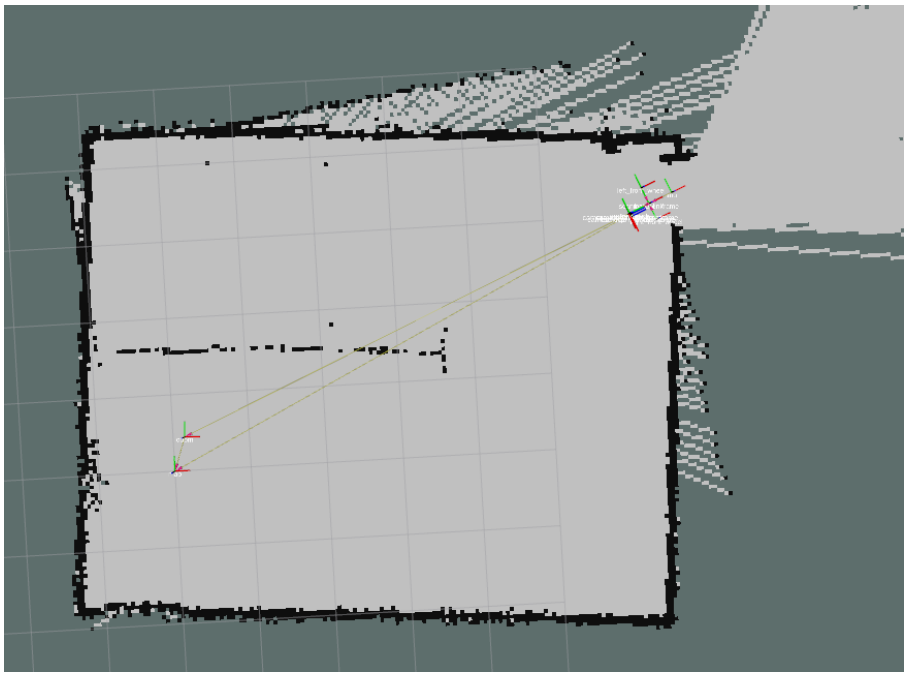


Figure 5.4 Mapping the lab room with Hector SLAM

5.2.1.3 RTAB-MAP

RTAB-MAP is a visual SLAM algorithm which uses a Monocular (RGBD) or Stereo camera to generate maps. Unlike LiDAR which maps 360°, the camera is only front facing and would require the wheelchair to rotate 360° to map the complete environment. Despite this, we keep our test data unchanged because we need all the algorithms to have a fair chance. The same data is replayed for the algorithm where the wheelchair went around the room once. Fig. 5.5 as a 3D map of the lab and Fig. 5.6 shows the 2D projection of the same.

In Fig. 5.6, we observe that there is a missing rather unmapped section towards the bottom left. This is also observable in Fig. 5.5. This is due to field of view of the camera at the start of the image sequence. The initial position of the wheelchair is marked by the

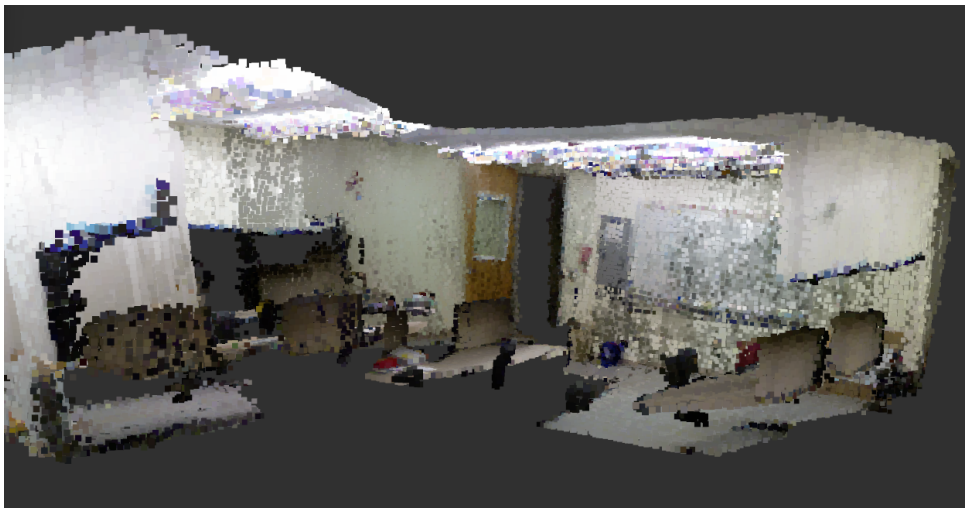


Figure 5.5 Mapping the lab room with RTAB-MAP - 3D

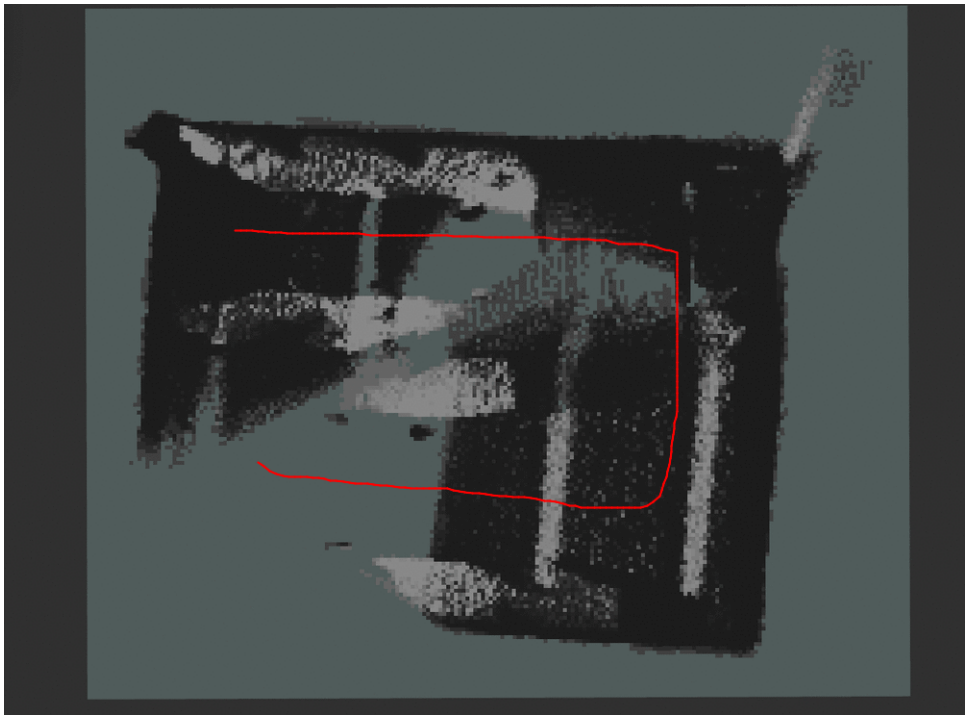


Figure 5.6 Mapping the lab room with RTAB-MAP - 2D

starting of the red line which is the estimated trajectory of the wheelchair. The initial pose of the wheelchair was estimated using the depth image by computing the distance

from the registered features in the first frame of the image sequence.

5.2.1.4 RGB-D SLAM

For RGB-D SLAM, only Kinect was used. It uses the feature matching technique to calculate the transformation between keyframes. Depth images are used to calculate distances from these features and the pose of the camera is calculated. Similar to RTAB-MAP and VINS-Mono, the mapping is constrained by the field of view of the Kinect Camera which is shown in Fig. 5.7. The generated map is similar to the map generated by RTAB-MAP since both of these algorithms use similar features - SIFT, SURF, ORB, etc. RTAB may use all these features together while RGB-D may use only one of the features during a run of the algorithm. The difference can be seen in the granularity of the two maps. Map generated by RTAB-MAP is more finer than the map generated using RGB-D which looks distorted.

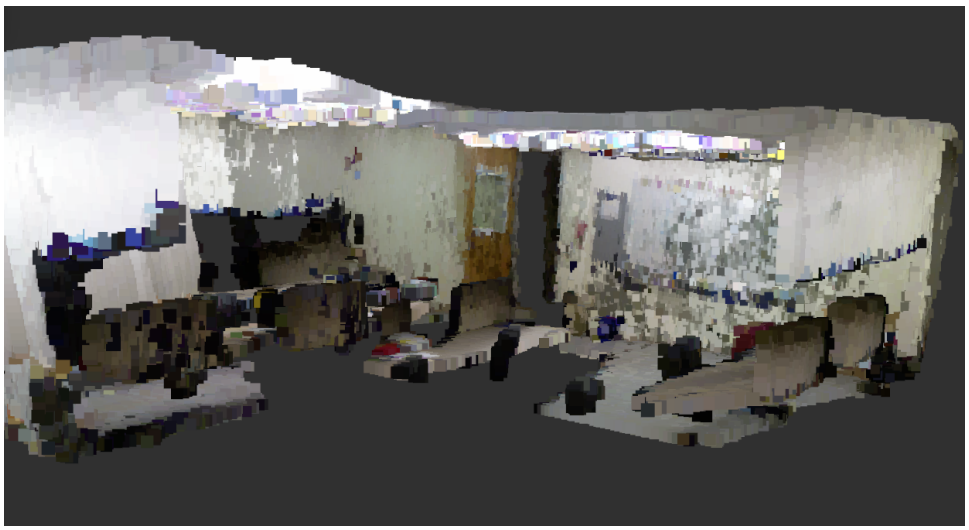


Figure 5.7 Mapping the lab room with RGB-D SLAM - 3D

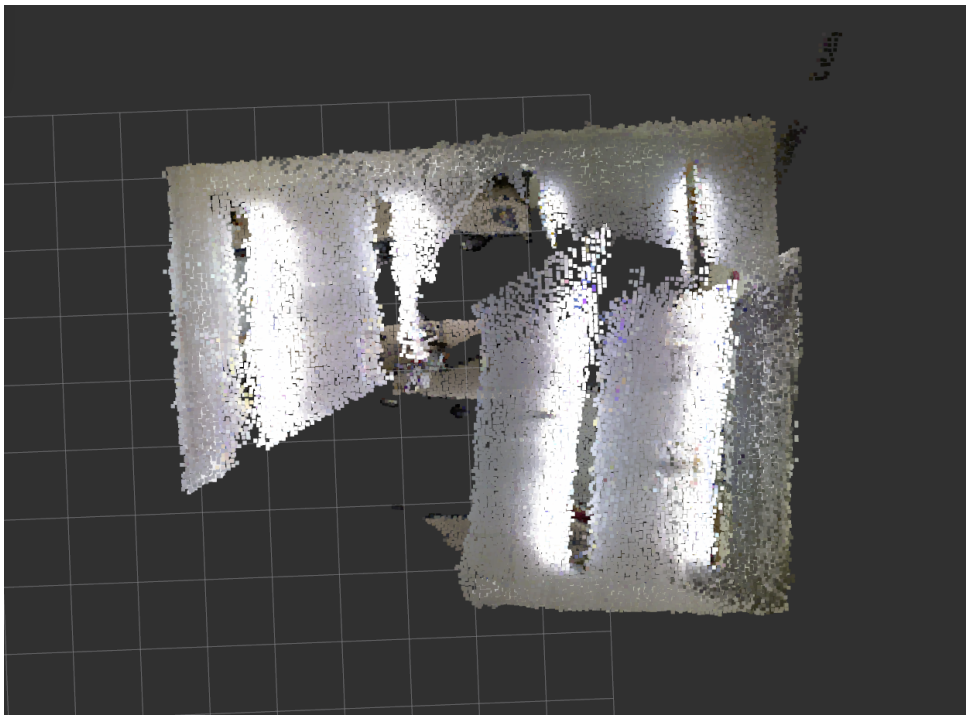


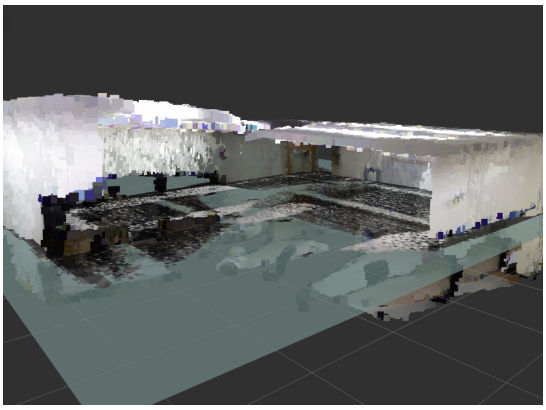
Figure 5.8 Mapping the lab room with RGB-D SLAM - 3D Top View

A top view of the RGB-D map is shown in Fig. 5.8. The inaccuracy in mapping is evident where a part of the map is overlapped with another.

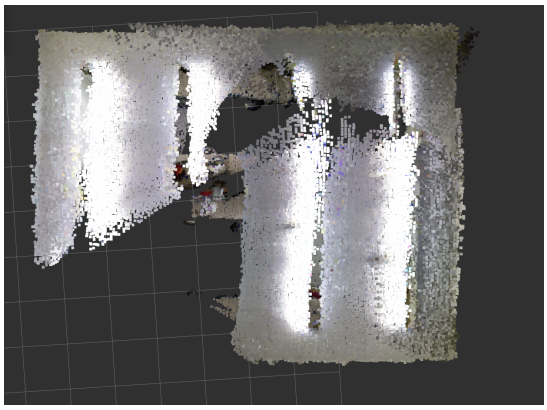
5.2.1.5 VINS-Mono

The image sequence was replayed for VINS-Mono and this time IMU is used to estimate the location of the wheelchair in the generated map. VINS-Mono does not incorporate the Depth Image produced by the Kinect camera. Instead, it works on a sequence of color (RGB) images to estimate the transformations between keyframes using a 3D-reconstruction optimization called bundle adjustment. VINS-Mono produces a 3D map of the room along with the 2D projection. This is shown in the Fig. 5.9a

A top view of the 3D map generated using VINS-Mono is shown in Fig. 5.9, which is bet-



(a) 2D map along with 2D projection using VINS-Mono



(b) Top view of the 3D map generated using VINS-Mono

Figure 5.9

ter than the one generated using RGB-D SLAM.

5.2.2 Test 2 - Adding the corridor to the previous map.

The maps generated in test-1 are saved and reused for test-2. Test 2 evaluates the map insertion in a pre-built map by testing the accuracy of scan/feature matching algorithm of the SLAM.

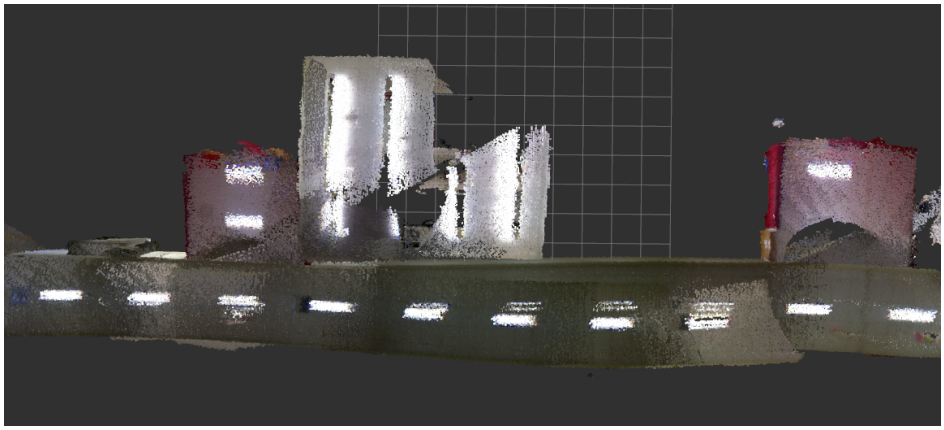


Figure 5.10 3D map of the corridor generated by the visual SLAM algorithms.

Before starting the test, the wheelchair is positioned near the door of the lab, and the algorithm is started. The algorithm is given adequate time to process and estimate the pose of the wheelchair. The wheelchair is then driven through the door to the outside of the lab and through the corridor which is mapped and shown in the results.

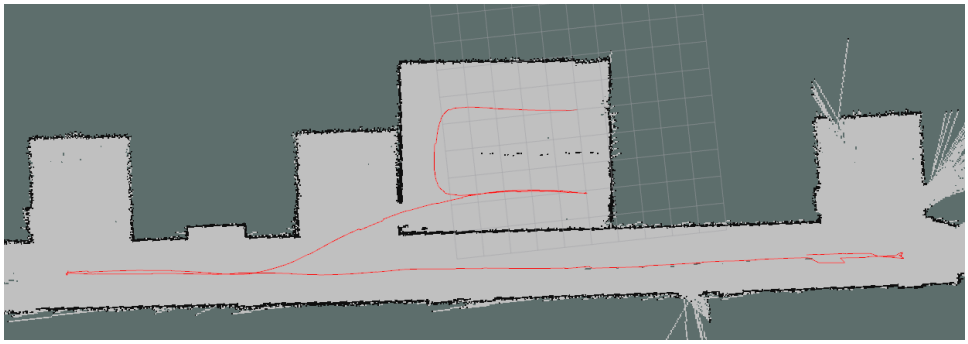


Figure 5.11 Mapping the corridor with Gmapping

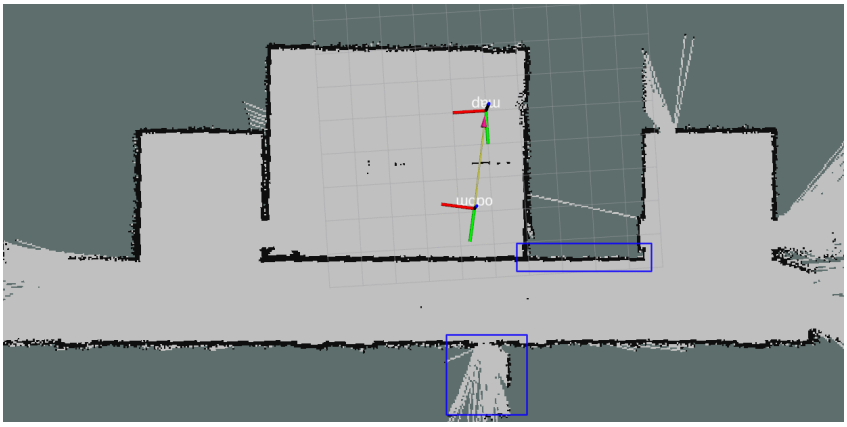


Figure 5.12 Mapping the corridor with Hector

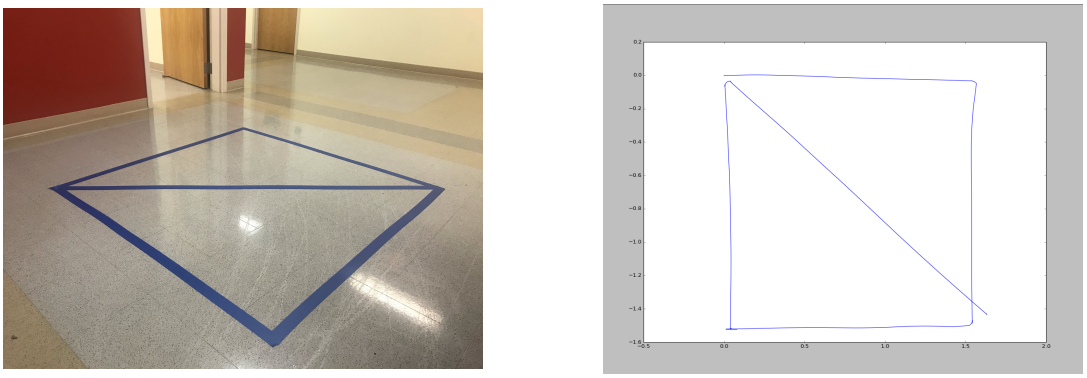
Test-2 essentially draws differences between the two LiDAR based SLAMs - Gmapping and Hector. The visual SLAM algorithms generate similar results as showin in Fig. 5.10.

Fig. 5.11 and Fig. 5.12 show the corridor mapped by Gmapping and Hector respectively. Notice the difference in the map generated by Hector SLAM, marked by two blue rectangles. Hector SLAM mapped a shorter corridor as compared to Gmapping. Compared to the ground truth in Fig. 5.2, Gmapping generates more accurate maps than Hector.

5.2.3 Test 3 - Following a set trajectory

Here, a ground truth trajectory is set and the wheelchair is made to follow the trajectory. The trajectory is stored as a continuous array of poses over time and plotted. This resultant trajectory compared with the ground truth trajectory which is a square of side 157.48 cm (= 62 inches) and one diagonal of length 22.76 cm (= 87.7 inches) as shown in Fig. 5.13 (a). The metrics defined in Section 5.1 i.e. ATE and RPE are calculated. The test is repeated 3 more times and each time the wheelchair speed is increased by 0.25 m/s. A hypothesis is formulated as follows:

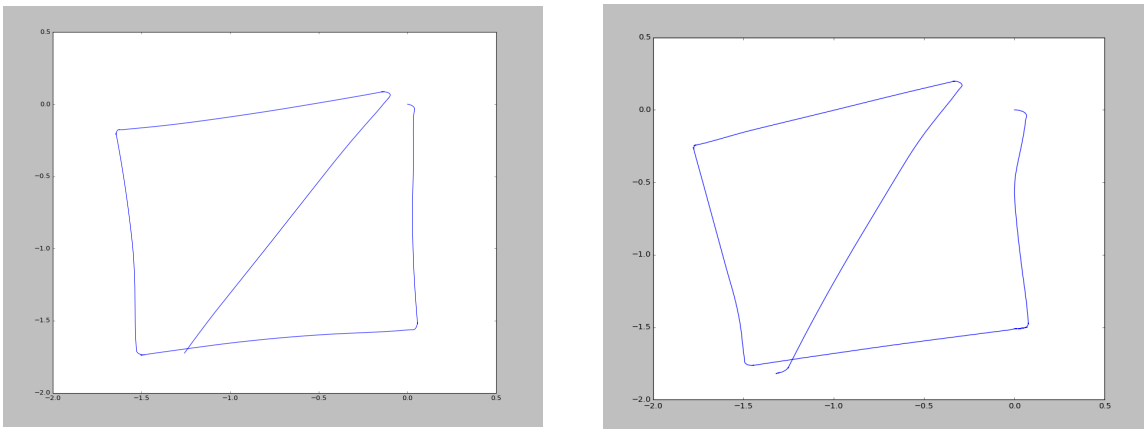
Hypothesis: The Absolute trajectory Error (ATE) will increase with increase in speed of the wheelchair.



(a) **(b)**

Figure 5.13 Ground truth trajectory set for comparison. (a) is the actual track, and (b) is the generated trajectory using EKF by driving the wheelchair over it at 0.5m/s speed.

To make the test standard and avoid contamination due to human error, instead of running the wheelchair over the track separately for different algorithms, the wheelchair was driven once for each speed setting mentioned above, and the data from LiDAR, Kinect, IMU, and encoders was recorded. To generate a ground truth trajectory, dead reckoning was done using an Extended Kalman Filter (EKF) to fuse the data from IMU and encoders 5.13 (b). This recorded data was replayed for each SLAM algorithm and a trajectories were generated for each. An example trajectory for Gmapping and VINS-Mono is shown in Fig. 5.14. Absolute trajectory error and Relative Pose Error for translation and Rotation was computed against the ground truth trajectory and the result is reported in tables 5.1, 5.2, and 5.3 and their corresponding column charts in Figures 5.15, 5.16, and 5.17.



(a) **(b)**

Figure 5.14 Trajectories for (a) Gmapping and (b) VINS-Mono at 1 m/s

Next chapter discusses these results and draws a conclusion based on the requirements and constraints.

Table 5.1 Absolute Trajectory Error (m) with increasing robot speeds for Gmapping, Hector, RTAB-Map, VINS-Mono, and RGB-D SLAM

Robot Speed (m/s)	Gmapping	Hector	RTAB-MAP	VINS-Mono	RGBD-SLAM
0.25	0.12	0.24	0.33	0.2	0.3
0.5	0.16	0.26	0.32	0.21	0.33
0.75	0.25	0.29	0.33	0.41	0.5
1	0.29	0.3	0.35	0.54	0.6

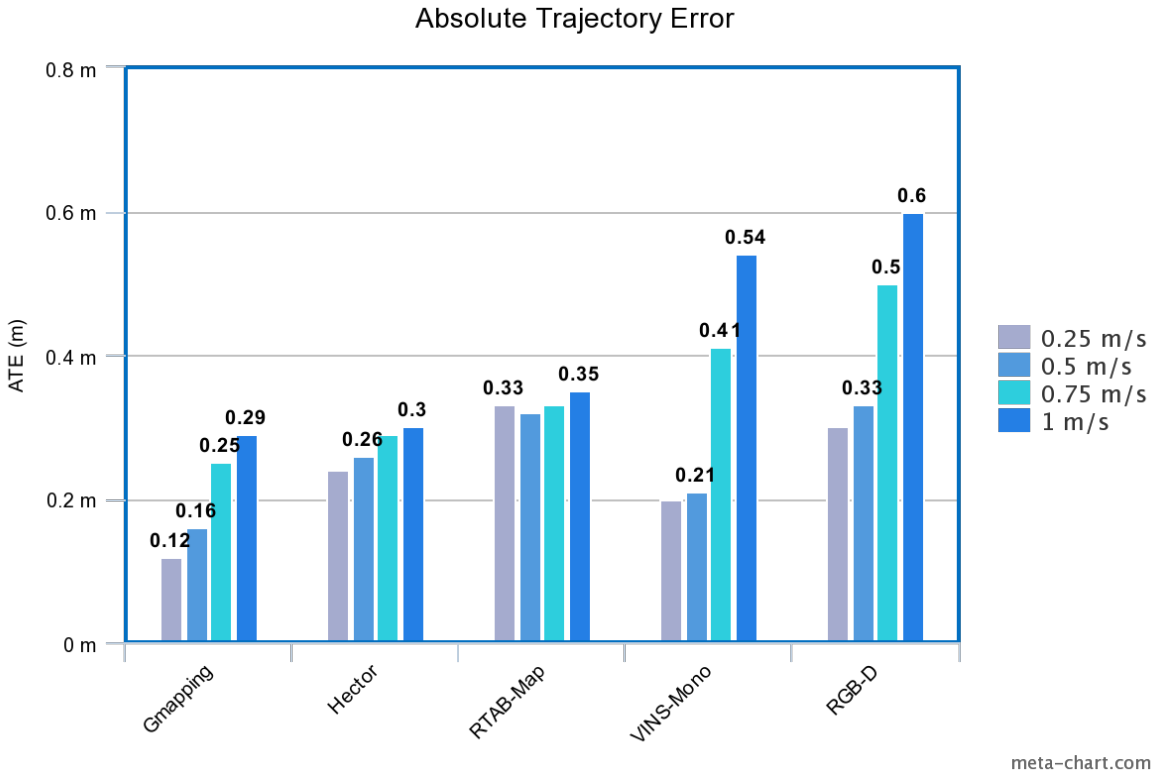


Figure 5.15 Column Chart corresponding Table 5.1

Table 5.2 Relative Pose Translation Error (m) with increasing robot speed for Gmapping, Hector, RTAB-Map, VINS-Mono, and RGB-D SLAM

Robot Speed (m/s)	Gmapping	Hector	RTAB-MAP	VINS-Mono	RGBD-SLAM
0.25	0.15	0.3	0.36	0.25	0.34
0.5	0.20	0.34	0.40	0.27	0.42
0.75	0.28	0.36	0.43	0.46	0.56
1	0.33	0.39	0.48	0.62	0.71

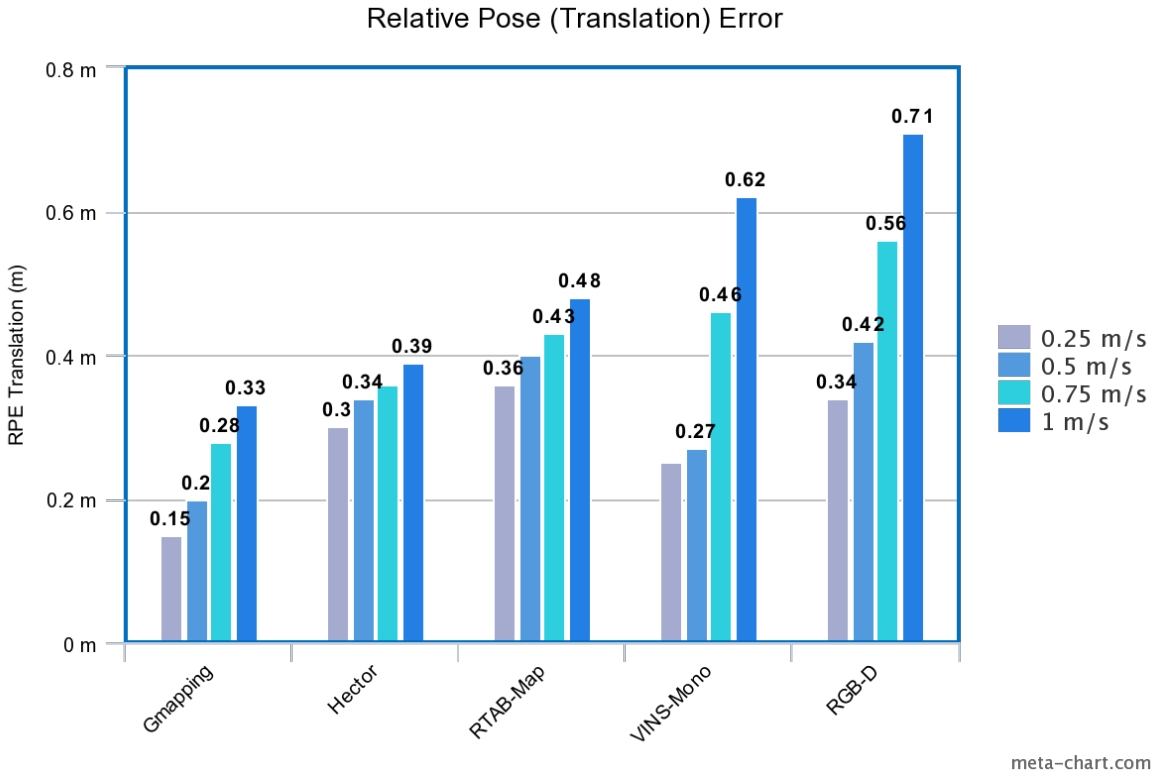


Figure 5.16 Column Chart corresponding Table 5.2

Table 5.3 Relative Pose Rotational Error (degrees) with increasing robot speed for Gmapping, Hector, RTAB-Map, VINS-Mono, and RGB-D SLAM

Robot Speed (m/s)	Gmapping	Hector	RTAB-MAP	VINS-Mono	RGBD-SLAM
0.25	1.26	2.25	2.12	1.29	2.08
0.5	1.54	3.32	2.74	1.57	3.19
0.75	2.21	2.86	3.41	5.42	3.31
1	3.24	4.23	3.77	6.65	5.13

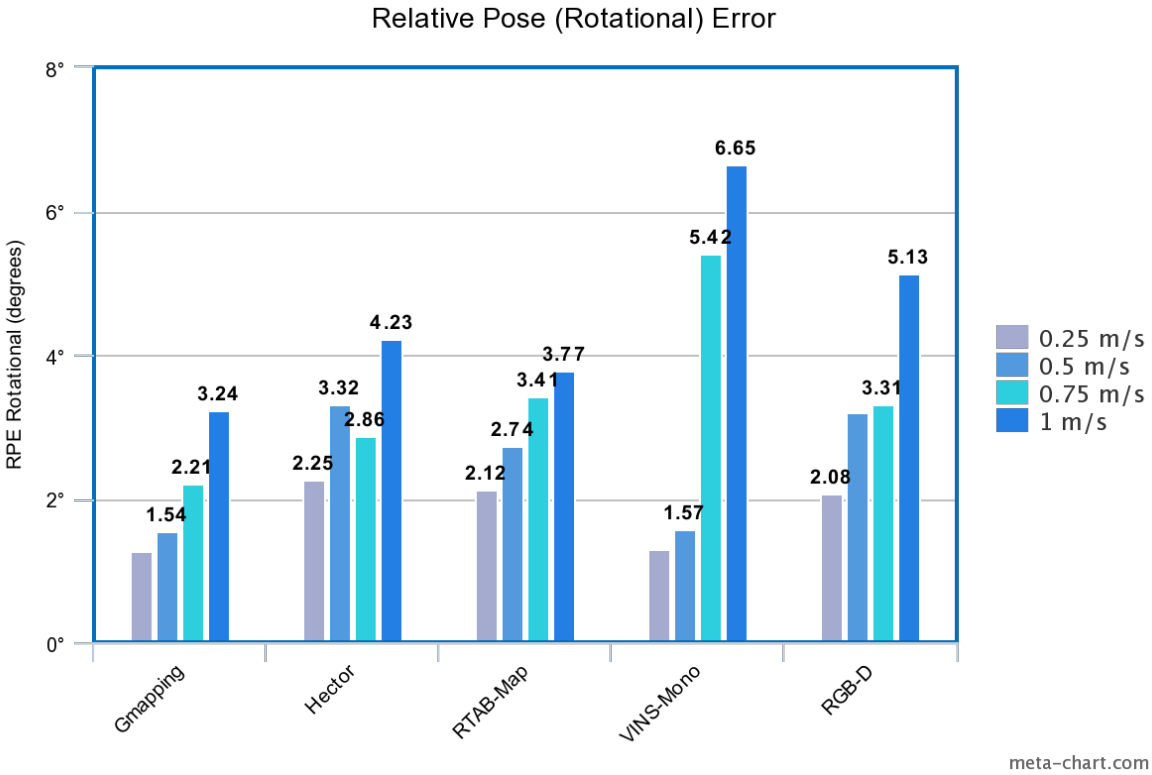


Figure 5.17 Column Chart corresponding Table 5.3

CHAPTER

6

DISCUSSION AND CONCLUSION

As seen from test one, for pure LiDAR based SLAM algorithms, both Gmapping and Hector are able to generate a 2D map of the lab room with accurate wall boundaries. Hector is able to learn more features than Gmapping. For visual-SLAM, all three algorithms (RTAB-MAP, VINS-Mono, and RGB-D SLAM) are able to map the lab room accurately given the constraint which is the field of view. Since, RTAB-MAP and RGB-D SLAM use similar image features (SIFT, SURF, and ORB) their mapping accuracy is better as compared to VINS-Mono. Loop closure is based on keyframe matching where comparing lightweight ORB features yields better results.

For test-2, Gmapping produces accurate map of the corridor unlike Hector. This was ex-

pected because of the fact that Gmapping is utilizing the available wheel odometry data. Hector SLAM relies completely on observed features and when there are no new features (corridor walls in this case), as shown in Fig. 5.12, the algorithm fails to update the map even though the wheelchair is moving through the environment. This results in loss of map data. Visual SLAM algorithms, in this case, are able to generate an accurate map of the environment. Although, RTAB-MAP is able to estimate faster and accurate transformations between keyframes, since it utilizes data from both LiDAR and Camera. Unlike RTAB-MAP, VINS-Mono uses only a monocular camera and employs a 3D-reconstruction algorithm like Structure from Motion(SFM) and bundle adjustment to generate transformations between frames which is computationally more expensive.

The numbers in the results table from test-3 show how the error in the estimated trajectory is increasing with increase in wheelchair speed. The hypothesis formulated for test-3 is proven to be correct for Gmapping, Hector, and RGB-D SLAM. Whereas, the results for RTAB-MAP show a consistently average performance. For VINS-Mono, even though it was originally developed for drones, shows significant drift in trajectory when employed on a wheelchair traveling at $1 m/s$. RGB-D SLAM has some trajectory error at lower speeds which increases and is observable at higher speeds.

For comparison between LiDAR based and visual-SLAM algorithms, it would be incorrect to relate increasing trajectory error with speed to run-time complexity of the algorithm. Hector SLAM, is computationally less expensive than every visual-SLAM algorithm yet is unable able to produce accurate maps due to lack of unique features between scans. But, from Table 5.1, we infer that among visual-SLAM algorithms, VINS-Mono which is computationally more expensive than RGB-D and RTAB-MAP, has higher error at higher speeds.

Unlike self driving cars, which can be located on a globally available map within seconds

using GPS, localization of indoor robots requires more processing. For indoor robots which are employed in warehouses for logistics and organization, algorithms are chosen according to the required operating speed and available computational capacity of the processor. An autonomous wheelchair designed for a patient is required to be functional at a range of speeds depending on the condition and preference of the patient. Since SLAM is an optimization of the perception part of the autonomous pipeline, as explained in Chapter 2, it should be able to generate a map with as many features (obstacles in some cases) learned as possible while simultaneously estimating the position of the robot. This map and location will be later used for *"Path Planning"* which will allow the robot to navigate through the environment autonomously.

A 3D map, generated using sequence of images, accommodates more features than a 2D map generated using only a 2D scan of the environment. But, a 2D scan can be used for faster robot localization. A 2D projection of the same 3D map still contain more features which will allow for better path planning. RTAB-MAP when used with LiDAR, Kinect (RGBD camera), and wheel encoders, utilizes data from LiDAR and wheel odometry for faster localization while building a 3D map using features from the input image. RTAB-Map is found to be robust and versatile in a sense it demonstrates a consistent performance for a range of wheelchair speeds.

BIBLIOGRAPHY

[Bis18] Biswas, D. et al. “"An automatic traffic density estimation using Single Shot Detection (SSD) and MobileNet-SSD"”. *"Physics and Chemistry of the Earth, Parts A/B/C"* ("2018").

[Bue07] Buehler, M. et al. *The 2005 DARPA grand challenge: the great robot race*. Vol. 36. Springer, 2007.

[Dic10] Dicianno, B. E. et al. “Joystick control for powered mobility: current state of technology and future directions.” *Physical medicine and rehabilitation clinics of North America* **21 1** (2010), pp. 79–86.

[Fil17] Filatov, A. et al. “2d slam quality evaluation methods”. *2017 21st Conference of Open Innovations Association (FRUCT)*. IEEE. 2017, pp. 120–126.

[FOL76] FOLE, F. “From da Vinci to the present-A review of airscrew theory for helicopters, propellers, windmills, and engines”. *9th Fluid and PlasmaDynamics Conference*. 1976, p. 367.

[FD06] Forlizzi, J. & DiSalvo, C. “Service robots in the domestic environment: a study of the roomba vacuum in the home”. *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*. ACM. 2006, pp. 258–265.

[Gri07] Grisetti, G. et al. “Improved techniques for grid mapping with rao-blackwellized particle filters”. *IEEE transactions on Robotics* **23**.1 (2007), p. 34.

[Gri05] Grisettiyz, G. et al. “Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling”. *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE. 2005, pp. 2432–2437.

[Hes16] Hess, W. et al. “Real-time loop closure in 2D LIDAR SLAM”. *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 1271–1278.

[Hod12] Hodges, A. *Alan Turing: The Enigma: The Enigma*. Random House, 2012.

[How17] Howard, A. G. et al. “Mobilenets: Efficient convolutional neural networks for mobile vision applications”. *arXiv preprint arXiv:1704.04861* (2017).

[Hu12] Hu, G. et al. “A robust RGB-D SLAM algorithm”. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 1714–1719.

[Kar17] Karami, E. et al. “Image matching using SIFT, SURF, BRIEF and ORB: performance comparison for distorted images”. *arXiv preprint arXiv:1710.02726* (2017).

[Kas18] Kasar, A. “Benchmarking and Comparing Popular Visual SLAM Algorithms”. *arXiv preprint arXiv:1811.09895* (2018).

[Koh11] Kohlbrecher, S. et al. “A flexible and scalable SLAM system with full 3D motion estimation”. *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics* (2011), pp. 155–160.

[Kon00] Kondoh, S. et al. “Self organization of cellular manufacturing systems”. *CIRP Annals* **49**.1 (2000), pp. 347–350.

[Kum09] Kummerle, R. et al. “On Measuring the Accuracy of SLAM Algorithms”. *Auton. Robots* **27**.4 (2009), pp. 387–407.

[LM11] LabbÃl’, M. & Michaud, F. “Memory management for real-time appearance-based loop closure detection”. *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, pp. 1271–1276.

[LM19] LabbÃl’, M. & Michaud, F. “RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation”. *Journal of Field Robotics* **36**.2 (2019), pp. 416–446. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21831`.

[LDW91] Leonard, J. J. & Durrant-Whyte, H. F. “Mobile robot localization by tracking geometric beacons”. *IEEE Transactions on robotics and Automation* **7**.3 (1991), pp. 376–382.

[LT10] Levinson, J. & Thrun, S. “Robust vehicle localization in urban environments using probabilistic maps”. *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 4372–4378.

[LM13] Li, M. & Mourikis, A. I. “High-precision, consistent EKF-based visual-inertial odometry”. *The International Journal of Robotics Research* **32**.6 (2013), 690âĂŞ711.

[Liu16] Liu, W. et al. “Ssd: Single shot multibox detector”. *European conference on computer vision*. Springer. 2016, pp. 21–37.

[LK81] Lucas, B. D. & Kanade, T. “An Iterative Image Registration Technique with an Application to Stereo Vision”. *IJCAI*. 1981.

[Lyn13] Lynen, S. et al. “A robust and modular multi-sensor fusion approach applied to mav navigation”. *2013 IEEE/RSJ international conference on intelligent robots and systems*. IEEE. 2013, pp. 3923–3929.

[Mur00] Murphy, K. P. “Bayesian map learning in dynamic environments”. *Advances in Neural Information Processing Systems*. 2000, pp. 1015–1021.

[Kam] “Performance analysis of the Microsoft Kinect sensor for 2D Simultaneous Localization and Mapping (SLAM) techniques”. *Sensors (Basel)* (2014).

[Pos12] Postolache, O. et al. “IEEE 1451.4 embedded smart sensors architecture for wheelchair user monitoring”. *2012 IEEE International Symposium on Medical Measurements and Applications Proceedings*. 2012, pp. 1–5.

[Pos14] Postolache, O. et al. “Toward developing a smart wheelchair for user physiological stress and physical activity monitoring”. *2014 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*. 2014, pp. 1–6.

[PN05] Premebida, C. & Nunes, U. “Segmentation and geometric primitives extraction from 2d laser range data for mobile robot applications”. *Robotica* **2005** (2005), pp. 17–25.

[RB04] Riisgaard, S. & Blas, M. R. “SLAM for Dummies A Tutorial Approach to Simultaneous Localization and Mapping By the âĂŸdummiesâĂŹ” (2004).

[ROB16] ROBOTICS, I. L. “A DPDHL perspective on implications and use cases for the logistics industry”. *DHL March* (2016).

[Sen] Sensortec, B. “Intelligent 9-axis absolute orientation sensor” ().

[She15] Shen, S. et al. “Tightly-coupled monocular visual-inertial fusion for autonomous flight of rotorcraft MAVs”. *2015 IEEE International Conference on Robotics and Automation (ICRA)* (2015).

[ST93] Shi & Tomasi. “Good features to track” (1993).

[Thr06] Thrun, S. et al. “Stanley: The robot that won the DARPA Grand Challenge”. *Journal of field Robotics* **23**.9 (2006), pp. 661–692.

[Tro89] Trombly, C. A. “Occupational Therapy for Physical Dysfunction”. 1989.

[Urm08] Urmson, C. et al. “Self-driving cars and the urban challenge”. *IEEE Intelligent Systems* **23**.2 (2008), pp. 66–68.

[Wei12] Weiss, S. et al. “Real-time onboard visual-inertial state estimation and self-calibration of mavs in unknown environments”. *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 957–964.

[Xu17] Xu, Y. et al. “3D point cloud map based vehicle localization using stereo camera”. *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 487–492.

[Zho14] Zhou, T. T. et al. *Unmanned drone, robot system for delivering mail, goods, humanoid security, crisis negotiation, mobile payments, smart humanoid mailbox and wearable personal exoskeleton heavy load flying machine*. US Patent App. 14/285,659. 2014.