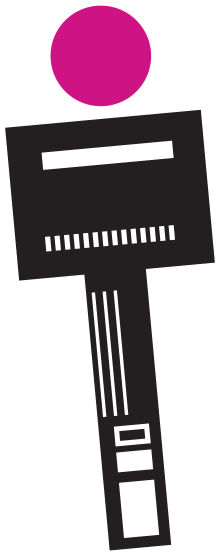


RAM ROBOTICS AND MECHATRONICS

DEVELOPMENT OF SLAM ALGORITHM FOR A PIPE INSPECTION SERPENTINE ROBOT

Shrijan Kumar

MSC ASSIGNMENT

Committee:
prof. dr. ir. G.J.M. Krijnen
N. Botteghi, MSc
dr. ir. D. Dresscher
dr. B. Sirmaçek
dr. ir. L.J. Spreeuwers

November, 2019

047RaM2019
Robotics and Mechatronics
EEMathCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Summary

The use of different mobile robots for the pipe inspection system is gradually gaining importance since they are capable of performing a faster and more accurate inspection. Due to the transportation of various energy-related utilities, these pipes need to be inspected regularly. These pipelines are generally small in diameter, which makes it very difficult for a human to do an inspection, thus building an autonomous pipe inspection robot is of immediate necessity. For this reason, the Robotics & Mechatronics (RAM) research group has developed a Pipe Inspection Robot for AuTonomous Exploration (PIRATE), a six-segments wheeled snake-like robot, at the University of Twente, which is used for inspecting pipes with a diameter of 100-150mm. In the field of mobile robot navigation, Simultaneous Localization and Mapping (SLAM) is a crucial technique that aims at consistently building the robot environment and simultaneously determine the robot location in that environment. Much research has been carried out on solving the SLAM problem, but despite significant progress, there are issues depending on the application.

In this context, this thesis aims to investigate and implement a SLAM algorithm for a pipe environment so that PIRATE can autonomously navigate through different pipe segments and carry out the inspection process. The first goal is to implement a different 2D-3D SLAM algorithm for the in-pipe environment in Gazebo, a ROS-based simulation framework. The next goal is to evaluate the built map based on its quality, localization error, computation, and loop closure. Further, the algorithm is tested on two physical robots, i.e., Jackal, a differential drive mobile robot and PIRATE.

One of the biggest challenges in SLAM is the lack of sufficient features or textures in an environment that makes it difficult for a range-based or a vision-based sensor to do mapping and localization. To overcome such a problem, wheel odometry can be used to estimate robot position, and thus construct the map. However, the wheel odometry is prone to error with time, but this error can be greatly reduced by various means, for example, sensor fusion between wheel encoders and IMU (accelerometer and gyroscope) can provide reliable odometry data. Driving a robot at a low speed reduces slippage, and proper wheel calibration removes the systematic error. Estimation of the wheel slip ratio and friction coefficient can also help to compensate for slippage and thus provide reliable odometry data. In this thesis, sensor fusion between wheel odometry and Inertial Measurement Unit (IMU) data is proposed. Based on the selection of the sensors, Grid-based Mapping (Gmapping), Google Cartographer algorithms will be exploited for mapping (2D and 3D) and localization inside different pipe segments. Gmapping performs particle-based state-estimation, while Cartographer uses graph-based approaches. Further, a 3D memory-efficient mapping algorithm called Octomap is integrated into the system. An Adaptive Monte Carlo Localization (AMCL) algorithm was used to reduce localization error on a pre-built map.

During experimentation, it is observed that Cartographer outperforms the Gmapping SLAM algorithm with respect to the construction of more precise maps and pose estimation. In addition to this, its low computation and memory usage make Cartographer, an ideal candidate for SLAM framework inside pipes. Moreover, AMCL performs exceptionally well with the fused data and range-based data inside the pipe network. Further, an Octomap is constructed on top of Cartographer, making it a perfect combination. This technique increases the scalability of large pipe networks because of its low computational and memory usage. However, there is still some inconsistency in the built map due to sensor noise, slippage, etc. To address the inconsistent behavior of these algorithms, suitable modifications are suggested.

Acknowledgement

This thesis work would not have been possible without the help and encouragement of many people. First of all, I would like to thanks ir.Nicolò Botteghi (MSc), my supervisor for allowing me to assist the research on navigating the PIRATE. It has been a gratifying learning experience for me to work with him. I would also like to thank dr. Beril Sirmacek, for her continuous support throughout my thesis work. Both of them consistently encouraged me and gave me the freedom to explore in the robotics domain, which I appreciate. I would also like to thank all of my remaining committee members, prof.dr.ir. G.J.M. Krijnen, dr.ir. D. Dresscher and dr.ir. L.J. Spreeuwers for their necessary supervision.

My most sincere thanks go to the PIRATE team for their technical guidance and support. I want to extend my gratitude to all my network of people at the University of Twente, in particular, Atul Hari and Yogesh Sharma for their technical insights and advice, as well as their friendship and moral support. Finally, I would like to thank my family and friends, especially K.Siddharth, ir. Udayan Sinha (MSc), and D.Tirindelli for their support, which is beyond measure.

Contents

List of Figures

List of Tables

List of Abbreviations

PIRATE Pipe Inspection Robot for AuTonomous Exploration

RAM Robotics and Mechatronics

SLAM Simultaneous Localization and Mapping

Gmapping Grid-based Mapping

LSD Large-scale direct

MDP Markov Decision Process

ICP Iterative Closest Point

RBPF Rao-Blackwellised Particle Filter

ROS Robot Operating System

LiDAR Light Detection and Ranging

RTAB-Map Real-Time Appearance-Based Mapping

STM Short-Term Memory

WM Working Memory

LTM Long-Term Memory

AMCL Adaptive Monte Carlo Localization

KLD Kullback-Leibler divergence

PICO PIRATE control

EKF Extended Kalman Filter

UKF Unscented Kalman filter

IMU Inertial Measurement Unit

POSE Position and Orientation

1 Introduction

1.1 Context and problem statement

Petrochemical pipelines have an extensive network in appropriate sites for various transportation. These pipelines vary in size from millimeters to meters and suffer through different loads originated from both interior and exterior of the pipe, as well as oxidation and corrosion of pipe surface. This sometimes leads to leakage and, therefore, pipelines have to be inspected at frequent intervals. Pipeline inspection is known to be a slow, time-consuming, and labor-intensive process. In most cases, the pipes are not easily accessible, and sometimes an entire plant must cease operations during inspections, hence making it difficult to monitor. Since many portions inside the pipe are not accessible to humans, it is often inspected from the outside, which again involves discarding of the insulating material. Sometimes an entire pipe gets replaced with the slightest doubt in terms of quality, thus inflating the maintenance cost. Replacing the defective portion sometimes leads to an enormous part of the region being excavated. In circumstances like these, it becomes essential to have information on the location of the damaged section.

To counter this problem, Pipe Inspection Robot for AuTonomous Exploration (PIRATE) has been developed by Robotics and Mechatronics (RAM) group at the University of Twente (UT) as shown in figure 1.1. Apart from leakage detection, PIRATE could also be used for quality checks; hence, weak spots can be easily identified. The PIRATE project was started by KIWA[1] in collaboration with UT in 2006. The development of this project is part of the European Smart Tooling project, which seeks to make the process industry safer and more cost-effective. The initial design was made in such a way that the PIRATE should be able to inspect pipes of 100-150 mm diameter. Moreover, it can move through different pipe segments like T-junction, 90° sharp bends, and other types of intersections. Over these years, the PIRATE went through many design changes but mostly focused on mechanical and electronic design.

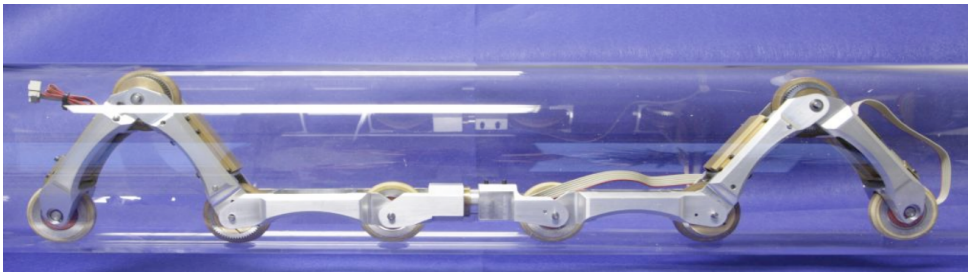


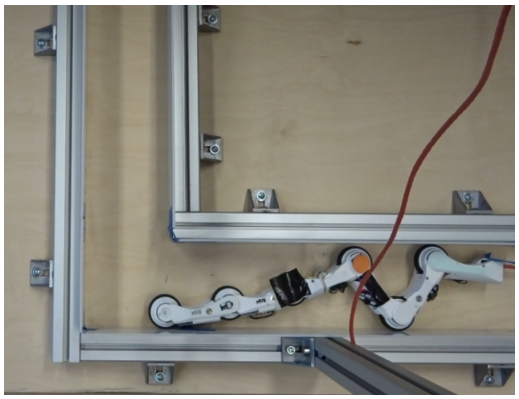
Figure 1.1: PIRATE inside a pipe [1]

Many works have been done on the autonomous motion of the robot, as depicted in figure 1.2, but always with the assumption of known state and environment [7]. In [8], a camera was placed overhead of the PIRATE for position and orientation (pose) estimation based on marker detection, as seen in figure 1.2a. Some work has been undertaken to develop a vision sensor using a monocular camera attached to the head of the PIRATE and a laser projector to project a circular-shaped pattern on the pipe wall [9], shown in figure 1.2b. This approach uses a camera that captures the laser projection and hence, uses suitable image processing techniques to map the pipe surface and identify defects. But using such methods, apart from heavy memory usage, it increases the computation if it is made to run in a long pipe since it has to process multiple images at once. Currently, research is still underway on the detection of the pipe junction

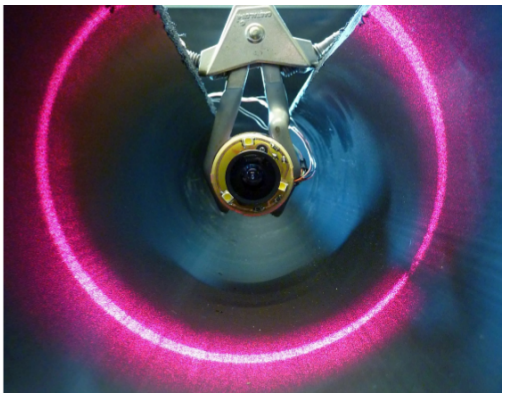
[1] KIWA (gastec), Apeldoorn, http://www.kiwa.nl

and the estimation of the bend angle within the pipe. Also, some research is being done based on the learning algorithm, which will allow the PIRATE to interact with different pipe configurations and to navigate independently; nonetheless, the robot still requires a map inside the pipe surface for safe navigation. In particular, the position of PIRATE within the map is needed to find defects within the pipeline. So, to achieve the next step of autonomy, localization, and mapping of the pipeline network need to be done.

The problem of getting a robot to build a map of the environment while moving through the same environment is known in robotics as the Simultaneous Localization and Mapping problem. Thus, this project focuses on developing a Simultaneous Localization and Mapping (SLAM) framework for the PIRATE in a pipeline network so that it will be easier to inspect and identify defects by just localizing the PIRATE within the pipe.



(a) Pose estimation using marker-based detection [8]



(b) Laser projection inside the pipe. Hence, shape of the pipe can be determined [9]

Figure 1.2: Prior work implemented on PIRATE

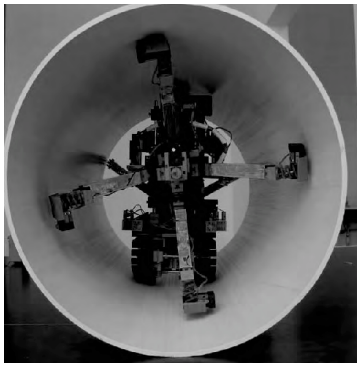
Various other types of robots are used to assess the condition of the pipes using different techniques. The conventional inspection includes wheeled-type,caterpillar-type in-pipe robot with an onboard camera system, as shown in figure 1.3. These robots are attached to a cable for energy supply, communicate transmission commands from a human operator to the robot. These types of robots are generally used for large diameter pipes.

Moreover, there is a wall pressed type robot, which is useful for vertical pipe inspection. MAKRO (Mehrsegmentiger Autonomer KanalROboter/multi-segmented autonomous sewer robot) was designed to autonomously navigate through sewer pipes of diameter 300 to 600mm at dry weather conditions. PIG(Pipe inspecting gauges) robots are commonly used for inspecting pipes with large diameters. It is one of the most frequently used robots for pipe cleaning when there is a decent quantity of flow in the pipeline. For smaller pipes, inchworm-type robots are used.

In most cases, the inspection work is carried out using a vision system. So, an operator remotely controls the robot and the camera system, thus recording any notable damages or abnormalities inside the pipe. The video-supported system requires high memory usage, and the reliability of such a system depends on the experience of the worker. However, the location of these defects is still unknown, as the position of the robot within the pipe network is not established. Thus, having a SLAM framework helps in the inspection process and adds versatility to the system i.e., it can be adapted by any mobile robot.

(a) Powervision Crawler [10]

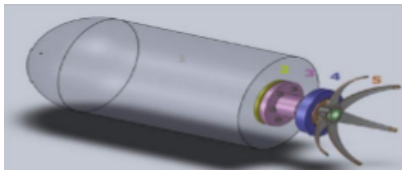
(b) H-Canyon Inspection Crawler [10]



(c) Wall pressed type robot [11]

(d) Inspection platform, MAKRO [12]

(e) Inchworm-type robot [13]



(f) PIG robot [14]

Figure 1.3: Robots that can move through different configuration of pipes

1.2 Research Objectives

The final goal of this project can be stated as "To build a map of a pipe and localize the PIRATE within that map". To achieve this goal, the following research questions need to be investigated.

1. Is it possible to build a map and localize inside the pipe network by only using a range or vision sensor on the robot?
2. How can a low-cost IMU (Inertial Measurement Unit) sensor measurement help to improve the localization of a robot in a pre-built map of the pipe network?
3. How can the mapping accuracy of the Gmapping algorithm be improved for different pipe segments?
4. How can Google Cartographer be exploited effectively for building a 2D-3D map inside the pipe environment?

The work done in this thesis is dedicated to answer these research questions and further make a comparative evaluation of the implemented mapping and localization approaches.

1.3 Related Work

In recent years much research has been done on the SLAM algorithm, and many different algorithms have been developed for numerous applications [15]. Achieving a solution to the SLAM problem has been one of the most challenging mobile robotics literature studies in the previous two decades. In this research, the analysis was mostly focused on the SLAM technique for

a closed network such as pipelines, tunnels, caves, etc. Most of the algorithms are based on Visual SLAM or Large-scale direct(LSD) SLAM [16] [17], where a monocular or a stereo camera is used for mapping and localization. However, these algorithms were implemented with sparse features[2] inside the pipe, which makes it easier to map and localize. Nevertheless, problems are encountered in environments where features are neither easily detectable nor abundant and, hence, these algorithms perform erratically and are unreliable in such an environment. Although, several other methods use cable length between the robot and the entrance of the pipeline [18], but the power cable is not reliable in case of a long pipeline network or a multi-junction pipeline since the cable curves or coils at the junctions. Similarly, the sound-based localization technique has also been tried by attaching a sound cable to the robot and adding a microphone to it [19]. Hence, based on the time of flight, the distance can be estimated within the closed structure. Again, it is difficult to travel in a long pipe and not efficient to map and localize the robot accurately. For water pipes, a 1D map can be created using hydrophone sensors. When the hydrophone sensor is excited, its vibrations are recorded inside the pipe while traveling [20]. For localization, particle filter was used to estimate the position of the robot.

Therefore, in this project based on the sensor selection, a suitable SLAM algorithm will be exploited in a simulation environment (Gazebo) in order to test and experiment realistically with real scenarios.

1.4 Approach

The SLAM algorithm will be tested inside a straight pipe, T-junction, 90° sharp bends, and other intersecting junctions, as shown in figure 1.4. All the configuration represents the real world pipe models that are made in **Blender**, a 3D computer graphics software toolset. Based on the sensor selection, it will be tested for the 2D algorithm, followed by 3D. The next step is to investigate the algorithm for different mapping conditions like loop closure and, afterward, to validate the simulation results, it will be tested on the physical robot.

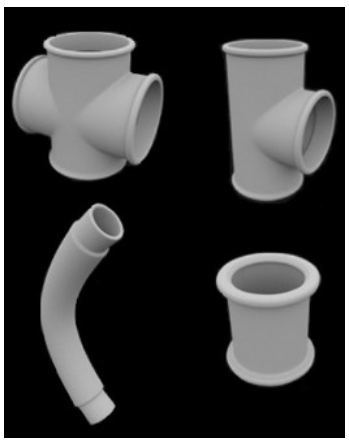


Figure 1.4: 3D Rendering of different pipe configurations

One of the key assumptions made in this work is that real-time testing will not be conducted in transparent pipes, as the sensing capability for both vision and range-based sensors in such an environment fail.

[2] sparse feature means lack of sufficient features. In pipe networks, these features correspond to intersecting junctions, bends, or small diameter changes.

1.5 Thesis Methodology

The workflow for this thesis has been illustrated in fig 1.5. The first stage is the selection of sensors based on the environment, which is a pipe network. In order to get reliable data, sensor fusion is performed based on the selected sensors. The heart of the workflow is the implementation and optimization of the SLAM framework for the in-pipe environment, based on the prior research study. Different 2D-3D algorithms, such as Gmapping, Cartographer, will be tested and further optimized to build an appropriate map of the environment. However, there are few independent mapping and localization techniques as well that can be applied to the given framework. For example, the localization technique like Adaptive Monte Carlo Localization (AMCL) can be tested on a pre-built map, or an Octomap can be constructed if the robot pose is known. Based on the simulation results, the implemented algorithm will be tested on the physical model.

1.6 Outline

The rest of this thesis report will be organized as follows,

Chapter 2 provides background information about SLAM and presents it's formulation in a probabilistic fashion. Moreover, it gives an overview about the description and current design of the PIRATE, and highlights the ROS software framework used in this assignment.

Chapter 3 will analyze the problem faced in state-of-the-art SLAM researches inside a pipe and will discuss thoroughly the choice of sensors and algorithm adopted in this thesis work. Further, it highlights the contribution to this research work.

Chapter 4 presents the experiments that will be conducted to validate the algorithm in both 2D and 3D. It will cover the experiments for both simulation and real-time.

Chapter 5 provides the results and evaluation of SLAM experimentation with the simulated and physical robot.

Chapter 6 summarizes the project with a conclusion and possible recommendations for future works.

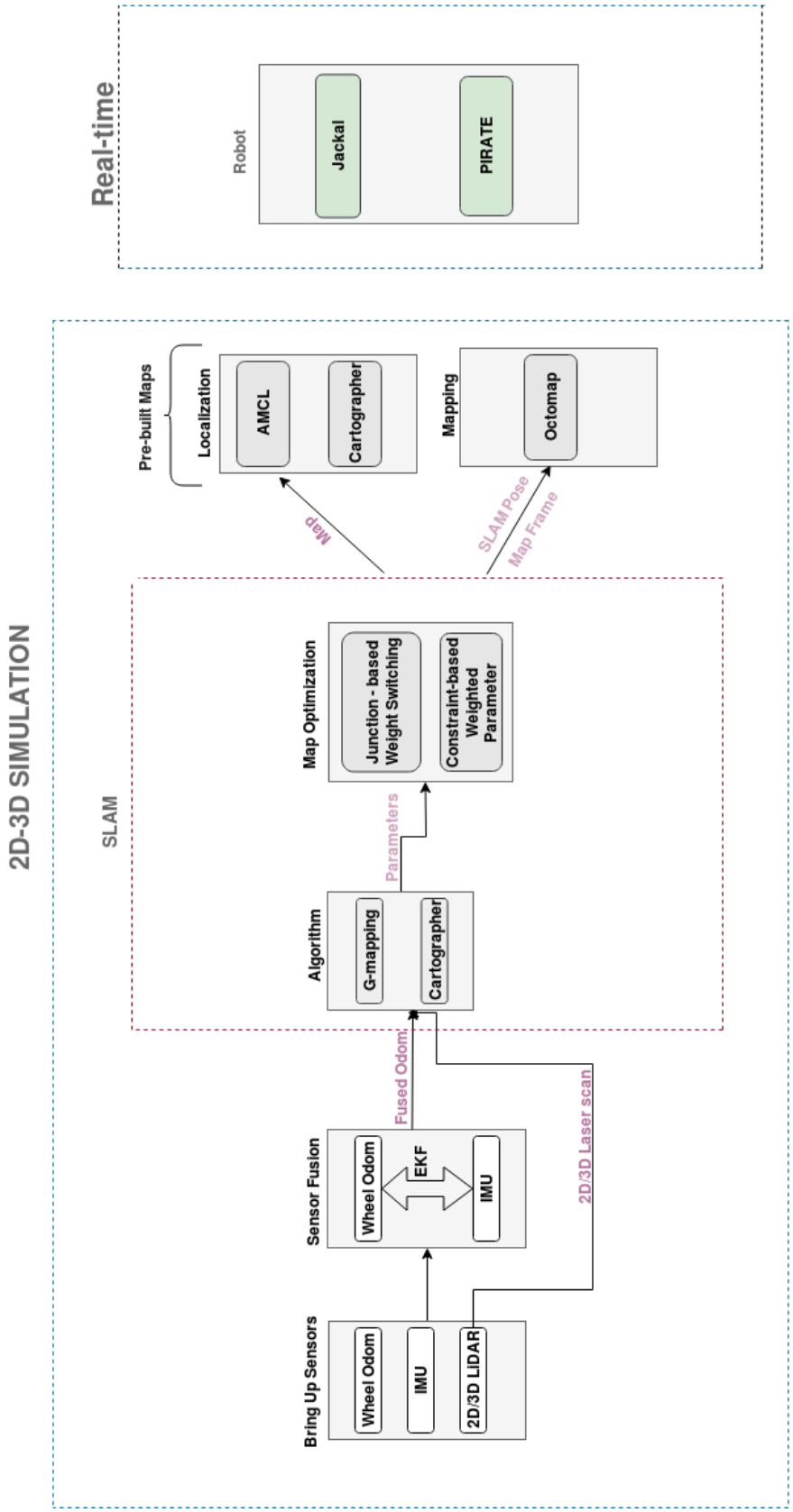


Figure 1.5: Workflow for Thesis Methodology

2 SLAM: Background Information

This chapter, firstly, provides background knowledge about Simultaneous Localization and Mapping (SLAM), which will help the reader understand the work carried out in this thesis. Moreover, it briefly explains the various SLAM algorithm applied during this assignment. Finally, it presents an overview of the hardware architecture of the PIRATE, followed by the software framework used in this project.

2.1 Simultaneous Localization and Mapping

The ability to navigate autonomously is one of the most fundamental yet crucial features of intelligent mobile robots. Simultaneous localization and mapping (SLAM) allow the robot to attain such autonomy by answering questions about its world appearance (mapping) and where its located in that world (localization). Both mapping and localization tasks cannot be decoupled and solved independently. Therefore, SLAM is often regarded in robotics as a "chicken-and-egg" problem: A good map is required for navigation, while a precise pose estimate is needed to construct the map [21]. To achieve this objective, it uses a variety of sensor data based on the algorithm. For example, a range-based SLAM uses LiDAR data to build and localize in a map while a vision-based SLAM uses a camera sensor to acquire visual data, and, in some cases, combines odometry data from wheel encoders to build the map. Figure 2.1 shows the general layout of a SLAM process.

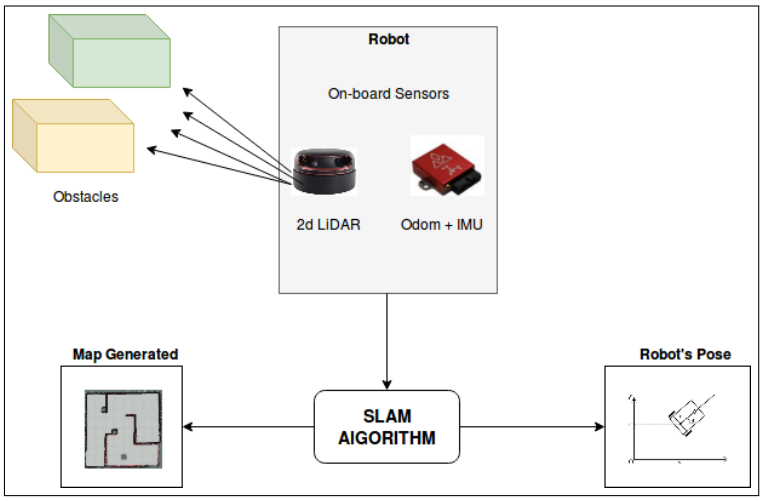


Figure 2.1: Overview of SLAM process while constructing map and localization

Hence, the solution to SLAM problems can be very efficient in cases where global measurements such as Global Positioning System (GPS) are not accessible, for example, when a robot or agent is operating indoors. SLAM comes with intrinsic uncertainty, like all sensing elements. The accuracy mostly depends on the following three factors:

- type of environment
- on-board sensor accuracy
- computational power required for processing

Out of the three factors, computational power depends on the complexity of the algorithm and how much power, a user is willing to spend on it. In order to have a clear insight into the SLAM problem, the core concept is briefly introduced [22].

2.1.1 SLAM Formulation

A robot executes a sequence of control commands and accumulates observation based on the type of sensors. Each control and observation, combined with an appropriate noise model, can be considered as a probabilistic constraint for the system. It tries to estimate the posterior distribution of the robot's pose and map based on the data acquired from on-board sensors. Mathematically, it can be defined as

$$p(x_{0:t}, m | z_{1:t}, u_{1:t}) \tag{2.1}$$

where, p represents the probability distribution of the function, $x_{0:t}$ is pose of the robot at discrete points in time (0 to t), m is the map of the environment, $z_{1:t}$ are sensor observations from time 1 to t, $u_{1:t}$ are control commands or odometry information from 1 to t. It can also be represented in the form of the Bayesian network, which gives a clear visualization of the dependencies of the variable in the SLAM problem. In figure 2.2, x_t represents the robot's state (position and orientation), u_t is the control signal, z_t is the observation obtained by the sensors at time t and m tells about the map information. The easiest way to estimate the joint probability of pose and map of the environment is by using the Bayesian probabilistic approach. Assuming the system dynamics obeys Markov Decision Process (MDP), the probability distribution function can be written using conditional probability as

$$p(x_t, m | z_{1:t}, u_{1:t}) = \eta \cdot \underbrace{p(z_t | x_t, m)}_{\text{Observation model}} \int \underbrace{p(x_t | x_{t-1}, u_t)}_{\text{Motion model}} \underbrace{p(x_{0:t-1}, m | z_{1:t-1}, u_{1:t-1})}_{\text{Prior distribution}} dx_{t-1} \tag{2.2}$$

In the above equation, the observation model gives the probability of the sensor data at time t, given the map m and state x at time t. The motion model tells how the system x evolved from t-1 to t given an executed control command u at time t. The prior distribution gives the previous state information while the term η is a normalizing constant in the Bayes' rule, which ensures equation 2.2 correctly represents the probability distribution. Based on the probabilistic approach, it tries to find the solution to two forms of SLAM problem i.e., full SLAM and online SLAM. Full SLAM calculates the entire robot's path, and its probability distribution function is denoted by equation 2.1 while Online SLAM calculates the robot's most recent pose and the map, given the control and measurement.

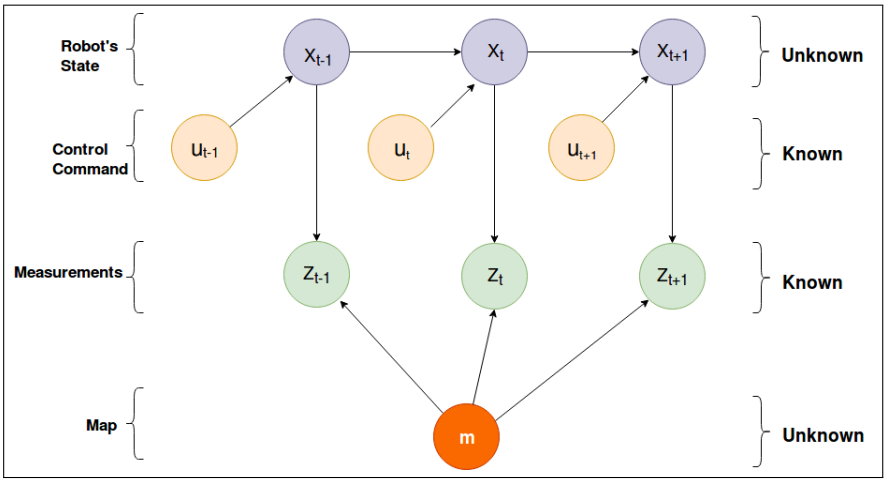


Figure 2.2: Graphical representation of SLAM problem

The most common issue in SLAM is data association. The challenge comes in deciding which noisy measurement corresponds to which feature or landmark in the map as depicted in figure 2.3. Associating a wrong feature information may induce a bad estimations of the map and

hence, might result in a deformed map which becomes difficult for applying other localization techniques as well.

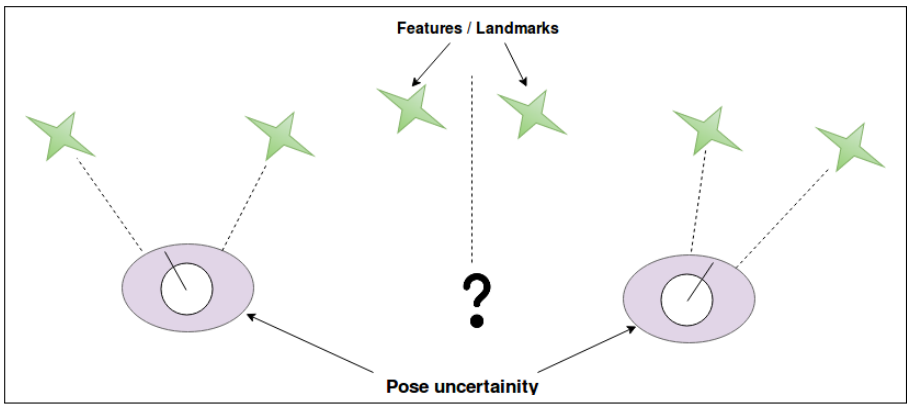


Figure 2.3: Data association problem due to motion ambiguity. Adapted from [2].

2.2 Environmental Representation

A suitable representation of the environment is needed to build an environment map and use the map for the robot's localization. The accuracy of such maps is correlated with the accuracy of the SLAM algorithm. In 2D, the most suitable representation of the indoor environment is Occupancy Grid Maps while in 3D, a memory-efficient, probabilistic map known as Octomap is used.

2.2.1 Occupancy Grid Maps

Occupancy Grid map is the most common technique to approximate and represent a 2D map environment that does not suffer from data association [23], as discussed in the previous section. It requires a Bayesian filtering algorithm to maintain its structure i.e., recursive update to the map. In this technique, the entire map of the environment is discretized into small rectangular cells. Each cell holds the occupancy information, which makes it simpler for estimation, and because of this reason, it has been used extensively for navigation and path-planning. A cell (i, j) can take three values i.e.

$$x(i,j) = \begin{cases} 1, \text{ if cell is occupied} \\ 0, \text{ if cell is empty} \\ -1, \text{ if cell is unknown} \end{cases}$$

In figure 2.4, white pixels correspond to free space, black pixels represent partially or fully occupied space while grey pixels tells about the unknown area in the map. This type of map cannot be absolutely accurate, but they can provide relevant information by choosing small enough cell size. The occupancy information of any cell can be easily known using range sensors. The downside of such map representation is it comes at the expense of higher memory requirement, making it disadvantageous for larger maps.

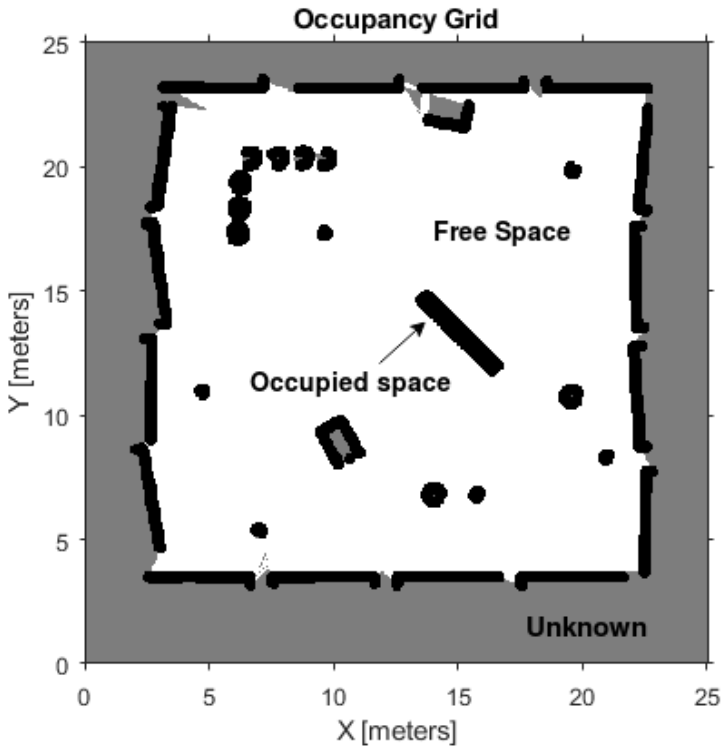


Figure 2.4: 2D Occupancy Grid Map. White pixels: free space, black pixels: occupied space, grey pixels: unknown space. Adapted from [3].

2.2.2 Octomap

Octomap is a probabilistic, flexible, compact, and multi-resolution 3D mapping procedure that is popular in robotics systems [24]. The framework is based on an octree (tree-based hierarchical data structure) that provides an efficient occupancy grid mapping. The basic idea behind the Octomap is that it separates the free and occupied cells from the map using octree. In 3D, it uses a grid of cubic volumes of equal size, called voxels, to model the 3D environment, as seen in figure 2.5. Each voxel can be further sub-divided into eight smaller volumes recursively until a minimum voxel size is reached, which determines the resolution of the octree. These voxels whose constituent octants are either free or occupied are treated as a single element. As a result of such subdivisions, it reduces the file size of the large-scale maps to a great extent. Octomap integrates the sensor measurement and updates itself in a probabilistic fashion to cope with sensor noise or any dynamical change in the environment. The octomap plugin [1] has been created in ROS for navigation and path planning of mobile robots. Figure 2.6 shows Octomap representation for an environment in ROS. The height-map is color-coded in Octomap. These Octomap can be generated from any point cloud map using the Octomap plugin. When an Octomap is being converted from a point cloud map, any resolution can be considered for the task depending on the size of the robot and computational power. Since Octomap is a mapping procedure, therefore a SLAM algorithm is required to populate the Octomap correctly.

[1] http://wiki.ros.org/octomap_server

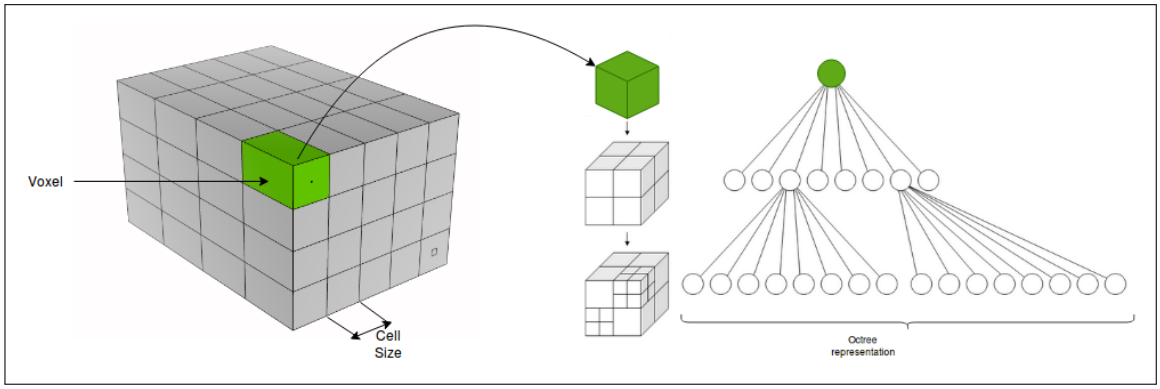


Figure 2.5: Voxel and Octree representation. The volumetric model is shown on left and corresponding tree representation on right.

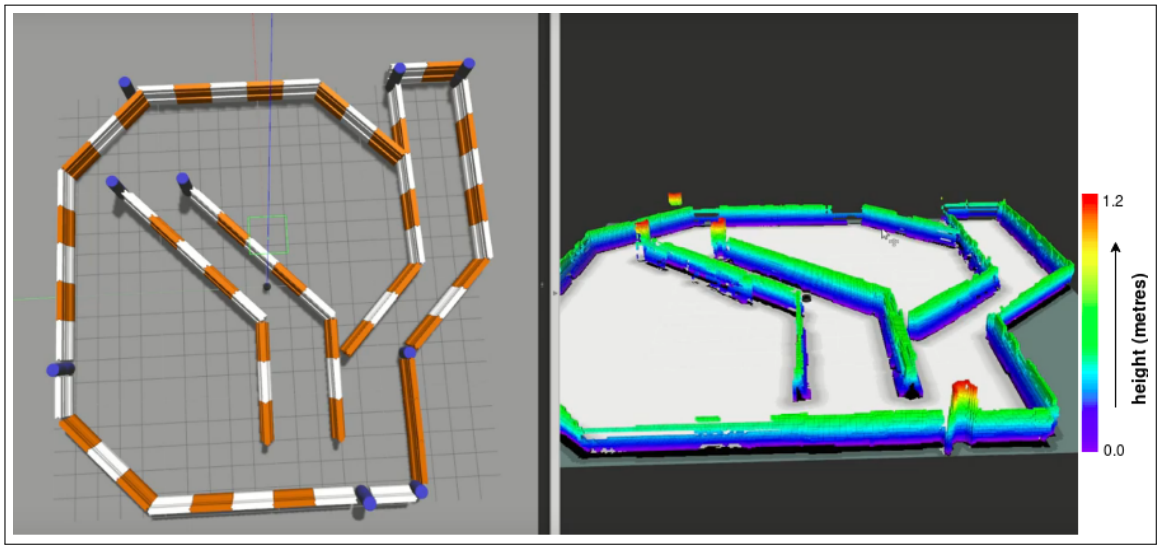


Figure 2.6: Octomap representation. An environment (barrier world) in Gazebo is displayed in the left image while its Octomap representation on the right. Color bar indicates the height of the barriers as depicted in Gazebo.

2.3 Sensors

Robot localization needs sensory information on the position and orientation of the robot within the constructed map, which are provided by two types of sensors: Exteroceptive and Proprioceptive sensors [25]. Exteroceptive sensors obtain information from the robot environment, for example., measurements of distance, light intensity, the amplitude of the sound. LiDAR, sonar, camera are few examples of Exteroceptive sensors; Whereas Proprioceptive sensors measure values internal to the system, for example, wheel position, joint angle, motor speed. These sensors can be wheel encoders, gyroscope, etc. Generally, the robot positioning is done by Proprioceptive and Exteroceptive, which are also classified as relative position measurement and absolute position measurement, as shown in figure 2.7. The relative positioning is calculated by integrating a sequence of measurements from the starting point. Dead reckoning is likely one of the oldest ways of relative localization. The absolute positioning measurement provides information on the location of the robot independent of previous position estimates; Because of this, the error in the position does not grow unbounded, as is the case of relative position measurement. GPS, vision systems are a few common examples of absolute positioning systems.

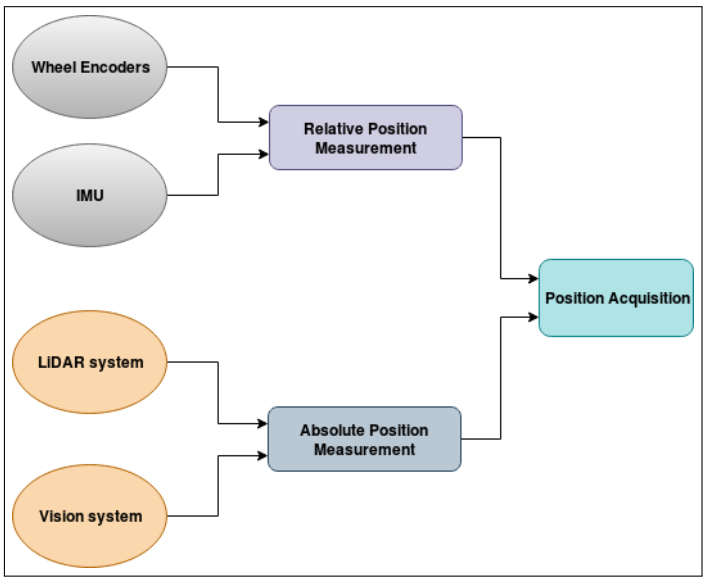


Figure 2.7: Different types of Position Measurement system

The most commonly used proprioceptive sensor is wheel encoders. The wheel encoders are connected to the wheel shafts that count the angular rotations and continuously increment them to the prior value. This gives an approximate value of the rotations each wheel undergoes. Dead-reckoning systems use these sensors for determining the pose, based on speed estimation. Dead reckoning is basically a mathematical procedure used for determining the present position. Most commonly, it is termed as Odometry, which is one of the simplistic implementations of dead reckoning. Wheel odometry is the most extensively used navigation technique for mobile robot positioning. It is well known for providing good short-term accuracy and is cheap. The basic concept of Odometry, however, is the integration of incremental motion data over time, which inevitably leads to an accumulation of errors. It is subtle to two types error which are

- **Systematic errors:** These are the errors that arise due to imperfect design and mechanical implementation of the mobile robot. It consists of mostly unequal wheel diameters, uncertainty in wheel-base.
- **Non-Systematic errors:** The mobile robot is susceptible to these kinds of errors which arise due to travel on an uneven floor or due to wheel slippage.

The systematic errors are deterministic, so they can be reduced by carefully designing and implementing calibration factors. Non-systematic errors are mostly non-deterministic errors, leading to uncertainties over time in the position estimation. However, the non-systematic dead reckoning errors can be reduced using **Sensor Fusion**. The basic idea of sensor fusion is to combine readings from various sensors in order to make a decent estimate of robot position and orientation.

2.4 Sensor Fusion

Many researchers believe that it is difficult to capture all the relevant information about the environment or robot's state using a single sensory modality. In order to overcome this issue, one of the most powerful techniques used in the Robotics domain is Sensor fusion. According to Hall and Llinas [26], " *Sensor fusion methods combine data from multiple sensors, and related information from associated databases, to achieve improved accuracies and more specific inferences than could be achieved by the use of a single sensor alone*".

Due to recent development in sensor fusion, many open-source frameworks utilized for fusing the data. One such package used in ROS is robot_localization [2] that uses two types of filter i.e Extended Kalman Filter(EKF) and Unscented Kalman filter(UKF). Because of its flexibility of adding different sensor modules, it serves as one of the most popular sensor fusion platforms in ROS. In this project, EKF is chosen for estimation because of its low computational power, faster convergence, and easy implementation in ROS framework [27].
The Extended Kalman filter (EKF) is a non-linear modified version of Kalman filter used for state estimation [28]. It basically linearizes about an estimate of the current mean and covariance. The non-linear model can be represented by a state and measurement equation:

$$x_k = f(x_{k-1}, u_{k-1}) + w_k \tag{2.3}$$

$$z_k = h(x_k) + v_k \tag{2.4}$$

where, x_k: state vector, u_{k-1}: control input, w_k: process noise, z_k: measurement vector, v_k: measurement noise

Here, the initial state represented as x_0 is known with a mean $\mu_0 = E[x_0]$ and a covariance $P_0 = E[(x_0 - \mu_0)(x_0 - \mu_0)^T]$. Here E is the Expectation operator. If the function f(.) and h(.) in equation 2.3 are linearized, then EKF will be equivalent to Kalman filter, and all the subsequent state will be Gaussian distributed. The most likely state for Gaussian is the mean of the posterior state represented by μ_{k-1}. In general, any non-linear function f(x) can be linearized about m using Taylor series i.e.

$$f(x) \approx f(m) + F_x(m)(x - m) \tag{2.5}$$

where F_x is the Jacobian matrix.

Similarly in EKF, on applying Taylor series to the non-linear function $f(x_{k-1}, u_{k-1})$ around μ_{k-1} in equation 2.3, we get

$$f(x_{k-1}, u_{k-1}) = f(u_{k-1}, \mu_{k-1}) + \underbrace{f'(u_{k-1}, \mu_{k-1})}_{\text{F}}(x_{k-1} - \mu_{k-1}) + \textbf{H.O.T} \tag{2.6}$$

where F represents state transition matrix, which is obtained by computing partial derivative (Jacobian) of f and higher order terms **H.O.T** are neglected. On applying the expectation operator in equation 2.3 and substituting equation 2.6, we get the current state estimate i.e., $\hat{x}_{k|k-1}$ as a function f() which is used to predict the state from the previous estimate. Here, it should be noted that both w_k and v_k, are assumed to be zero-mean normally-distributed random variables and are temporally uncorrelated (white noise), therefore on taking the expectation operator for both is 0 , i.e., $E[w_k] = E[v_k] = 0$. The covariance P_k can be derived from the error of the estimate x_k by μ_k

$$P_k = E[e_k e_k^T] \tag{2.7}$$

where, $e_k = x_k - \mu_k$.
Thus, the covariance of the estimation is the sum of the covariance obtained from the previous estimate and process noise covariance (Q_{k-1}).
The same linearization technique can be followed for the measurement function h in equation 2.3. Again Taylor series is applied around the mean value $\bar{\mu}_k$, i.e.,

$$h(x_k) = h(\bar{\mu}_k) + \underbrace{h'(\bar{\mu}_k)}_{\text{H}}(x_k - \bar{\mu}_k) \tag{2.8}$$

[2] http://wiki.ros.org/robot_localization

where H is the Jacobian representing the measurement matrix obtained by computing partial derivative of h, on further applying the Expectation operator on equation 2.4 and substituting 2.8, the measurement residual Δy_k can be calculated as a function of h() which is used for computing predicted measurement. Likewise, Innovation (or residual) covariance(S_k), i.e., the covariance matrix associated with the measurement error can be calculated. R_k is the sensor noise covariance matrix. Kalman filter (K) uses these matrices for estimation and, thus, determines how much weight to put on the current predicted state and current observation. Using the Kalman matrix, the state estimate and covariance estimate P_k is updated. The overall EKF computation is demonstrated through a flowchart, as shown in figure 2.8.

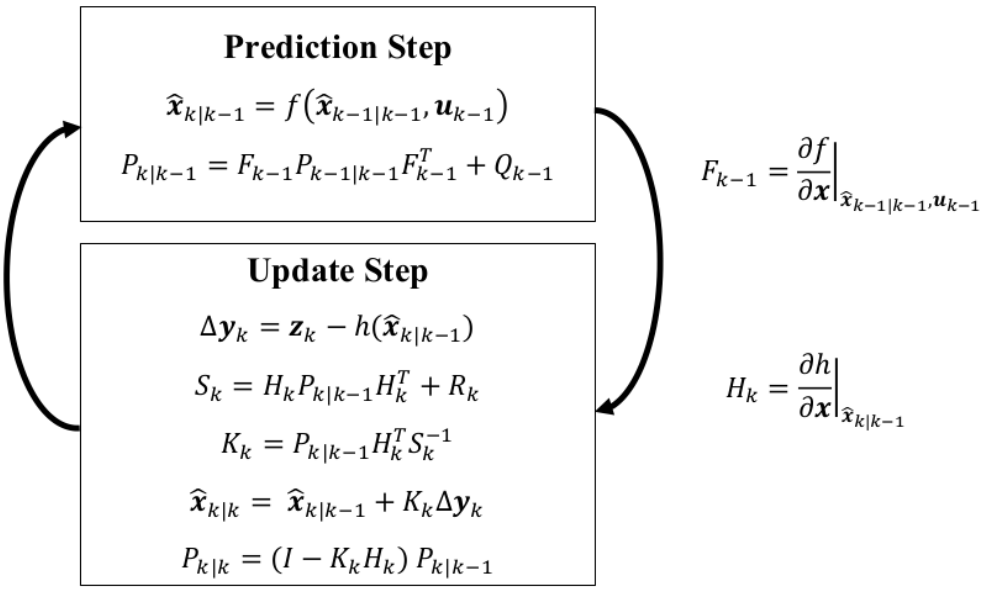


Figure 2.8: Flowchart to show the computation of standard EKF

2.5 Scan Matching

The distance traveled by a robot can be easily known using wheel odometry sensors. However, odometry sensors become inaccurate in longer run due to wheel slippage and drift and hence, are known as dead reckoning. Robot's pose can be improved using exteroceptive sensors such as LiDAR, camera. Scan matching is the process where a laser range scan acquired from a LiDAR is translated and rotated in order to match a priori scan or map. The premise of scan matching is that the most likely pose of the robot is the one that gives the greatest match between the previous scene and the current scene as perceived by the lidar. Hence, it calculates the portion of the two scans which overlaps in the world coordinate frame. Scan matching maximizes the likelihood of the current position and map relative to the previous position and map [29]. It can be written as

$$x_t^* = \underset{x_t}{\operatorname{argmax}} \{ p(z_t|x_t,\ m_{t-1})\ p(x_t|u_{t-1}, x_{t-1}^*) \} \tag{2.9}$$

The first probability distribution function in equation 2.9, is the observation model where the term z_t corresponds to the observation likelihood given the pose of the robot x_t at time t, and the map estimated so far m_{t-1}. The second probability distribution function represents the motion model of the robot, as discussed previously. The '*' mark symbol is used to assume known mapping with known poses. So, it takes the odometry and the current scan and tries to align with the previous scan. Therefore, scan matching is often exposed as an optimization problem.

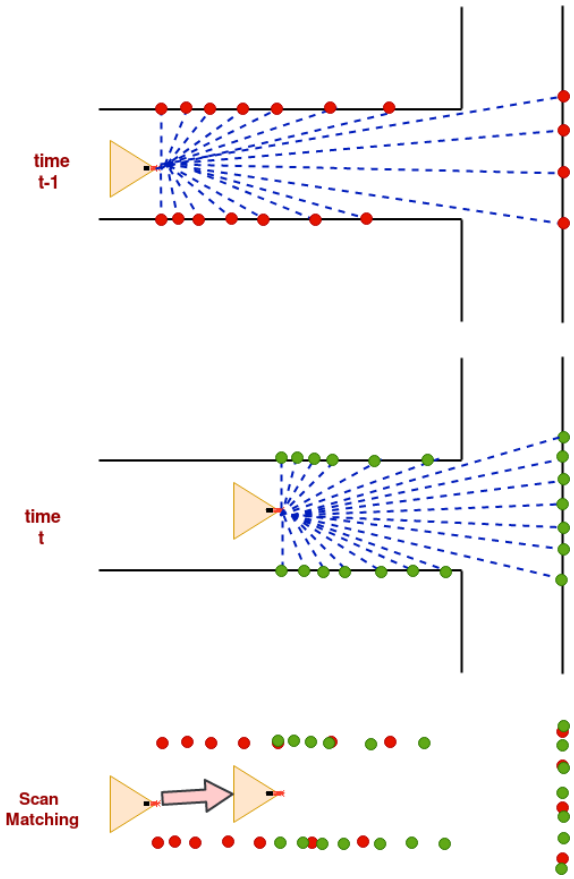


Figure 2.9: Scan Matching method used to determine robot movement. The top figure shows the robot is moving along with a 2D LiDAR in a T-junction pipe. It scans the environment at time t-1. In the middle figure, the robot moves a linear distance of ~1m and scan the environment again at time t. The bottom figure illustrates the scan matching techniques by aligning both the scans at time t-1 and t.

To understand this, consider a robot equipped with a LiDAR and is moving inside T-junction pipe as shown in figure 2.9. The blue dotted lines represent the laser beams emitted from the lidar. The laser beam register the pipe walls through red and green dots. Initially, at time t-1, the robot takes a scan of the environment and register these red dots. Based on the motion model, the robot drives a bit forward and takes another scan and register the green dots. Since the coordinates of the red dots are already stored, it tries to determine the transformation between pose at time t-1 and time t by aligning both the red and green dots. This is known as scan-matching. There are different scan matching techniques, but the most popular one is Iterative Closest Point (ICP), which tries to find the transformation between two point cloud data by minimizing the error function.

2.6 FastSLAM and its derivative

There are many SLAM framework that is already developed, but reusability of such algorithm is limited to the platform, application, and sensors. One such approach is FastSLAM that uses Particle filter to estimate the robot's path and hence build the map. It provides a solution to the Full SLAM problem i.e. calculates the entire robot's path over time. One of the most popular approach is to use Rao-Blackwellised Particle Filter, a FASTSLAM-based algorithm that solves the full SLAM problem [30]. Each particle in RBPF represents the map and possible robot trajectory. Entire RBPF can be generalized in 3 steps:

- *Sampling*: Estimated robot trajectory is obtained by initial sampling from a proposed distribution
- *Importance Weighting*: Each particle is assigned with a weight w which indicates the likelihood of the estimated map and robot trajectory
- *Resampling*: RBPF takes into account particles with higher weight and discards the weaker samples. It then resamples these discarded particles to obtain the updated distribution of the particles that represents the most likely robot's state estimate

FastSLAM has been implemented in one of the ROS packages known as Gmapping [3].

2.6.1 Gmapping

Gmapping is one of the FastSLAM algorithm based on Rao Blackwellized Particle filter (RBPF). This algorithm is often referred to as state-of-the-art for particle filter based SLAM. The Gmapping package was presented on OpenSLAM [4] initially and later wrapped for the use of ROS standard package. Gmapping allows to create 2D occupancy grid maps and localize within the map based on odometry data and laser scan measurements. In this approach, it uses RBPF, where each particle carries its own grid map of the environment and possible trajectory of the robot. RBPF factorizes the SLAM problem by estimating the first robot's pose and then estimating the map i.e.

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) = p(x_{1:t}|z_{1:t}, u_{1:t})\ p(m|x_{1:t}, z_{1:t}) \tag{2.10}$$

In equation 2.10, the posterior probability of robot pose x_t and map m given observations z and control command u is shown as a product of two probabilities. In particle filter based approach, the hypothesis of robot pose and the map are stored in a set of particles where each sample (particle) s is defined as a tuple $s = (x, w, M)$, where x represents the pose vector, scalar weight w and map matrix M. This tuple stores the history of robot pose, map computed so far and particles weight. Based on the particle weight, it indicates how likely the particles represent the map and pose history of the robot given control signal and observation. The grid map constructed from the particles contains the occupancy probabilities of a possible obstacle location for each grid cells. Since each particle representing its known map and poses, the memory requirements and computation are heavily dependent on the number of particles used in the algorithm.
The particles should be used efficiently in order to limit the number of required particles for solving the SLAM problem. In order to do so, the probability distribution function used to generate particles can be further improved by integrating FastSLAM 2.0 [31], which uses advanced filtering and resampling techniques to reduce the number of particles. FastSLAM 2.0 introduces 2 step approach to improve the sampling methodology i.e. it first sample the particles based on the motion model and then perform scan-matching to improve the robot pose and the map of the environment. Moreover, it also helps in improving mapping efficiency, for example, loop closure. Hence, Gmapping takes into account not only the movement of the robot through the motion model but also recent observations from the range sensor to calculate an accurate distribution of particles that reduces robot's pose uncertainty and thus, maintaining the accuracy of the resulting map. The entire process iteratively creates the map and updates the pose of the particles. These iterations can be highlighted using pseudocode for the Gmapping algorithm as discussed in the appendix.

Therefore, the overall Gmapping algorithm can be summarized as below

1. The algorithm is initialized with the initial guess of the robot's pose.

[3] http://wiki.ros.org/gmapping
[4] https://openslam-org.github.io/

2. Using the initial guess of the robot's pose, scan-matching is performed on the built map based on the motion model, hence limiting the search region. Based on the matching score of the scan-matching, a new pose estimate $\hat{x}_t$ is generated. In case scan-matching fails, it uses the motion model to calculate the pose and weight, and step 3 is ignored in such cases.

3. A new pose estimate is derived using the motion model, observations, prior pose of the robot and map built till then. Using Gaussian distribution, particles are distributed near the scan-matching estimate. The weights of the particle w_t are updated accordingly.

4. Particle weight is updated when new observations from the sensors are added to the map.

5. Resampling is performed when an effective number of particles are less than the threshold. The effective number of particles (N_{eff}) can be calculated using equation 2.11, where $i\tilde{w}_t$ is normalized weight

$$N_{eff} = \frac{1}{\sum_{i=1}^{N}(\tilde{w}_t)^2} \tag{2.11}$$

2.7 GraphSLAM and its derivative

GraphSLAM is another way to solve the full SLAM problem, similar to the FastSLAM. It is one of the most efficient, easy-to-implement SLAM-based approach which is most commonly used nowadays [32]. The main idea behind graph-based approach is to use a graph containing nodes that represent the pose of the robot and features in the map. Factor connecting two nodes(robot pose 1 to robot pose 2 or robot to features) is highlighted by an edge. The edges in the graph represent a set of constraints (motion constraints) between successive poses. There are even relative measurement constraints based on the features available, corresponding to robot poses. It uses these nodes and edges based on the sensor data to form a sparse graph. This graph is used to formulate a sum of non-linear quadratic constraints. A maximum likelihood of robot poses and corresponding map can be obtained by optimizing these non-linear constraints in the sparse graph. GraphSLAM extracts the robot poses labeled as X0,...,X3 and four features m1,...,m4 and represents them in a graphical aspect as shown in figure 2.10. The dashed edges link successive robot poses while solid edges link the distance to the features as sensed by the robot. These links form the non-linear quadratic constraint i.e. motion constraints incorporate the motion model while measurement constraints incorporate measurement model.
Since the optimization algorithm is suitable for linear functions, GraphSLAM linearizes these constraints to form a pose graph in order to compute the map posterior. These linearized constraints in the graph can be converted to an information matrix or sparse matrix which can be efficiently used. This matrix contains all the information about the robot's trajectories in an environment and grows in size as the robot moves. This sparseness allows the GraphSLAM to implement a variable elimination algorithm, transforming the graph into a much smaller one that is only characterized by robot poses [23].

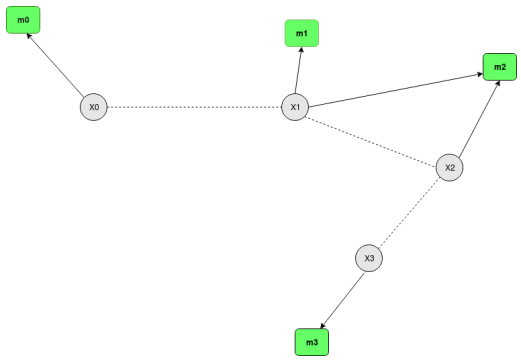


Figure 2.10: Illustration of GraphSLAM. Robot poses are defined as circles while the green square represents the observed features.

Figure 2.11 illustrates the formation of information matrix using graph. Based on this graph, the information matrix can be readily constructed and used in an optimization framework to describe the associations between features and robot poses. Localization problems can be easily modeled using a graphical approach and solution to such a problem can be obtained by minimizing the cost function, which is of the type [33].

$$F(x,m) = \sum_{ij} e_{ij}(x,m)^T \Omega_{ij} e_{ij}(x,m) \tag{2.12}$$

In the equation 2.12, m is the map, x is the vector containing robot poses, e_{ij} is the error function that computes the distances between measurement and prediction, and Ω_{ij} represents associated information matrix. The optimal values m^* and x^* can be obtained by optimizing the equation 2.13.

$$(x^*, m^*) = \underset{x,m}{\operatorname{argmin}} F(x,m) \tag{2.13}$$

Hence, minimizing the target function F obtained by summing all the constraints in the graph, yields the most likely map and robot trajectory.

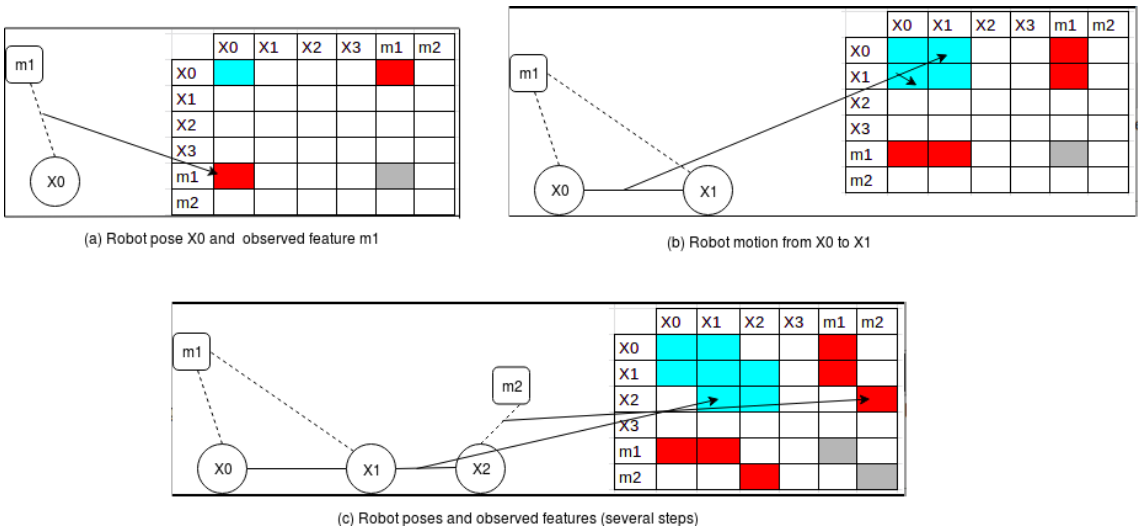


Figure 2.11: Demonstration of acquisition of the associated information matrix in GraphSLAM. Blue square represents robot poses (X0, X1,X2) while features (m1 and m2) w.r.t robot poses are illustrated by red square. All the blue squares are adjacent to each other due to the motion constraint. Grey square shows the features available in the graph.

2.7.1 Google Cartographer

Google Cartographer [5] is an open-source 2D, 3D graph-based SLAM which was later integrated with ROS as a standalone package. It produces an occupancy grid map in 2D and point cloud map in 3D. However, the reusability of such an algorithm is dependent on the platform and sensor configurations. Cartographer can be viewed as two distinct but linked subsystems ashown in figure 2.12. The first one is called Local SLAM (also known as local trajectory builder). It serves as the frontend for the Cartographer. In Local SLAM, it acquires data from range sensors and other proprioceptive sensors such as wheel encoders, IMU to form submaps. A submap is a small chunk of the environment formed from many laser scans. These submaps are just a downscaled representation of a portion of the map that is sufficiently accurate for short representations. As discussed earlier in GraphSLAM, all the relative constraints can be witnessed while building the submaps.

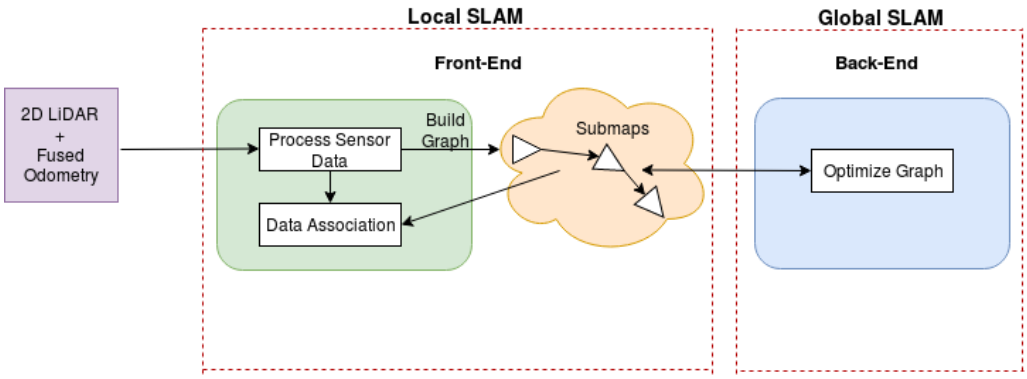


Figure 2.12: Division of Google Cartographer algorithm into Local SLAM (Front-end) and Global SLAM (back-end). The constraint-based weight parameter is added to the Local SLAM and parameters are tuned for loop closure in Global SLAM

The other subsystem is called Global SLAM, which serves as the backend for the Cartographer, and its sole purpose is to find loop closure constraints. Multiple scan-matching is done against these recent submaps, due to which it accumulates pose estimation error. Pose optimization is run frequently to avoid the accumulation of errors. Once a submap is finished, no more laser scans are inserted to it, and then all the finished submaps take part in scan matching for **loop closure** (revisiting the same location on the map again) [34]. The scan matcher iterates automatically through the finished submaps to find a laser scan. If a relatively good match is identified in a search window from an estimated position in the submaps, it is added to the optimization problem as a loop closure constraint. Hence, solving this optimization problem helps in closing the loop immediately when a location is revisited. Figure 2.13 shows the architecture for Google Cartographer.
Voxel filter in Cartographer helps in downsampling the raw points hit by the laser into cubes of fixed size, and it keeps the centroid of each cube. Cartographer also uses an adaptive voxel filter. This filter attempts to determine the optimal voxel size (under a maximum width) to obtain a target amount of points. The pose extrapolator block is that it accumulates the other sensors along with range finder to estimate where the next scan should be added into the submap. In order to avoid insertion of too many scans, it goes through the motion filter block to check if motion is found between two scans. The sparse pose adjustment (SPA) block in the Global SLAM runs in the background whose function is to rearrange submaps so that they create a consistent global map. In constructing a 3D map, 3D range sensor is used along with proprio-

[5] http://wiki.ros.org/cartographer

ceptive sensors. The integration of IMU is a requisite to be able to sense gravity (z-direction). The 3D scan is exported as a 3D point cloud consisting of a combination of the produced pose graph and the raw lidar point cloud information. The lidar produces a sequence of 3D point cloud frames that represent the environment at each position of the robot. On a higher abstraction layer, it can be summarized that the role of the Local SLAM is to produce excellent submaps, and that of Global SLAM is to bind them consistently together.

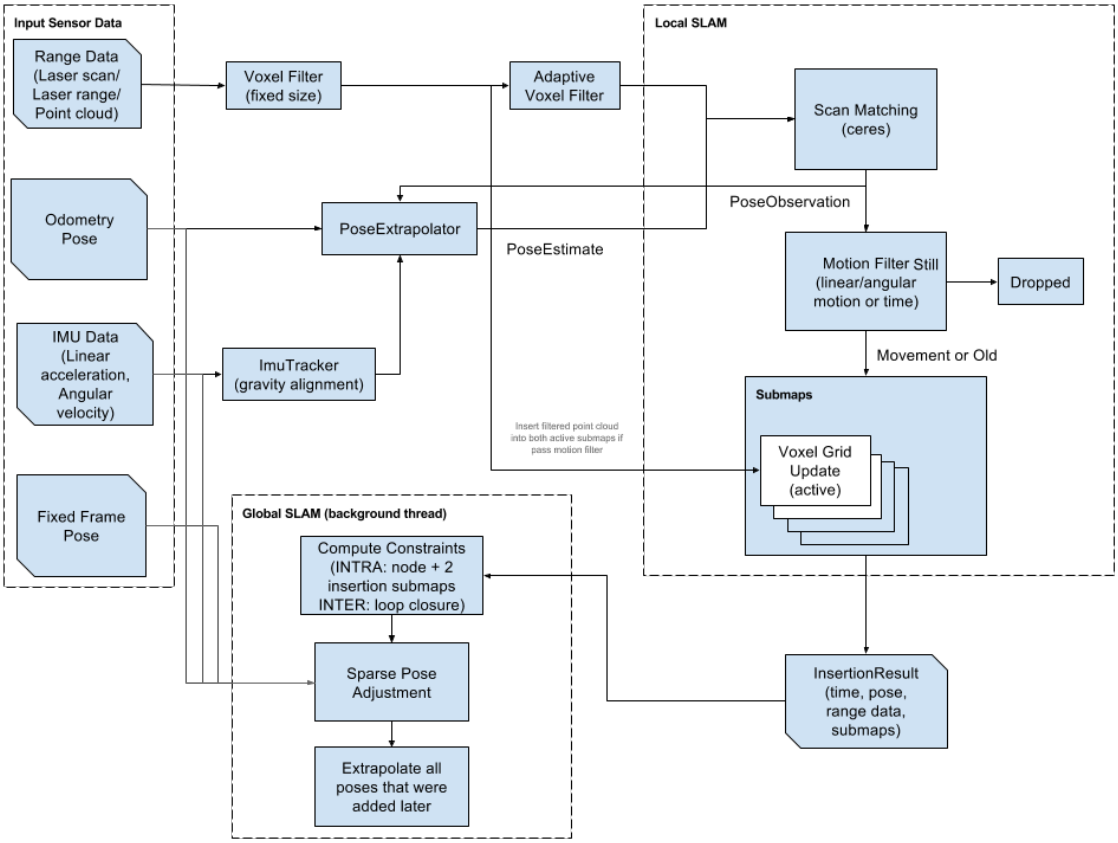


Figure 2.13: High-level architecture of Google Cartographer [4]

2.8 Other Localization technique: Adaptive Monte Carlo Localization(AMCL)

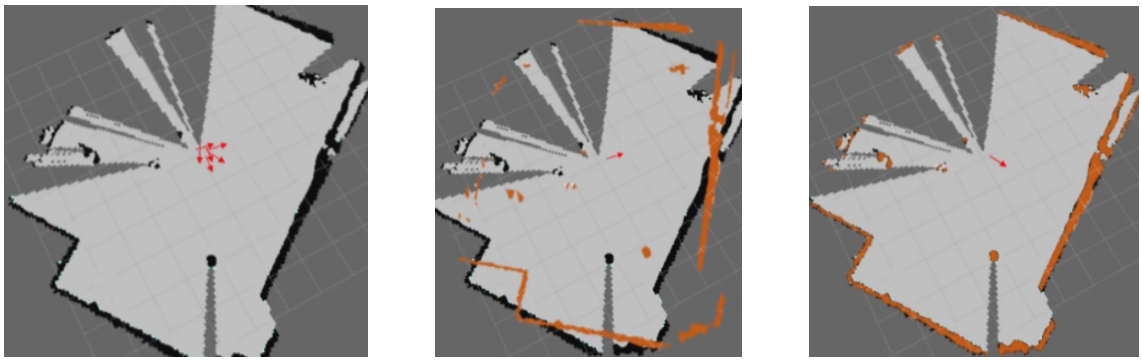
The most common algorithm used in the field of robotics is Monte Carlo Localization, an algorithm that uses particle filter for localization [35]. Localization involves estimating the position and orientation of the robot, collectively called as pose, as it moves and senses through the environment. So, each particle contains its own map of the environment and a possible state, i.e., a hypothesis of where the robot is. The algorithm typically starts with a random distribution of particles throughout the pose space. When the robot starts to move, the particle shifts and tries to predict its new state. Whenever the robot observes a distinguishable feature, then the weights of the particles around that feature increases with a Gaussian distribution pattern. Now when the robot moves, the particles are re-sampled with equal weights based on recursive Bayesian estimation, however, the number of particles in the region where the previous belief was high will be more compared to the rest of the space. This iteration continues until the particles converge around the current pose of the robot.
Consider the 2D occupancy grid map generated by Gmapping, as shown in figure 5.5. The arrow indicates the pose obtained from wheel odometry, which is inaccurate. So in order to generate a set of hypothesis, discrete particles are drawn from Gaussian distribution with the mean value being the odometry pose data and with some covariance that defines the size of the

region around which the particles are drawn as seen in the left figure 2.14a. For each of these particles, laser scan is orientated, and a map is built with respect to the particle pose data as shown in figure 2.14b and 2.14c and then a correlation score (S) is computed using the equation 2.14

$$S = \frac{\sum_m \sum_n (A_{mn} - \bar{A})(B_{mn} - \bar{B})}{\sqrt{(\sum_m \sum_n (A_{mn} - \bar{A})^2)(\sum_m \sum_n (B_{mn} - \bar{B})^2)}} \tag{2.14}$$

where A = map, B = laser scan, m and n are the (x,y) coordinates of the pixels, $\bar{A}$ and $\bar{B}$= mean values of the pixels of both the map respectively. Since it is an Occupancy Grid map, the black pixels have a value of 1, and white pixels have a value of 0. So the correlation score basically calculates how much the black, and orange pixels overlap each other. This process is repeated for each particle, and the respective correlation score is noted. The particle that aligns well with the map has a high correlation score compare to others. So, the particle with the highest correlation score is chosen as the pose of the robot for the current state. This process of correlation is repeated for each sensor update. The particle weights are updated using the relation

$$W_t = W_{t-1} * S \tag{2.15}$$



(a) **(b)** **(c)**

Figure 2.14: Evaluation of Scan Correlation. In left image (a) particles align using the odometry data, (b) and (c) shows the alignment of particle with respect to the map based on the evaluation scan correlation.

In this way, the performance of each particle can be observed and likewise, draw more particles from that neighborhood when needed. Resampling is done in order to maintain the number of particles, as discussed before. But such type of localization is computationally expensive as it has to calculate the correlation score for each particle iteratively. So on increasing the number of particles will eventually increase the computation time and reducing the number of particles might result in poor localization.

To improve such a form of localization, AMCL is used more often, which optimizes MCL by sampling the particles in an adaptive manner based on error estimates using the Kullback-Leibler divergence (KLD) [36]. It basically controls the number of particles based on the difference in odometry and particle-based location. So, initially the particle cloud is large when the robot starts from its initial position due to high uncertainty in the pose but as the robot moves the particles converge and the particle cloud size reduces as shown in figure 5.6. Thus, the overall performance of MCL is improved using KLD sampling.

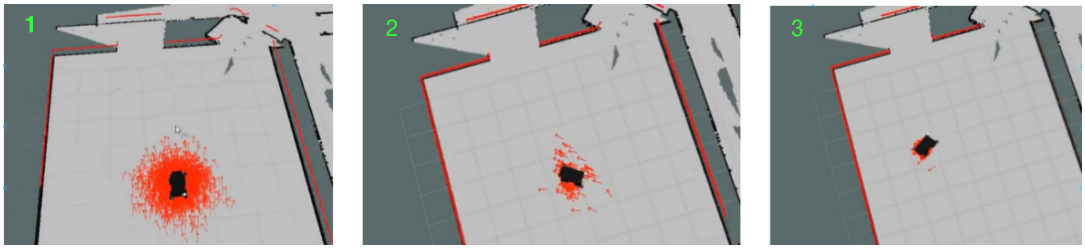
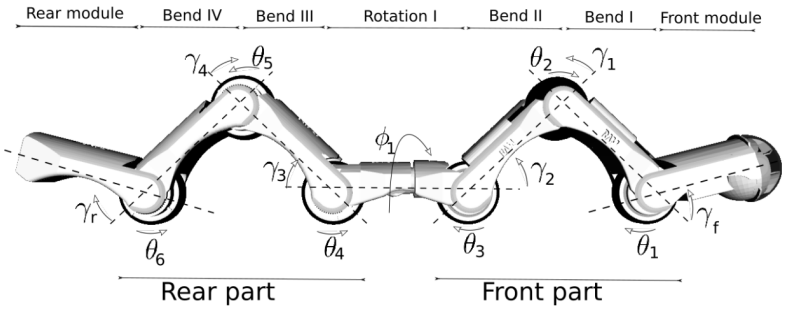


Figure 2.15: Demonstration of AMCL in ROS. Left image 1 shows dense particle cloud due to high uncertainty. Image 2 exhibit the particle alignment once the robot starts moving while image 3 shows the convergence of the particles.

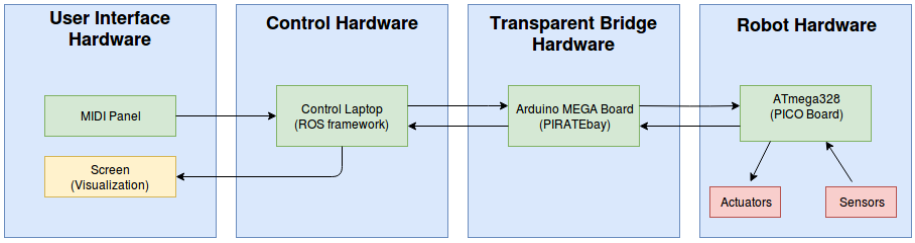
2.9 PIRATE

Pipe Inspection Robot for Autonomous Exploration (PIRATE), a six-wheeled snake-like robot designed by E. Dertien [7] at RAM with the sole purpose of inspecting small diameter gas but, lately it has evolved into a much more versatile inspection robot. Lately, PIRATE has gone through many design changes. The current version of PIRATE model has seven physical modules

- 4 Bending module
- 1 Rotational module
- 1 Front module
- 1 Rear module



(a) Modules inside PIRATE [7]



(b) Hardware Architecture. Adapted from [37]

Figure 2.16: PIRATE Model

Figure 2.16a shows the kinematic model of the current PIRATE. The hardware architecture of the PIRATE can be visualized in figure 2.16b. It contains several PICO boards (PIRATE control) for actuating motors and helps in processing up to two sensors. An Arduino MEGA board serves as the centralized master who acts as a transparent bridge between the high level (laptop) and

low-level (PICO) control. High-level control is provided by ROS framework. Input from the users can either be given via the command line or via a MIDI panel. All wheels and joints are equipped with position and present sensors and are powered by DC micromotors [8].

2.10 Software Framework

This thesis was conducted in Linux (Ubuntu 16.04 LTS) with ROS (Robot Operating System) and Gazebo as a core platform for execution.

2.10.1 Robot Operating System (ROS)

ROS is an open-source, software framework that allows the user to write applications that can be easily operated on robotic hardware. It offers various functionalities such as package management, hardware abstraction, communication between different processes. **Rviz** is a 3d-visualizer tool used for displaying sensor data and state information from ROS. Figure 2.18 shows the general ROS architecture. The key elements in ROS are

- **Roscore** is the main process that manages all of the ROS system.
- **Node** is an executable that uses ROS to interact with other nodes. They help in controlling different functions.
- **Messages** are data that provide information to nodes. Node communicate via messages.
- Messages in ROS are routed via **Topics**. Nodes can simultaneously publish multiple messages to a topic and subscribe to a topic to receive messages.

These nodes are typically used to process raw sensory information, for example, from a camera, motor encoders, distance sensors, etc. Once the sensory information is processed according to the needs of the desired application, it is published to a ROS topic in the form of messages. The reason behind choosing the ROS framework is that apart from easy sensor integration, it also includes many of the features needed, such as robotic control and SLAM algorithms. ROS makes it easy to use software written in either Python, C++. Moreover, since the PIRATE high-level control is implemented in ROS, this thesis utilizes the ROS platform as its primary foundation.

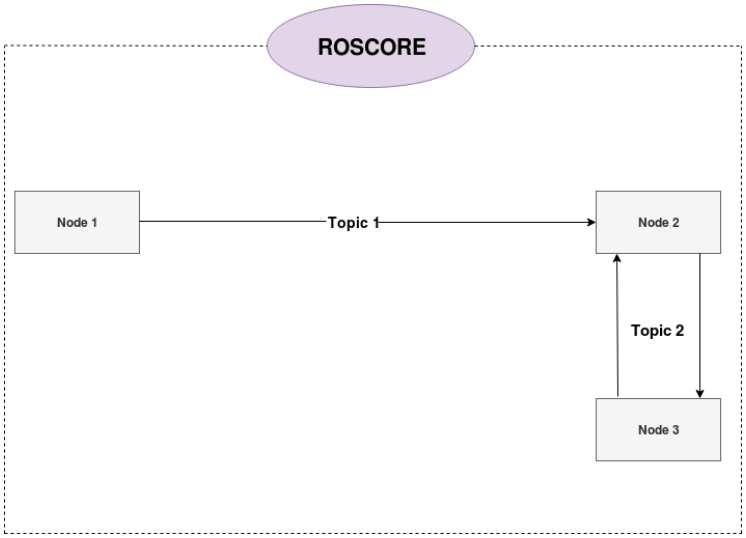


Figure 2.17: ROS Architecture. Node 1 is communicating to node 2 via topic 1 and node 2 is communicating with node 3 via topic 2. ROS process operates under a master called ROSCORE. Communication in ROS corresponds to publishing and subscribing of a given topic.

ROS provides transformation (tf) libraries, which enables the ROS node to visualize coordinate frames and transform data between different frames. It is a standardized way of managing the position of the robot. There are two types of tf nodes i.e., publishers that publish transform between coordinate frame and listeners, which listen to tf and cache all data heard. A system can have multiple broadcasters which provide data about a different part of the robot.
The different coordinate frames defined in ROS which are used during this assignment are discussed below:-

- base_link:- This frame is also regarded as the 'robot's frame'. It is attached rigidly to the center of the mobile base in between the two wheels. All the integrated sensors on the robot, as well as external, are linked to the base_link. Using this frame, it is very easy to visualize the robot's position in a given map.
- Odom:- Odom frame is the frame located at the point where the robot started moving. The position and orientation (pose) of a robot in the odom frame is smooth and continuous. The odom frame is quite useful as it is accurate for the short term and mostly serves as a short-term local reference, but drift due to the wheels' slip makes it a poor frame for long-term reference.
- map:- Just like odom frame, the map frame also serves as a reference frame with the only difference is that the pose of the robot, relative to the map frame, does not change significantly during the long-term as compared to odom frame. All the SLAM process builds in this frame.
- sensor frame:- Mostly, these frames depend on the type of sensors. For example, a 2d-lidar has only one frame, while stereo cameras have multiple frames. Depending on the usability, only the necessary frames are chosen, which are required for data transformation i.e., either from sensor frame to base_link or from sensor frame to any other reference frame.

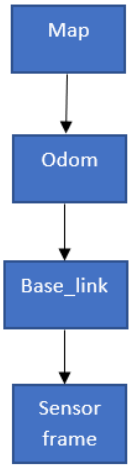


Figure 2.18: Illustrates the tree representation showing the relationship between parent frame and child frame as commonly seen in mobile robots.

2.10.2 Gazebo

It is the simulation environment which provides a set of ROS API's that allows users to model robots and get various sensors information. A real-world environment can be easily constructed in Gazebo, as shown in figure 2.19. Using ROS API, it enables the user to manipulate

the properties of the environment over ROS, as well as spawn and introspect on the state of models in the environment.

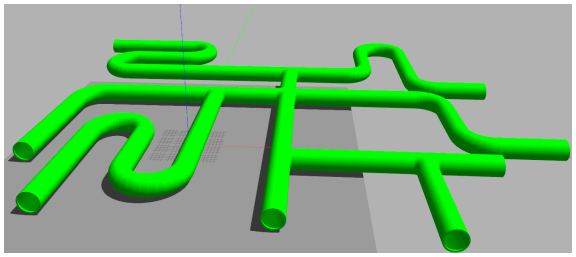


Figure 2.19: Representation of a pipe model in Gazebo

3 Analysis and Design

In this chapter, the main focus is to analyze the most appropriate solution to meet the project objectives and expectations. The first section highlights the robot model, which will be used for simulation. The other sections then introduce the main approaches used in this thesis, together forming a methodology for each of the research questions as described in section 1.2.

As mentioned earlier in the reported literature in section 1.3, a lot of studies has been performed on SLAM algorithms within a closed network, such as pipelines, tunnels, caves, etc., but mostly with the assumption that there are some textures or features in the environment. The problem arises when the environment is textureless and has sparse features. Under such circumstances, it becomes challenging to implement the SLAM framework. Moreover, none of them compared different SLAM algorithms for such an environment. Such a comparison would be of great interest when dealing with a textureless and constrained environment like in the case of a pipe network. In this thesis, therefore, it emphasizes how different SLAM frameworks can be constructed and contrasted for such a challenging environment. To summarize, the goal of this chapter is

"To analyze the research objectives and find a suitable alternative for constructing a SLAM framework inside pipelines."

3.1 Robot Model

PIRATE has a complex mechanical framework with several couplings and joints. It also has several wheels that clamp to the pipe wall while moving forward or backward. It is recommended that the algorithm be evaluated first in simulation and then on a real robot so that design flaws can be investigated. However, Gazebo is unable to capture the motion dynamics of the PIRATE as it leads to frequent vibrations when its wheels clamp to the pipe wall. Two possible solutions are

- Design of a custom simulator that captures all the motion dynamics of the PIRATE
- Use a robot model that would work with the existing simulator but still models the dynamics required for the SLAM problem.

Designing a custom simulator is rather difficult and is likely to cost a considerable amount of time and effort. Moreover, the SLAM approach used in this thesis requires an odometry measurement irrespective of the motion model that formulates it. Thus, even in the absence of a complete dynamic model, it is still possible to implement SLAM framework. In principle, it is possible to construct the SLAM framework independent of the robot model using similar sensor data and the environment. Hence, choosing a 4-wheeled differential drive robot is sufficient to carry out the simulations.

3.2 Environmental Perception

A truly autonomous robot is one that can perceive its environment, make choices on the basis of what it perceives, and then actuate a motion within that environment. These perceptive capabilities are vital for mobile robots, which are imparted through the exteroceptive sensors. Thus, perception is a measurement-based process for understanding the environment. To move efficiently in an unknown or uncertain environment, a mobile robot must use observations made by these external sensors to build data for SLAM applications. LiDAR and camera are the two most frequently used sensors in mobile robot navigation for perceiving the environment. The range and vision-based system have become an essential element of robotic autonomy and navigation because of its accuracy, range, data interpretation, and simple integration.

These systems are frequently used in tunnels, subterranean caves and pipelines with abundant features and textures for SLAM, path-planning, exploration [38], [39], [40], [41]. However, it will be interesting to see how these sensing techniques could be beneficial in localization and building maps in a textureless pipe environment with sparse features. In this thesis, therefore, we first investigate how these sensors perform in such a challenging environment by proposing a solution to the first research question,
Is it possible to build a map and localize inside the pipe network by only using a range or vision sensor on the robot?

3.2.1 Visual and LiDAR-based system inside pipe network

LIDAR, or Light Detection and Ranging, is a remote sensing system that determines the range of the target by sending a series of pulses and checks how long a laser pulse takes to reach the object and bounce back to the sensor. Due to the precision and simplicity of the measurement principle, data interpretation is easy. These sensors typically have a range of 30 to 120 meters with a minimum sweep angle of 60 degrees, hence regardless of the angle of installation, it supplies reliable measurement data for a whole host of tasks; Whereas a visual-based system consists of a camera that uses image processing techniques to extract valuable information from an image. Both LiDAR and camera try to extract key features from the environment and then use a suitable algorithm to provide absolute positioning of the robot, i.e., LiDAR odometry and visual odometry. The LIDAR odometry is generally used to estimate the position of the robot based on planar laser scans. This odometry works on the principle of scan matching technique, as discussed in section 2.5, i.e., it takes the laser scans and measures the motion of the lidar for two consecutive sweeps.
Similarly, visual odometry also tries to estimate the pose of the robot based on some measurements from an image(s). The most common vision-based feature extraction algorithms are described in Appendix E. The working principle is somewhat similar to scan matching, but instead of scans, it uses features from an image for estimation. Generally, visual odometry includes these steps for localization [42], namely,

1. Acquisition of camera images
2. Extraction of features (corners, textures) from images
3. Perform matching between consecutive image frames
4. Calculation of the pose by measuring the displacement of the pixel between frames

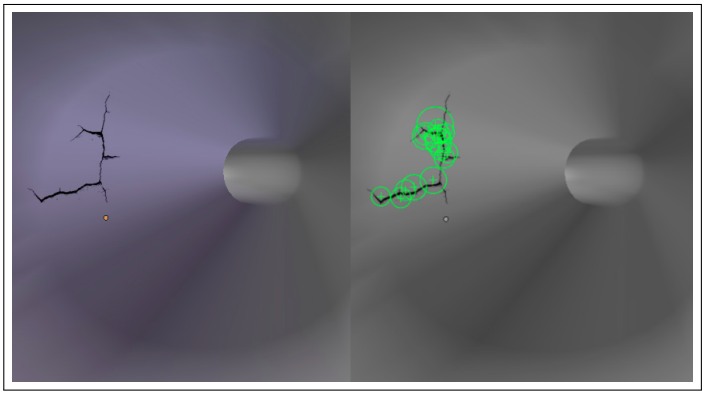
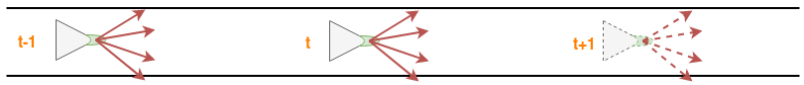


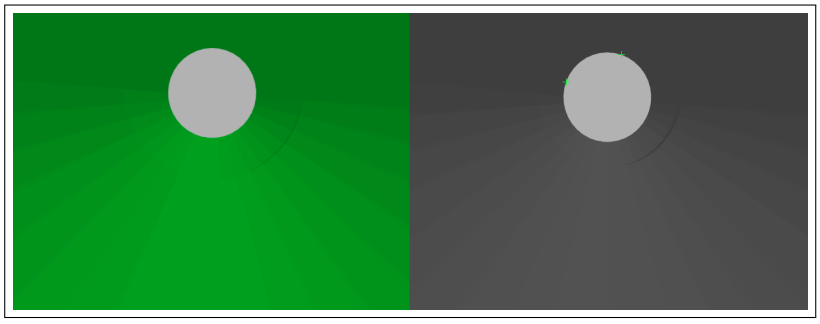
Figure 3.1: Demonstrates how textures in a pipe can be identified using a vision algorithm. The left image shows the original pipe, while the right one is the grey-scaled version used for image processing. The green circle in the image shows how the camera can detect the cracks inside a pipe.

In a pipe network, the most critical aspect of mapping the environment is how well the robot can localize itself within the pipe using these exteroceptive sensors. Also, it depends on how precisely the junctions and the side-walls of the pipe are detected. In the reported literature, most of the pipes used had sufficient textures and features. The camera can certainly be useful in a textured pipe as it can easily detect these textures using vision algorithm as shown in figure 3.1; Whereas LiDAR can be useful in pipes with small diameter changes as it can identify these changes based on the time taken by the laser to rebound. In the presence of sufficiently salient features (junctions, bends), both sensors can be useful in pose estimation.
However, the main challenge arises in the absence of textures and sufficient features. In the case of a textureless pipe with sparse features, the pipe segment can be divided into two parts for a simple analysis of the sensor behavior.

- The first part is considered to be a straight long uniform pipe without any textures. When moving in such an environment, both these sensors can be ineffective since the robot will perceive the same environment at every time instant, as demonstrated in figure 3.2. In figure 3.2a, it shows how the LiDAR will scan the same point at every time instant until it discovers a feature in the environment. This scenario is similar to a robot moving in a long narrow hallway with no corridors. Figure 3.2b illustrates lack of salient features in the environment. In such situations, both LiDAR and visual odometry could not provide accurate odometry information, as laser or image data do not provide valuable information from the environment. Thus, robot localization will fail, and as a result, will have a large impact on map building. To overcome such situations, the use of Proprioceptive sensors like wheel encoders can be handy.



(a)



(b)

Figure 3.2: Illustrates a straight textureless, uniform pipe scenario. In figure (a) demonstrates a robot equipped with a 2D LiDAR moving inside a straight uniform pipe. Different robot position is shown at a different time instant. In figure (b) illustrates the camera view inside a straight textureless pipe. The left image shows the original pipe, while the right one is the gray-scaled version used for extracting features from the image. Green '+' marks show the identification of the far located edges of the pipe.

- The second part can be considered to be a pipe segment with sparse features, which can be a junction, bend, or small diameter change. In this case, a T-junction pipe is considered. When the robot is near to the junction, it is easier to detect these junctions using LiDAR or a camera. Figure 2.9(b) shows one similar situation when the robot is equipped with a LIDAR. Moreover, on using a camera, it is able to identify the edges of the pipe junction, as seen in figure 3.3, on applying the vision algorithm. Using these

features, visual odometry or LiDAR odometry can be the right choice for pose estimation of the robot.

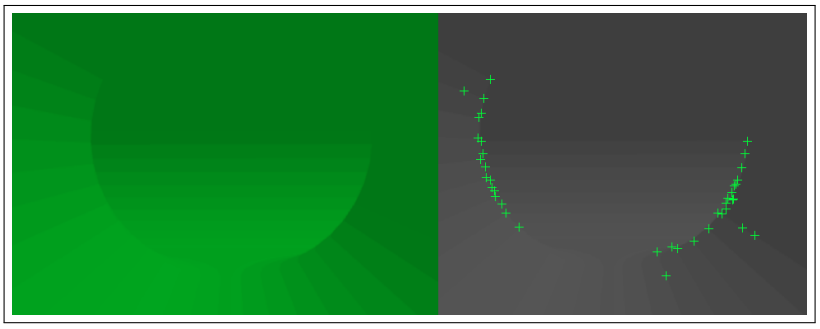


Figure 3.3: Illustrates how visually features can be extracted when the robot is near to a T-junction pipe. Again left image shows the original pipe while the right one is the grey-scaled version used for image processing. The green '+' mark in the image shows the identification of features in the image.

Based on this analysis, it is evident that both LiDAR and camera cannot be used as a stand-alone sensor for robot localization and map building inside a textureless pipe with sparse features. However, a LiDAR can perform better than a camera in such a challenging environment since they are effective in low-visibility. Insufficient illumination is one of the biggest challenges for optical sensing systems. These pipelines are poorly lit environments, which can drastically affect the visual odometry. The most common solution is to provide adequate lighting through LEDs, but because of this, it creates shadows that can affect the estimation of the pixel displacement between image frames and, thus, contribute to errors in estimating the robot's position. Moreover, the computation of LiDAR data is much easier as compared to a stereo or a monocular camera, which requires continuous batch processing of multiple raw images at once.

Nevertheless, both of these sensors can fail if the robot moves in a straight uniform, textureless pipe. Therefore, in such situations, the use of odometry data from the wheels becomes quite significant. It can be used along with LiDAR to localize in different pipe segments. Hence, dead reckoning is required.

3.3 Robot Localization inside pipe

Localization is one of the most crucial competencies needed by an autonomous mobile robot because knowledge of the robot's pose is a necessary precursor to decision-making on future actions. Localization involves more than merely deciding an actual position in space; it means building a map, then defining the position of the robot relative to that map. The main goal of the PIRATE is to inspect cracks, corroded portions, and other imperfections inside the pipeline. Nevertheless, the exact location of these defects can only be determined if the position of the PIRATE is well known. One such scenario is demonstrated in figure 3.4, where the location of the corroded portion inside the pipe can be known by estimating the position of the robot within the straight pipe map. Thus, instead of replacing the entire pipeline, which could be expensive and unnecessary, only the damaged portion of the pipe can be replaced. This estimation in position is governed by the robot onboard sensors that play an integral role in localization. It is because of the inaccuracy of these sensors that localization poses difficult challenges. As mentioned in the previous section, neither LiDAR nor camera can independently map and localize inside the pipe environment. The use of proprioceptive sensors (Wheel Encoders, IMU) can aid in estimating the position of the robot, particularly when moving in a long straight pipe.

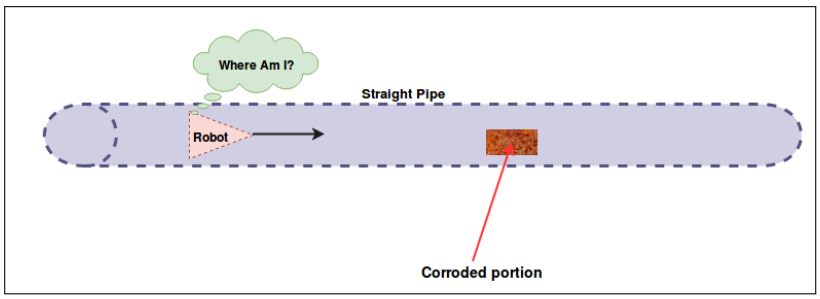


Figure 3.4: Illustrates how robot localization is crucial for identifying defect location in a pipe

Also, in earlier works, the PIRATE used laser projection to detect the mitre bend inside the pipe, as discussed in chapter 1. However, when the PIRATE reaches very close to the bend, it is not able to detect the curve since it goes out of the camera view, and it makes it difficult for navigation inside the pipe. Therefore, even in such situations, the use of wheel odometry information becomes quite relevant. Wheel odometry measurements are incrementally used in conjunction with the robot's motion model to find the robot's current position with respect to a global reference coordinate system (Odom frame). Nonetheless, these wheel encoders are susceptible to various types of error, and the most common type is wheel slippage, as described in chapter 2. Using additional sensors such as IMU can help to reduce the accumulated error. Therefore, in this section, we explore how odometry data can be further enhanced and used for localization by proposing a solution to the second research question, i.e.,
How can a low-cost IMU (Inertial Measurement Unit) sensor measurement help to improve the localization of a robot in a pre-built map of the pipe network?

3.3.1 Performance Improvement in Localization inside a pipe network

An IMU is a low-cost sensor board that provides accurate orientation data. The sensor board consists of 3DOF gyroscope (for measuring rotational velocity), 3DOF accelerometer (for measuring the direction of gravity), and can have a 3DOF magnetometer as well (for sensing Earth's magnetic field). For the PIRATE, an IMU has been integrated into the middle module for determining the correct orientation along the pipe center. Although they can provide reliable orientation information, the translation measurement, which can be obtained by integrating the linear acceleration sensor measurement twice, is highly susceptible to drift. To overcome this problem, one probable solution is **Sensor fusion**, as discussed in section 2.4, i.e.,combining different sensor data for state estimation. In this case, the estimated state would be robot pose, the error of which would be comparatively less than that of the measured data from a single sensor. Therefore, the measurement from the wheel encoder can be fused with the IMU to provide a better estimation of robot pose. Since the dynamical model for both differential drive [43] and PIRATE [44] is non-linear, EKF is chosen for estimation.

Assuming the robot is moving in a planar environment, the effects of z-position, roll and pitch angle can be excluded. Hence the state estimation vector can be written as x_k : $[x, y, \phi]$. The measurement vector z_k consists of measurement data coming from different sensors i.e.
Wheel Encoder: [x, y, ϕ, $\dot{x}$, $\dot{y}$, $\dot{\phi}$]
IMU: [α, β, ϕ, $\dot{x}$, $\dot{y}$, $\dot{z}$, $\dot{\alpha}$, $\dot{\beta}$, $\dot{\phi}$, $\ddot{x}$, $\ddot{y}$, $\ddot{z}$]

Here x, y, represents the pose of the robot, α, β, ϕ are orientation, $\dot{x}$, $\dot{y}$, $\dot{z}$ denotes their linear velocities $\ddot{x}$, $\ddot{y}$, $\ddot{z}$ are linear acceleration and $\dot{\alpha}$, $\dot{\beta}$, $\dot{\phi}$ are angular velocities respectively.

Table 3.1: Sensor configuration passed to the EKF to get the state estimate

Sensors/Parameters	x	y	z	α	β	ϕ	$\dot{x}$	$\dot{y}$	$\dot{z}$	$\dot{\alpha}$	$\dot{\beta}$	$\dot{\phi}$
Wheel Encoders	0	0	0	0	0	0	1	1	1	0	0	1
IMU	0	0	0	1	1	1	0	0	0	1	1	1

Table 3.1 shows the sensor configuration used for fusing odometry from wheel encoders and IMU. The configuration parameters are chosen on the basis that the translation information is taken from the wheel odometry, and the orientation data is chosen from the IMU because only the orientation information from the IMU can be trusted. In the configuration setup, a parameter with value 1 is used for final state estimation. These measurements are combined and fed to the EKF that provides the best state estimate for the robot. But it should be kept in mind that not all parameters are used for fusion since it leads to undesired robot behavior and unstable pose estimation from EKF.

For the localization problem, as discussed in section 2.8, Monte Carlo Localization (MCL) is the most commonly used algorithm for estimating the pose of the robot. However, the critical problem with the MCL is to preserve the random distribution of particles throughout the state space, which goes out of control if the problem is high dimensional. Due to this reason, it is much easier to use an adaptive particle filter that converges much faster and is computationally efficient. Since AMCL is based on KLD sampling, having reliable fused data between wheel encoders and IMU can result in better pose estimation. Less error in odometry would result in faster convergence as the particles are regulated based on the difference in odometry and particle position. Figure 3.5 shows how the fused proprioceptive sensor data (wheel encoder and IMU) can be combined with exteroceptive sensor data (2D LiDAR) for AMCL implementation. Here, it is presumed that the map of the pipe is readily available through one of the SLAM algorithms (Gmapping or Cartographer). The Map_server is a rosservice to provide the built map to AMCL. The initial pose of the robot must be imparted to the AMCL to distribute the particles in the vicinity of that area. Also, it is possible to provide a complete covariance matrix that illustrates the uncertainty of the pose estimate with which AMCL starts operating.

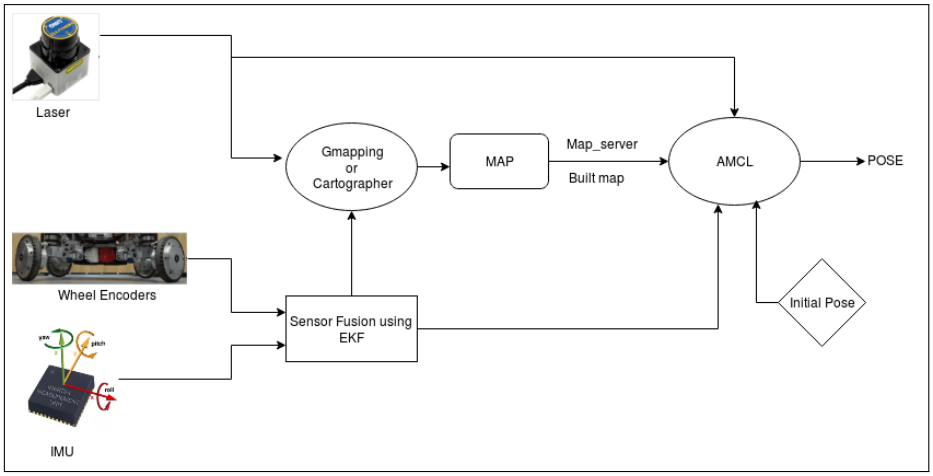


Figure 3.5: Workflow for Implementation of AMCL

In the practical application of the AMCL, a large number of samples are generated when the robot starts from its initial position as there is high uncertainty. When the robot starts moving the posterior converges and this reduces the number of samples for pose estimation. Therefore, on using a low-cost IMU sensor, odometry data is anticipated to be more accurate and can be further utilized for AMCL localization within the pipe network. Experiments will be performed

in the later chapter to validate this hypothesis. Thus, the location of the defect in the pipe can easily be determined by correctly estimating the position of the robot, thereby minimizing maintenance time in industries.

3.4 Particle filter-based SLAM for in-pipe environment

To navigate in an environment, a robot needs a map in which it can localize itself. For instance, a map can be used for path planning or providing an intuitive visualization for a human operator. In the previous section, it was assumed that a pre-built map was available for the use of localization. However, it is quite challenging to perform both mapping and localization at the same time. Nonetheless, accurate localization in an unknown area requires accurate mapping. Many SLAM implementations have been created for various platforms and applications. Gmapping, an RBPF-based algorithm, has been one of the most widely used SLAM frameworks for indoor navigation. In the reported literature [20], [45], most of the pipeline inspection process does include particle filter based-localization, and their performance is quite remarkable. The advantage of using RBPF-based Gmapping is that it factorizes the joint posterior into two separate steps: firstly, it estimates the trajectory of the robot using particle filter and, secondly, it uses the obtained trajectory to estimate the posterior of the map ('mapping with known poses'), i.e., occupancy grid mapping. After the analysis of the previous two research questions, it was concluded that a 2D LiDAR combined with fused odometry could be useful in both mapping and localization inside the pipe network. Because Gmapping uses 2D LiDAR and odometry data as input, it can be an appropriate choice for SLAM implementation inside the pipe network.

The reusability of such an algorithm is often restricted due to the platform and sensor-specific dependencies and optimizations. Changes must be implemented to these algorithms for a specific application. But, using the algorithm directly inside the pipe network may not be suitable as it could lead to an inaccurate map and pose error. No SLAM algorithm can give a completely accurate result due to sensor inaccuracies. The goal, however, is always to improve the performance of these algorithms so that it can reproduce the physical world as much as possible. Therefore, in this section, we investigate and find a solution to the third research question, i.e., *How can the mapping accuracy of the Gmapping algorithm be improved for different pipe segments?*

3.4.1 Design of weight-switching Gmapping algorithm for in-pipe environment

The hypothesis developed at the end of the first research question was to use odometry data from the wheels when the robot moves in a straight pipe segment and to use LiDAR data when the robot is near a junction or bend. Using this hypothesis, prior observations are made by running the Gmapping algorithm directly onto a mobile robot inside a long straight pipe with a junction. These observations can be listed as

- When the robot is moving inside a long straight featureless pipe, the scans coming from LiDAR could see two side-pipe wall surfaces. The scan matcher uses these scans equally well to match and correlate for hypothetical poses in a straight line through the pipe surface. The generated matching score by the scan matcher is utilized by the Gmapping. It picks up the highest score for estimating robot pose and constructing the map. Large jumps could be noticed in the pose estimate, making it remarkably poor for constructing a map inside the pipe. This happened because of the narrow hallway problem, as discussed in chapter 3.

- The scan matching score is always positive, and it resulted in a higher scan matching score near the junctions ranging from 600-750 and a relatively lower score ranging from 350-480 when the robot is moving in a straight pipe with no features. The average match-

ing score is plotted against the distance traversed by the robot inside the pipe, as shown in figure 3.6. Initially, the matching score is high since the first scan covers the front section of the pipe that contributes as a feature to the environment. When the robot is in the middle section of the pipe, which is just a long straight pipe with no features, the matching score drops down to less than 400. In the end, section, when it discovers a junction, there is a sudden rise in the matching score.

Based on these observations, a novel localization method named ***Weight-Switching*** is introduced, which allows the robot to navigate across different pipe segments. To do so, first, a threshold value is defined in the existing algorithm to keep a check on the scan matching score. Depending on the score, the motion model and observation model are used to update the robot's state. Therefore, every time it compares the average scan matching score to the threshold value. If the score is greater than the threshold, more weight is given to the measurement model as it signifies that it has enough features to rely on the LiDAR data for state estimation; otherwise, it uses the motion model for state estimation, which can be demonstrated in figure 3.7. To further optimize and improve the performance of the Weight-Switching technique, a weighting factor α is added to the scan data.

$$Matching\ Score = \alpha \cdot Matching\ Score \tag{3.1}$$

The value of α can range from 0.1 to 1, where 0.1 denoting a straight pipe segment, 1 denoting any intersecting junctions and 0.5 for any small bends. Other small diameter changes are not considered in this thesis and are out of scope for this project. Using this α value, the algorithm can be easily implemented based on the pipe configuration. The overall algorithm is illustrated using a flowchart as shown in figure 3.8.

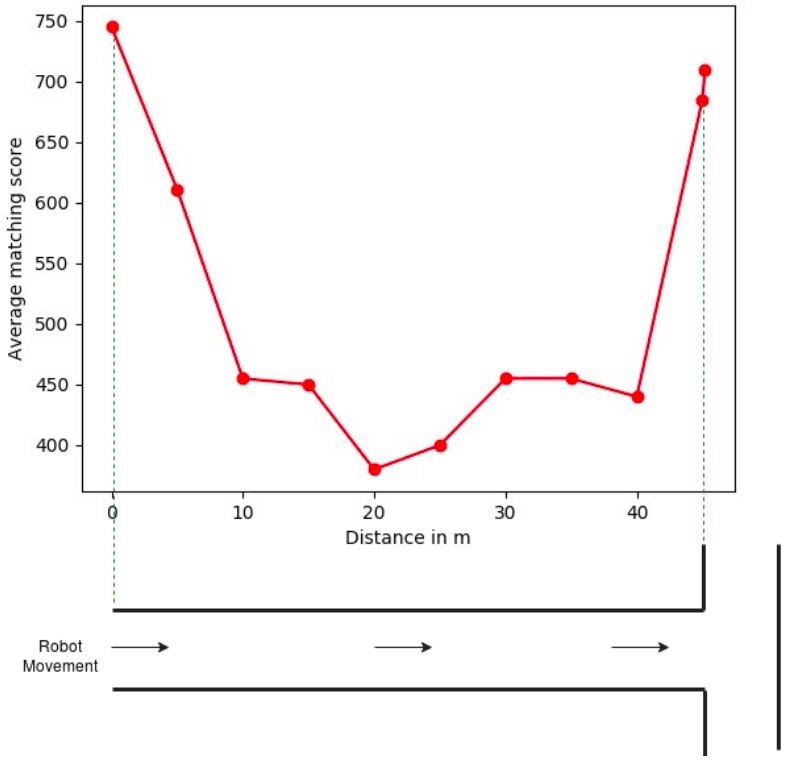


Figure 3.6: Illustrates how the average matching score varies across different sections of the pipe. The top figure shows the plot of the score generated by the scan matcher while moving in a T-junction pipe represented by the bottom figure.

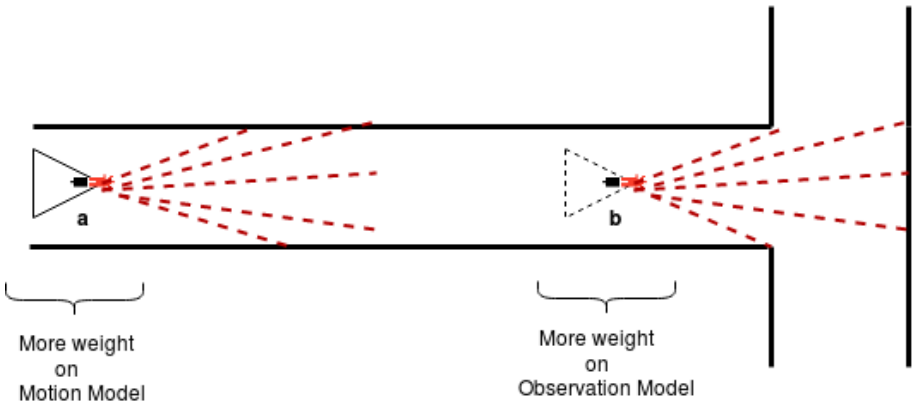


Figure 3.7: Demonstrates how the observation and motion model can be used inside the pipe network effectively. At the point 'a', the robot just uses the motion model due to lack of sparse features, but when the robot reaches point 'b', it uses the observation model to update the state once it discovers a junction inside the pipe.

A pseudocode is written that takes α into account and can be used to improve the overall Gmapping performance for a straight pipe with a junction as shown in Algorithm 1.

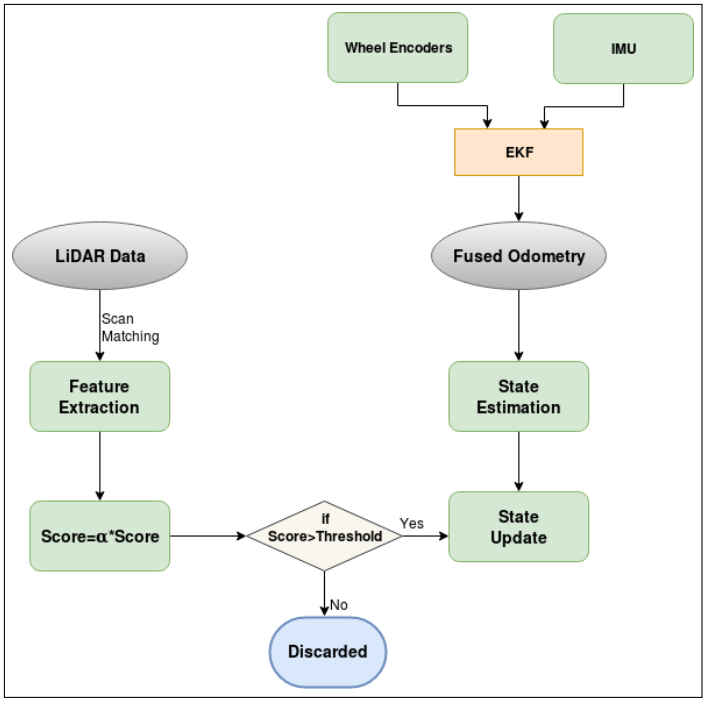


Figure 3.8: A flowchart representation of Weight Switching algorithm

Considering the particle filter-based Gmapping, the main novelty introduced in this section was a pipe segment-based ***Weight-Switching*** approach for the scan measurement which can improve the performance of Gmapping in creating a 2D Occupancy grid map for different pipe segments.

Algorithm 1 Gmapping SLAM algorithm performance improvement for a straight pipe with a junction

Require: : **(Initialized Once)**
T_{score}, threshold value
α, weighing junction parameter
Require: : **(Every cycle)**
z_t, most recent laser scan
u_t, most recent odometry measurement (x, y, θ)
score, Matching score based on laser scan
loop:
M_t = integrateScan(M_{t-1}, x_t, z_t)
if junction detected **then**
$\alpha = 1$
else
$\alpha = 0.1$
end if
$score = \alpha * score$
if score>T_{score} **then**
$x_t \sim$ p(x_t| x_{t-1}, z_t, u_t)
else
$x_t \sim$ p(x_t| x_{t-1}, u_t)
end if
close

3.5 Graph-based SLAM for in-pipe environment

As discussed in section 2.7, Graph-based SLAM uses a graph in which the nodes represent the robot poses or landmarks, and the edges connecting the two nodes reflect the sensor measurement that constrains the related poses. As pipelines form an extensive network, inspection robots will operate in these environments over a long period. The advantage of using a graph-based approach is that it scales well and is robust (loop closure) over long drives, thus making it suitable for pipeline environments. Also, it only uses pose nodes for graph size reduction, thus reduces the computation and memory requirement by a considerable margin. Google Cartographer is one such algorithm that uses a graph-based approach to build a 2D-3D map and localize within that map. Like Gmapping, it also combines wheel odometry data and LiDAR data to produce a map. The main reason behind using Cartographer apart from Gmapping is because of its flexibility and optimization techniques for solving the 2D-3D SLAM problem. Moreover, it provides out-of-the-box parameters which help in mapping and localization efficiently. Nonetheless, there are issues related to memory consumption when creating a 3D map of an environment.
Thus, in this section, we analyze how such an algorithm can be adapted for a pipeline network so that it can produce large-scale maps both in 2D and 3D for navigation of inspection robots. Moreover, we will investigate how 3D maps can be produced efficiently using Cartographer for large-scale pipe network by proposing a solution to the fourth research question, i.e.,
How can Google Cartographer be exploited effectively for building a 2D-3D map inside the pipe network?

3.5.1 Design of constraint-based weight-switching Cartographer algorithm for in-pipe environment

A similar approach of Weight-Switching in Gmapping is adopted for Cartographer as well. However, instead of using a junction-based weight switching, constraints are used for giving weights to the odometry and the LiDAR scan data across different pipe configurations. As discussed in chapter 2, the Local SLAM contains submaps, which are linked by constraints in the graph while in Global SLAM, loop closure is detected. The Cartographer basically combines all these constraints and finds the most likely configuration of the robot trajectory along with the mapping process. These constraints are translational and rotational based on the robot motion. Relative motion constraints are created by the use of proprioceptive sensors, while the measurement constraints are established by the use of exteroceptive information.
In figure 3.9, it shows how the graph can be constructed for a robot moving inside a pipe. When the robot moves in a straight pipe segment, the relative motion constraint will only have a translational constraint, and at the junction, will have a combination of translational and rotational constraints. The measurement constraints will always be there once the robot starts moving inside the pipe. Based on these two constraints, the submaps are created at an instance. So, whenever a submap is created using relative motion constraint having both translational and rotational constraint, suitable weights can be added to the measurement data since the data from the LiDAR will be quite accurate near the junctions or bends. Using an in-built search window in Cartographer, it is possible to identify whether the submap includes only a translational constraint or a combination of translational and rotational constraints. Therefore, based on the relative motion constraint, suitable weight can be applied to the Ceres Scan Matcher, i.e.,

- If the robot is moving in a straight pipe, then the submap containing the relative motion constraint will only have a translational constraint. In this case, translational and rotational weight for Ceres Scan matcher can have a lower value (0.1 to 0.2) for constructing a pose graph.
- If the robot is near a pipe junction, then the submap containing the relative motion constraint will have both translational and rotational constraints. In this case, translational and rotational weight for Ceres Scan matcher can have a higher value (0.9 to 1) for constructing a pose graph.

The values for the scan matcher are chosen based on how much LiDAR information can be trusted. Based on the hypothesis of the first research question, LiDAR is effective only near the junction or bends inside the pipe. For this reason, higher translational and rotational values are chosen when it is near the junction and lower when it is moving in a straight pipe. Therefore, by adding suitable weights to the Ceres Scan matcher based on these constraints, it is possible to improve localization and thus result in better map estimation.
In 3D mapping, the point cloud map usually requires enough memory space, which is a challenge if a large-scale pipe network is to be mapped. To make it memory efficient, an Octomap is integrated to the Cartographer as demonstrated by the flowchart in figure 3.10. Since Octomap is only a mapping procedure, it can utilize the pose graph developed by the Cartographer. Therefore, the proposed constraint-based localization approach can be applied to the Octomap and thus produce efficient 3D mapping.

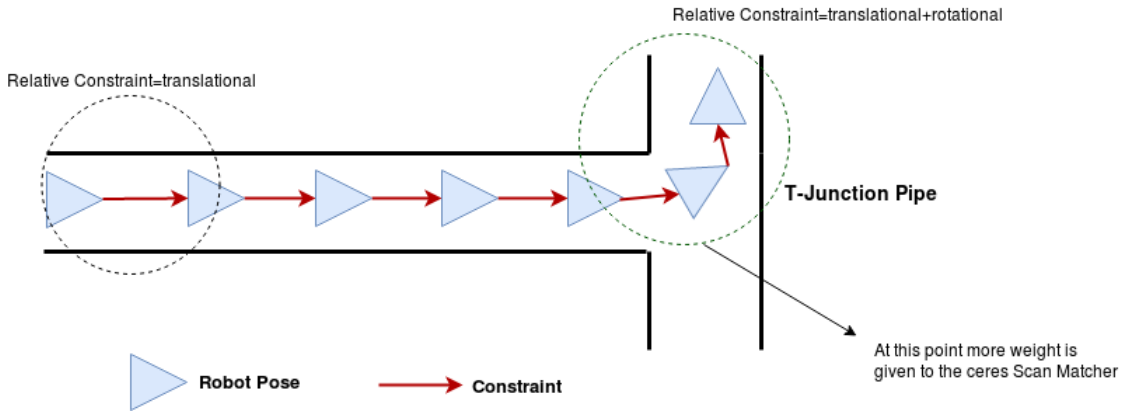


Figure 3.9: Illustrates a graph showing a robot moving inside a pipe. Based on the constraint, a suitable weight can be applied.

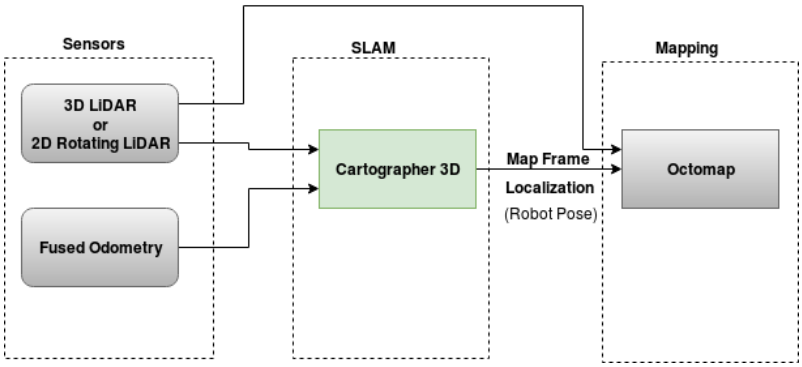


Figure 3.10: Illustrates how an Octomap be integrated to Cartographer

3.6 Methodology

As mentioned in section 1.2, the goal of this assignment is to map (2D and 3D) and localize the robot inside a pipe. In order to do so, the selected SLAM algorithm will be tested in both simulation and real-time.

- In ROS-based **simulation environment**, a 4-wheeled differential drive robot equipped with wheel encoders, IMU and 2D/3D LiDAR, is taken. In principle, if the selected algorithm is able to map and localize the mobile robot inside the pipe, it can be applied to the physical robot (PIRATE) as well. The chosen algorithm will be tested inside a long straight pipe, complex bends, and different intersecting junctions.
- The **physical setup** consists of 2 parts: first, it will be tested in a pipe-like environment on Jackal, a 4-wheeled differential drive robot equipped with a 2D LiDAR. Next, it will be tested on PIRATE in a straight and a T-junction pipe. The LiDAR hardware should be selected based on the PIRATE dimension and weight. The data from the PIRATE can be recorded using a rosbag file and then further can be used for SLAM implementation, as shown in figure 3.11.

Petrochemical pipelines are generally long straight pipes with fewer junctions. These junctions can be treated as a feature and can be used by the LiDAR to correct the pose estimate of the robot, and thereby build the map of its environment accurately. The idea is to give more weight to the motion model (odometry data) to estimate the pose when moving in a straight pipe, and once a junction is detected, switch the weight more towards the measurement model (LiDAR

data). The odometry data is prone to errors at the turnings (heading error), even after sensor fusion, as chances of wheel slippage is more and, hence, likely to estimate the pose incorrectly. In order to avoid such scenarios, the LiDAR data can be used to correct the pose of the robot since the LiDAR works quite accurately once it sees a feature in the environment. Thus, in order to map and localize inside the pipe, a weight-switching strategy is adopted for the chosen SLAM algorithm.

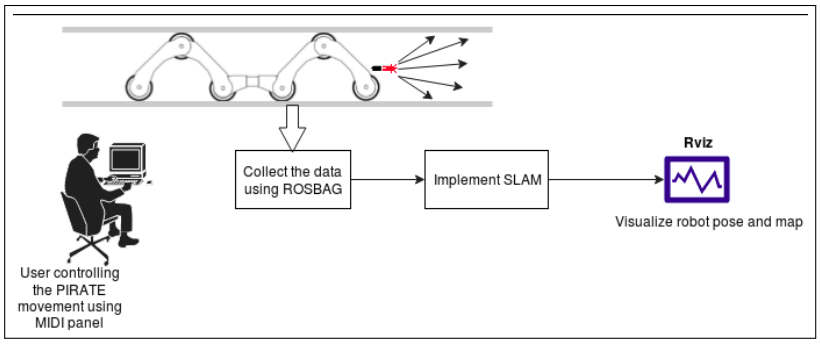


Figure 3.11: Workflow of SLAM implementation for PIRATE

The two environments, as discussed above, are taken as a reference to test the SLAM algorithm. While implementing it on the PIRATE, the odometry data should be carefully handled since the PIRATE clamps and unclamps its bending module when moving through a junction. Overall, the performance of the algorithm can be evaluated based on the following elements:

- **Mapping efficiency**: The result of SLAM is a map made by the mobile robot, which is an abstraction of the real world. The map efficiency can be evaluated based on its quality and inaccuracies compared to the ground truth map. In this case the pipe length from the 2D-3D map can be used for comparison.
- **Localization error**: While constructing the map of an environment, there may be an error in the robot's pose. The deviation of the estimated robot position should be compared to the actual position.
- **Loop Closure**: It is a crucial factor for determining the robustness of any SLAM algorithm. Hence, it should be tested whether the proposed algorithm can recognize the re-visited areas or not.
- **Computational power**: The various SLAM algorithm should also be compared based on the CPU utilization for execution.

3.7 Summary

This chapter presents a brief analysis of the research question as defined in 1.2. Moreover, it focuses on the domain analysis needed for the design and implementation of the SLAM algorithm. A 2D LiDAR is chosen for perception along with fused odometry (wheel encoders and IMU). In Gmapping, a junction-based weight-switching design is proposed for mapping the pipe environment. A similar approach is adopted for Google Cartographer, but instead of junction-based, constraints are used to distribute the weights. Finally, it highlights the methodology to be used to evaluate the proposed algorithm. The two environments (simulation and physical setup) are chosen as a reference to test the respective algorithms. Experiments will be performed on each of the algorithms by varying sensor configuration and other conditions, as discussed in the next chapter.

4 Experimental Design

This chapter presents a series of experiments that will be carried out:

- To check the effectiveness of vision and range-based sensors inside pipe network
- To validate the proposed SLAM approaches inside different pipe segments

The chapter starts with the introduction of an experimental setup where it describes the computational hardware and simulation platform used in this project. As mentioned in the methodology in section 3.6, two setups are used, i.e., simulation and real-time setup. Thus, this chapter then presents the design of simulated experiments followed by real-time experiments. Every experiment has different test scenarios for both simulated and real-time setup. The focus is on the implementation of proposed SLAM strategies inside various pipe segments by addressing the research questions briefly. To summarize, the goal of this chapter is

"To design a series of experiments in order to answer the proposed research questions."

4.1 Experimental Setup

To test the various SLAM methods, a set of experiments to be carried out must be designed and prepared. A prior experimental setup is required. For the experiments conducted in simulation and real-time, the processing power used: a Dell Inspiron 7000 laptop was used with an Intel 2.7GHz Core i7-7th Gen, 8GB of working memory, and an Nvidia GeForce GT940MX GPU. The laptop runs Ubuntu 16.04 and ROS Kinetic. For the simulation, a ROS-based Gazebo simulator is used to represent the 3D environment. Gazebo offers a differential drive plugin and various other sensor plugins that can be used in ROS. The differential drive plugin accepts velocity commands and publishes odometry information. Hence, for the experiments in simulation, a 4-wheeled differential drive robot is taken, equipped with wheel encoders, IMU and Hokuyo 2D, or Velodyne-16 3D LiDAR. In addition to these sensors, a dynamixel motor is also added later for rotating a 2D Hokuyo LiDAR. The specific LiDARs were selected based on the compatibility of Gazebo. However, it should be noted that a 3D LiDAR requires a protoc (protocol buffer) version higher than 3.0. As for the pipe models, they were first built in Blender and later exported to Gazebo via .stl or .obj file format. Initially, it was observed that on exporting the pipe model to Gazebo, there was an issue with the transparency of the pipe model. This was later resolved by increasing the thickness of the model in Blender and re-exporting it to Gazebo.

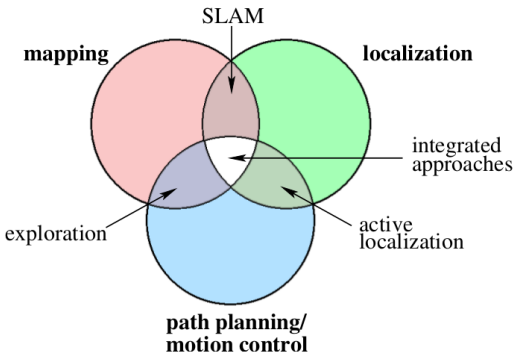


Figure 4.1: Task that need to be performed by a fully autonomous robot [5].

To be a truly autonomous robot, one has to perform multiple tasks, which include mapping, localization, and path-planning, as shown in figure 4.1. As mentioned previously, SLAM is

a technique that only creates a map of an unknown area while also determining the robot's pose in that map. Path planning is defined after the SLAM process. A robot can plan a path autonomously only if the map of the environment is known. Therefore, all the experiments are conducted passively, i.e., teleoperating a robot using a keyboard. This is often classified as **Passive SLAM**. A SLAM framework requires proprioceptive and exteroceptive sensor data and all the transformations data that connect these sensors to the robot frame. All the data captured through a rosbag file are used for implementing different SLAM algorithms. Using the same bag file helps in developing a comparative analysis between different SLAM algorithms.

Table 4.1: Simulation configuration

Robot model	4-wheel differential drive
Robot dimension	790 × 610 × 390 mm
Robot max speed	linear:0.2m/s, angular:0.25m/s
Wheel friction	0.9
Pipe Diameter	1.2m
Environments	Straight pipe, T-junction pipe, closed-loop pipe
real-time factor	1

Since the robot is operated manually, the pipe diameter in Gazebo is chosen as 1.2m for simple robot maneuvers; otherwise, there is a risk of colliding to the pipe walls. All the necessary parameters are defined in table 4.1. There are some standard parameters commonly used in Gazebo, such as the wheel friction for differential drive robot, real-time factor. However, it should be noted that the real-time factor, as mentioned in the table, indicates the computation of Gazebo. If this factor goes below 0.6, then it means it is not able to get enough processing power as required. In such cases, it fails to run the algorithm correctly, and thus, the quality of the map degrades. Hence, this factor is monitored while performing experiments in Gazebo.
For obtaining the ground truth pose of the robot, Gazebo offers a ROS topic containing the pose for each of its models. A ROS-based service is used to retrieve the pose of the robot from the Gazebo topic. However, to generate a 2D Occupancy ground truth map of Gazebo, a higher version of a Gazebo is required, which is compatible with higher Linux version. Nevertheless, there are some open-source packages that help to build a 2D Occupancy Grid Map. However, since the pipe is a cylindrical enclosed model in Gazebo, it is unable to detect the free space inside the model in which the robot moves. As a result of this, the generated occupancy grid map will have different cells (free, occupied, unknown) and resolution, thus cannot be used for validation.

4.2 Simulation Experiments

Different pipe segments are taken to validate the hypothesis developed after each research question. In order to answer the research questions, four experiments are conducted on the virtual simulation environment.

4.2.1 Experiment 1

AIM:- *To check the effectiveness of visual and LiDAR-based system inside different pipe segments*

The first experiment is performed to gain a better understanding of how the camera and the 2D LiDAR works in a textureless, uniform pipe network with minimal features. Measurement information from these sensors is essential for building maps and providing absolute robot positioning. It is, therefore, vital to pick the most effective sensor before constructing the SLAM framework. As mentioned earlier, the camera and the LiDAR works quite well in an environment with abundant features, but it is interesting to see their sensing capabilities in an envi-

ronment with sparse features. This experiment is divided into two parts to get a better intuition about the performance of a vision and range-based system.

- When a robot equipped with either a LiDAR or a camera navigates within a textureless uniform straight pipe network, it is necessary to know whether or not a robot can localize within this environment by using either of the two sensor data.
- Likewise, it is crucial to know how well either of the two sensors can detect junctions or bends inside the pipe so that it can be used as a feature to help the robot localize.

Therefore, this experiment will be conducted inside two different pipe segment, i.e., a long straight textureless pipe and a pipe with a T-junction as shown in figure 5.1, where the main purpose of the experiment is to see how well the robot can localize inside these two environments using either of the two sensors. For localization of the robot within these environments, visual odometry from the monocular camera and LiDAR odometry from the 2D LiDAR are used.

Validation:- To validate the results obtained from both visual and LiDAR odometry, both will be compared with the ground truth position of the robot. Therefore, using the trajectory obtained by individual odometry, the accuracy of the LiDAR and the camera are qualitatively analyzed. Higher localization error will result in poor map construction and thus poor sensor efficiency. The selected sensor from this experiment will be further used for constructing the SLAM framework. The results of this experiment are presented and discussed in the subsection 5.1.1. which answers the first research question as described in section 1.2.

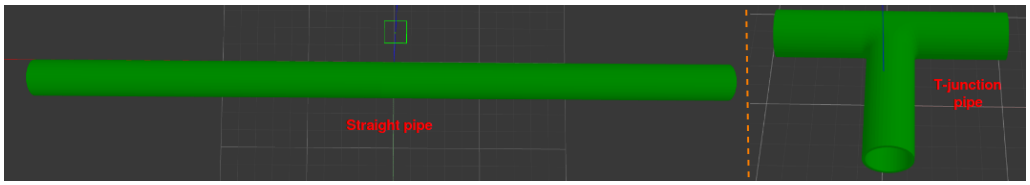


Figure 4.2: Two test setup used for validation of the visual and LiDAR-based system.

4.2.2 Experiment 2

AIM:- *To evaluate how sensor fusion affects localization inside pipe network*

The main aim of this experiment is to test how the sensor fusion aid in localizing the robot within a pipe network. As discussed in the section 2.3 that the odometry data from wheel encoders is not reliable, so combining the data with IMU will improve the overall odometry performance. Mobile robots are usually equipped with many sensors, particularly in a complex environment. Because of sensor noise, it is always a better choice to collect the data from multiple sensors and employ sensor fusion to increase the information content of the robot's input. In this case, relying solely on wheel encoders can result in a poor state estimation due to systemic or non-systematic error. Most importantly, due to the slippery pipe surface, error in odometry can incur, especially near the junctions, when the robot has to turn. In such instances, fusing with IMU can provide accurate orientation information and, therefore, can be used for localization. In order to test the effectiveness of sensor fusion in localization, this experiment is divided into two parts:

- Based on the hypothesis of the first research question, wheel odometry data is required for state estimation. Therefore, in the first part of the experiment, it is necessary to know whether or not the fused odometry data (wheel encoder combined with IMU) can be trusted before applying for the localization technique.

- In the next part of the experiment, AMCL can be used for localization if the odometry information is accurate. Nevertheless, it is crucial to know whether or not applying the fused data with AMCL improves the localization within the pipe network.

For the first part of the experiment, a standard UMBmark test [46] is performed to measure the accuracy of fused odometry data. In UMBmark test, a robot is made to run clockwise and anti-clockwise direction along a square path (5m × 5m) multiple times. The drift in the odometry data can be easily visualized through this test. In the second part, we assume that two sets of maps of the pipe environment are provided to AMCL to focus only on the localization. One set contains a map of straight pipe and a complex pipe made from Gmapping without sensor fusion, and the other set is made using sensor fusion. As we accomplished from the analysis of the first research question that a LiDAR will fail terribly in localization when the robot is moved in a straight textureless pipe. From the analysis of the second research question, it was found out that wheel odometry data is susceptible to slippage, especially at the turnings, making the odometry data inaccurate. Therefore, after applying the fused odometry data in AMCL, it will be first tested in a long straight textureless pipe map, and then to test the reliability of the fused odometry data, it will be tested in a complex pipe map with multiple junctions as shown in figure 4.3.

Validation:- To validate the results obtained from the first part of the experiment, standard deviation of the error will be plotted for odometry with fusion and without fusion in order to see the improvement in the fused odometry data. For the second part, the validation will be done qualitatively since it is required to observe whether the robot is able to localize within the pipe accurately or not. This can be visualized through the particles point cloud in Rviz. So during pipe inspection in order to identify the defect location, robot position should be known within the map. The result of this experiment answers the second research question, and it is discussed in subsection 5.1.2.

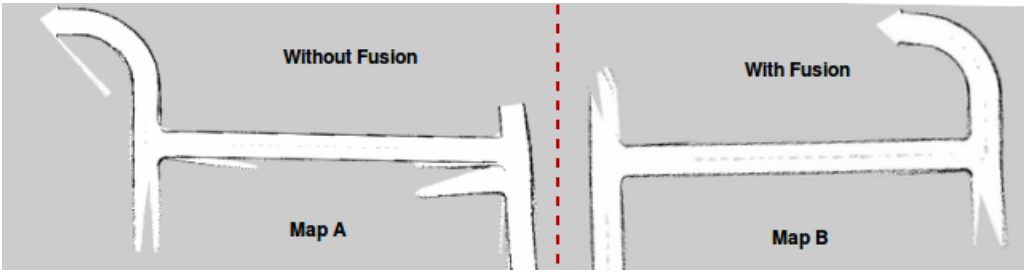


Figure 4.3: Pre-built map used for AMCL implementation. Map generated using Gmapping. Map A is created without sensor fusion and Map B is created using sensor fusion.

4.2.3 Experiment 3

AIM:- *To evaluate the performance of Weight-Switching Gmapping in different pipe segments*

The main objective of this experiment is to test how well the proposed Weight-Switching Gmapping algorithm can build a 2D map and localize within that map for different pipe segments. Gmapping takes a 2D LiDAR scan measurement and odometry data to construct the map of the environment and localize it within that map. From the analysis of the second research question, it is inferred that implementing sensor fusion will boost the localization technique and thus build an accurate map. Therefore, the fused data is used with 2D LiDAR in this case. This experiment is conducted to get a better intuition of how the junction-based weighting parameter α helps to improve the localization of Gmapping in different pipe segments. As discussed in the previous chapter, the two most challenging pipe environments are a long straight textureless pipe and a long pipe with sparse features. Therefore, Gmapping is tested in two environments. The first environment is taken as a long straight pipe to test whether the algorithm

can resolve the narrow-hallway problem or not. The second environment is chosen as a long straight pipe with a T-junction as shown in figure 4.4. This is done to check whether the LiDAR can correct the pose of the robot or not once it sees a junction. As proposed in section 3.4, it should use the odometry data from the motion model when it is moving in a long straight pipe and use LiDAR data from the measurement model once it is near the junction or bend. In this way, it is expected to produce a better map compare to the existing Gmapping algorithm.

Validation:- To test the proposed SLAM framework, it is compared to the existing Gmapping algorithm. The 2D Occupancy grid map of the pipe segment obtained from the two Gmapping algorithms will be used to compare the two algorithms. In this case, the results obtained are evaluated quantitatively. Each algorithm is made to run five times inside the pipe segment since, due to the sensor noise, the estimated map differs each time. Therefore, the average length of the pipe is measured from the built map for the five trials. The result of this experiment answers the third research question and is discussed in section subsection 5.1.3.

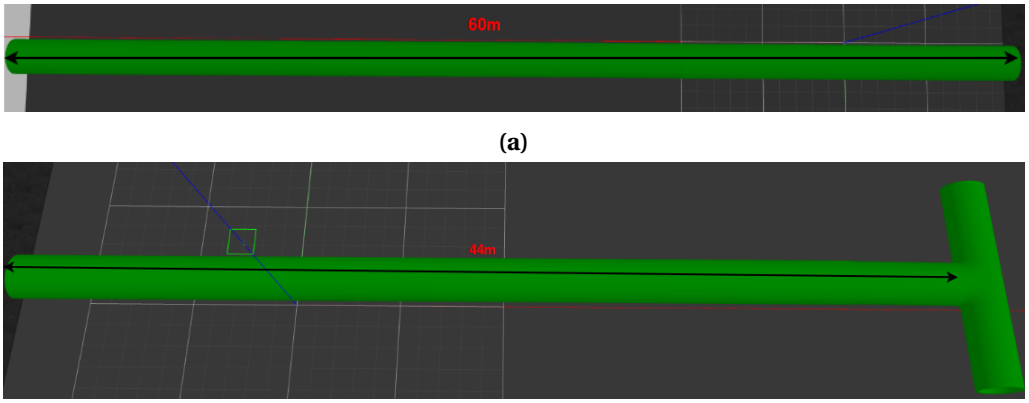


Figure 4.4: Different pipe segments used for testing. Figure (a) shows a 60m long straight pipe while figure (b) shows a T-junction pipe model.

List of Parameters:- The various other important parameters used for the implementation of Weight-Switching Gmapping for the in-pipe environment are highlighted in table 4.2. The values for these parameters are mostly chosen after running multiple experiments and analyzing the behavior of the Gmapping framework inside the pipe. The computational load depends on mainly these parameters. A lower map_update_interval data updates the occupancy grid more often, but it comes at the expense of higher computational load. Selecting a number of particles is not intuitively clear since a lower number might lead to an inaccurate state estimation while a higher number might introduce memory consumption and CPU load. The parameters linearUpdate and angularUpdate implies how much a robot has to translate or rotate in order for the latest scan data to be processed [47]. Setting a large value for these two parameters will reduce the computation but would result in poor map quality. There is a trade-off between accuracy and computation in selecting these parameters. Hence, for a better map quality of the pipe network, a low value was chosen for both.

Table 4.2: Parameters used for implementing Weight-Switching Gmapping inside pipe network

Parameter name	**Description**	**Value**
map_update_interval	Time interval for map update	3.0
particles	No of particles used	100
linearUpdate	Scanning process time when robot translates this far	0.2
angularUpdate	Scanning process time when robot rotates this far	0.136

4.2.4 Experiment 4

AIM:- *To evaluate the performance of constraint-based Weight-Switching Cartographer in large-scale pipe segments*

Just like Gmapping, this experiment is done to check the performance of the Weight-Switching in large-scale pipe segments. The main objectives of this experiment are

- Build 2D-3D map of the complex pipe network, i.e., check the scalability of the algorithm
- Establish Loop Closure
- Build efficient 3D map by integrating Octomap

As input to the Cartographer, fused odometry measurements are combined with 2D or 3D LiDAR data. One of the biggest challenges of using a cartographer for any application is the integration of the algorithm. This experiment is, therefore, conducted to check whether it is possible to successfully create a 2D-3D map of the pipe environment using the proposed algorithm. As discussed in 3.5, the constraint-based switching approach updates the LiDAR weight when it is near to the junction so that the LiDAR data can be used for pose correction. When moving in a long straight pipe, it gives more weight to the odometry from the motion model and less weight to the LiDAR measurement to build the submaps. Since most of the pipelines have an extensive network, these submaps can be beneficial in mapping such environment.
Petrochemical pipes have mostly straight networks with some junctions and bends and likely to have any closed-loop network. However, if there is a closed-loop, it can be used as a loop closure constraint to correct the robot pose. Therefore, it is always an advantage to have a loop closure in the SLAM algorithm. In the case of 3D mapping, point cloud maps typically consume quite a bit of memory space when a large map is built. In such cases, having an efficient 3D mapping of the physical world would be handy; hence, the integration of Octomap with the Cartographer is needed. From this experiment, it will be crucial to know how effectively Octomap can be built on top of the Cartographer to represent the physical world. Cartographer will be tested in several pipe segments as shown in figure 4.5 and 4.6.

- It is first evaluated in a long straight pipe and a straight pipe with a T-junction as shown in figure 4.4, so that it can be used later to conduct a comparative analysis with Gmapping algorithm.
- The scalability and loop closure constraint is then evaluated in a complex pipe both in 2D and 3D.

Validation:- To test the performance of constraint-based weight switching, the length of the pipe from the estimated map will be calculated. For scalability and loop-closure, it will be analyzed qualitatively from the generated 2D-3D map. At last, memory-usage of a large-scale point cloud map will be compared to the Octomap, to examine the memory efficiency of Octomap. The result of this experiment answers the fourth research question and is discussed in subsection 5.1.4.

List of Parameters:- Most parameters that need to be tuned belong to the Local SLAM and had to be chosen carefully since each parameter can affect the mapping quality. It works for both 2D and 3D.

- *submapGenerate*: It is the time interval taken to publish submap poses. A small-time interval 0.3 is selected so that it does not miss any portion of the robot pose.
- *POSE_GRAPH.optimize*: There might be some slippage due to wheel odometry while forming submaps. Setting this parameter to 0 would disable Global SLAM interference in

tuning local SLAM; otherwise, it would keep on trying to optimize every node present in the Local SLAM, thus increasing computation as well.

- *TRAJECTORY_Scan.score*: In order to avoid the Ceres Scan Matcher deviate from the prior match, a high value is set. So, in this case, 600 is selected, which means the scan matcher has to generate a higher score in another position than this to be accepted. At the pipe junction or bends, since the score will be higher, it can be used for correcting the pose of the robot.
- *TRAJECTORY_2D.submaps.num*: It configures the size of the submaps in the Local SLAM. The value should be chosen low so that small drifts do not destroy too much of the map
- *POSE_GRAPH.optimization.odometry*: The translational weight can be set to a higher value depending on wheel slippage. However the uncertainty is generally during rotation so the rotational weight should be kept 0. This trusts the current velocity estimate more, which reduces slipping when the scan matching fails in straight pipe.

Table 4.3: Parameters used in Cartographer for in-pipe environment

Parameter name	**Description**	Value
submapGenerate	Time interval taken to publish submap poses	0.3
POSE_GRAPH.optimize	Optimizing each node by Global SLAM	0
TRAJECTORY_Scan.score	Threshold score for scan matcher	600
TRAJECTORY_2D.submaps.num	Size of the submaps in the Local SLAM	0.5
POSE_GRAPH.optimization.odometry	Odometry error correction	50

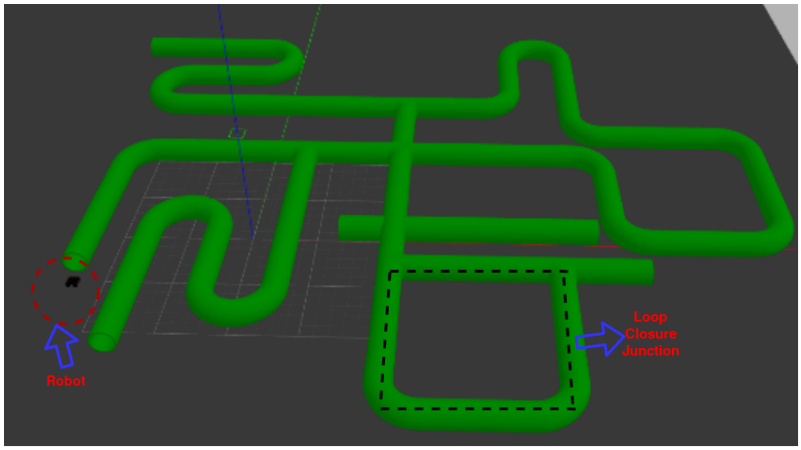


Figure 4.5: Representation of a complex pipe in Gazebo used for 2D mapping

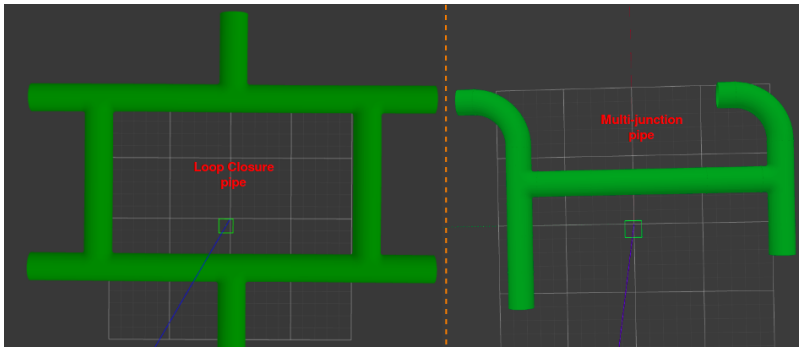


Figure 4.6: Representation of a closed-loop and a multi-junction pipe in Gazebo used for 3D mapping

4.3 Real-time Experiments

For evaluating the algorithms in real-time, it is tested on two physical robots, i.e., on a Clearpath Jackal differential drive robot and PIRATE, which are available at RAM lab at the University of Twente.

4.3.1 Experiment 5

AIM:- *To evaluate the performance of proposed Gmapping and Cartographer algorithm in real-time*

The main purpose of this experiment is to assess the performance of the proposed algorithm in real-time. The experiment will be performed in two parts:

- Testing with the Jackal robot will be conducted in a pipe-like environment since the robot is too big to fit into a pipe.
- Testing with the PIRATE will be conducted in a straight pipe with a diameter of 120mm. However, due to hardware issues, the clamped wheels of PIRATE sometimes looses its contact with the pipe wall while making a turn. Thus, for testing purpose only a straight pipe is used.

Validation:- The estimated pipe length from the obtained 2D map will be used to evaluate the performance of both robots.

Figure 4.7: Real-World Experiment Robots available in RAM lab at University of Twente. Left image is of the Clearpath path differential drive mobile robot (Jackal). Right image shows PIRATE inside 125mm pipe.

5 Results

This chapter will present and discuss the experimental results used to validate the hypothesis established for each research question. Furthermore, based on the selected sensor, an optimum combination of the mapping and localization algorithm for the in-pipe environment is what discussed here. Lastly, a critical assessment that focuses on the strength and weaknesses of the suggested solutions is stated.
The main objectives of the experiment designed in the previous chapter are:

- The first goal was to test the reliability of the sensors inside the pipe network before using it for constructing the SLAM framework.
- The next goal was to evaluate the proposed SLAM approaches based on map quality, localization error, computational load, and loop closure.

Here, the results of all the experiments planned in the previous chapter are discussed. First, it presents the simulation results and then the outcomes of the real-time experiment. Next, it portrays a comparative analysis of the SLAM algorithm based on the results obtained, followed by a critical evaluation. Thus, to summarize the goal of this chapter is

"To describe the experimental results for evaluating the SLAM effectiveness inside pipe environment."

5.1 Simulation Results

Early experiments were conducted during the design phase. Some of the tests as shown in the figure 3.2b, 3.3, 3.6 were used to analyze and develop the proposed SLAM framework. In this section, the results of the four experiments that were designed in the previous chapter to answer the research question are presented. As an oversight of what the experiments will demonstrate is encapsulated here:

1. **Experiment 1:** Among camera or LiDAR, which perform best inside pipe environment?
2. **Experiment 2:** How sensor fusion will help in localization within pipe network?
3. **Experiment 3:** How does Weight-Switching Gmapping algorithm perform inside different pipe segments?
4. **Experiment 4:** How constraint-based Weight-Switching Cartographer algorithm be applied to large-scale pipe network for mapping in 2D and 3D?

5.1.1 Experiment 1 Results

In this experiment, the performance of the LiDAR sensor and the camera are measured. The purpose of this experiment is to test whether the two sensors can be used for mapping and localization inside different pipe segments. From the reported literature, it is known that both the sensor perform exceptionally well in an environment with abundant features or textures. Nevertheless, the pipe may have textures and multiple junctions, but there may be several pipes without textures and fewer junctions. Thus, one can never depend on the environment. The sensor performance is tested in such an environment, as shown in figure 5.1. Odometry information is estimated from both the sensor. First, it is measured in a 40 m long, textureless pipe. Figure 5.1b indicates that both sensors are unable to localize inside the long pipe. From the figure it is seen that, the estimated distance of the robot is only ~1.5m after traveling in a 40m pipe. Visual and LiDAR odometry data were not able to estimate the location of the robot

due to the narrow-hallway problem. In this case, both LiDAR and the camera will perceive the same point at every time before they discover a feature in the area. When a robot is made to run in a T-junction pipe, both visual and LiDAR odometry successfully estimate the robot pose since they can easily extract features from the environment, which allows them to estimate the robot's location correctly. It shows that these sensors are capable of detecting junctions in a pipe and can be used for pose estimation as shown in figure 5.2. However, in the case of a long straight pipe due to lack of salient features, they cannot be used for localization. Hence, wheel odometry data is required in such cases.

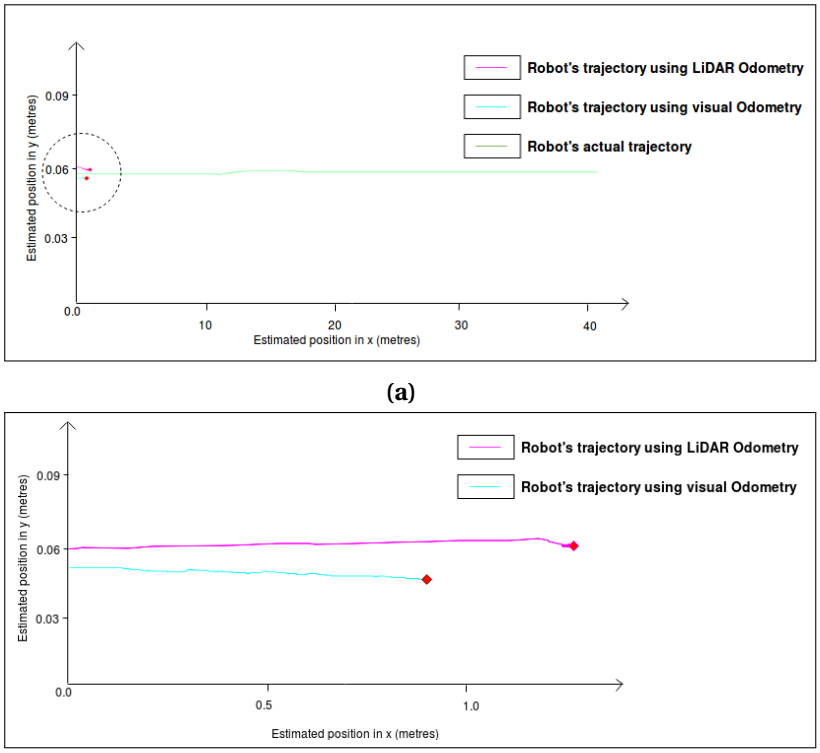


Figure 5.1: Visual and LiDAR odometry in a 40m textureless long pipe. Red dot shows the robot position. The dotted circle in the first figure has been highlighted in the second figure for LiDAR and visual odometry.

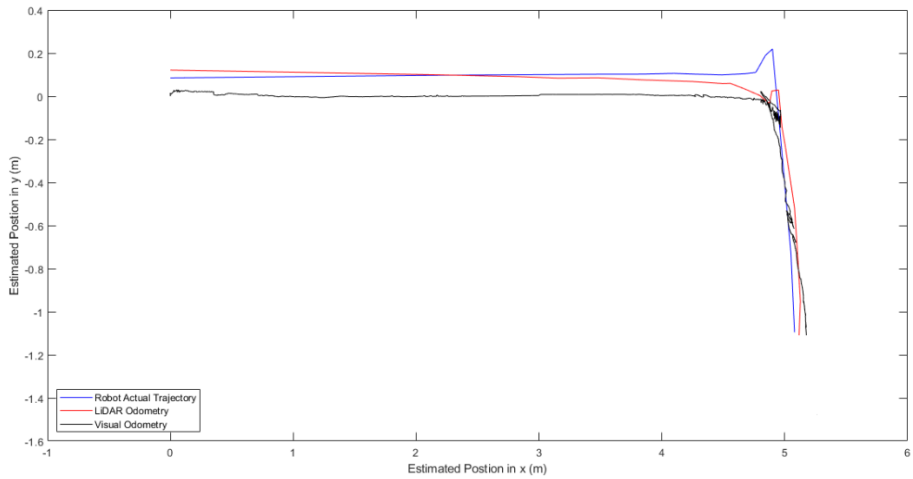


Figure 5.2: Visual and LiDAR odometry in a T-junction pipe

5.1.2 Experiment 2 Results

This experiment aims to evaluate how sensor fusion affects localization inside a pipe environment. It is first conducted to check the reliability of fused sensor data between wheel encoders and IMU and then uses this data for localization. From the results of the previous experiment, it is evident that neither LiDAR nor the camera can be used as a stand-alone sensor for pose estimation, and therefore, the data from proprioceptive sensors are needed for state estimation. However, as discussed in chapter 2, the odometry data is susceptible to error due to slippage or wheel imperfections. In most cases, wheel slippage error is one of the contributing factors among all of them. The pipe surface can be smooth, and there is a risk that the wheel may slip, particularly when turning, which can lead to an orientation error. To minimize this error, sensor fusion is done using EKF to make odometry data more accurate and can, therefore, be used for pose estimations. The University of Michigan Benchmark (UMBmark) experiment is conducted to evaluate the fused data. Figure 5.3 shows one of the scenarios where the robot is made to move counter-clockwise direction in 5×5m square. As shown in the figure, the green line (fused data) is much closer to the blue line (ground truth) than the red line (raw odometry data). Similarly, this test was done 3 times both in clockwise and counter-clockwise direction and the standard deviation plot is shown in figure 5.4. The standard deviation for the raw odometry data is more than that of fused data. This result shows that sensor fusion improves the odometry error and thus can be used for localization.

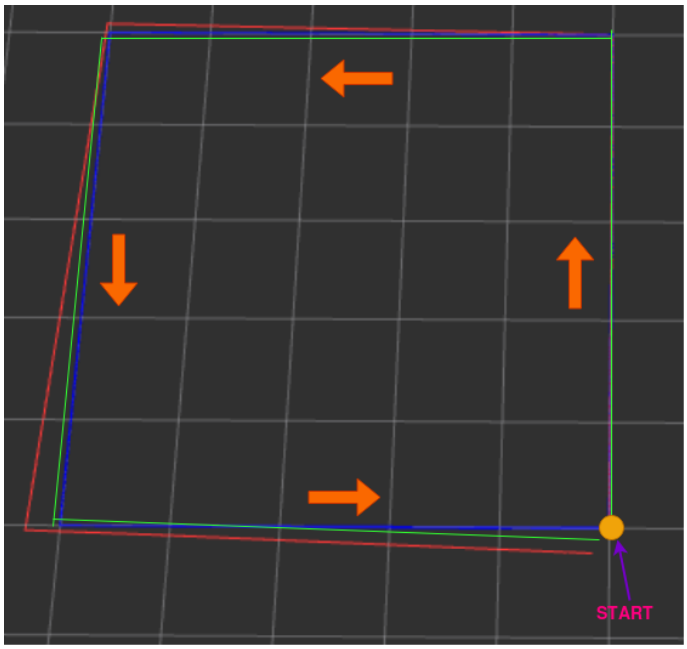


Figure 5.3: Representation of UMBmark benchmark test for calculating dead reckoning error. Blue line is the ground truth, green line is the fused odometry (wheel encoders + IMU) data and red is the raw odometry data.

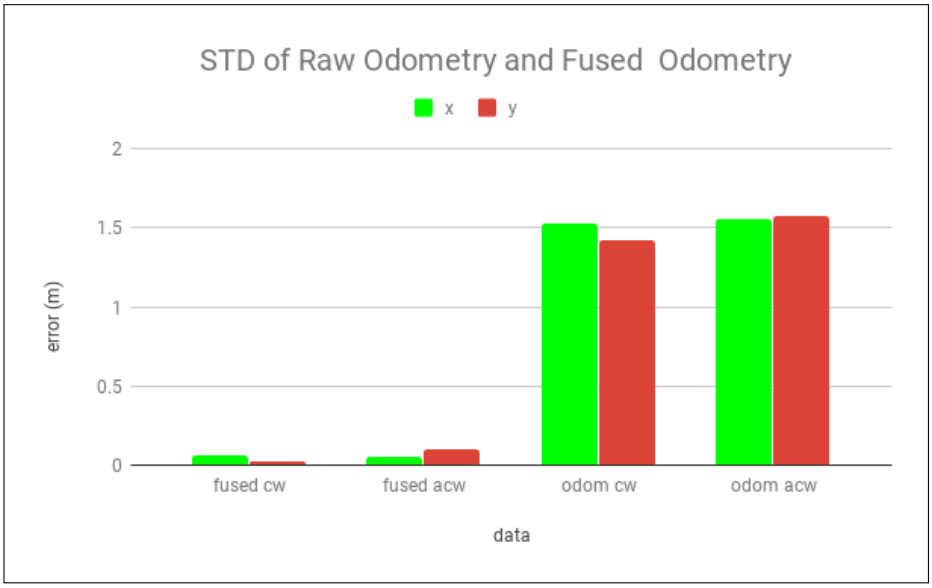
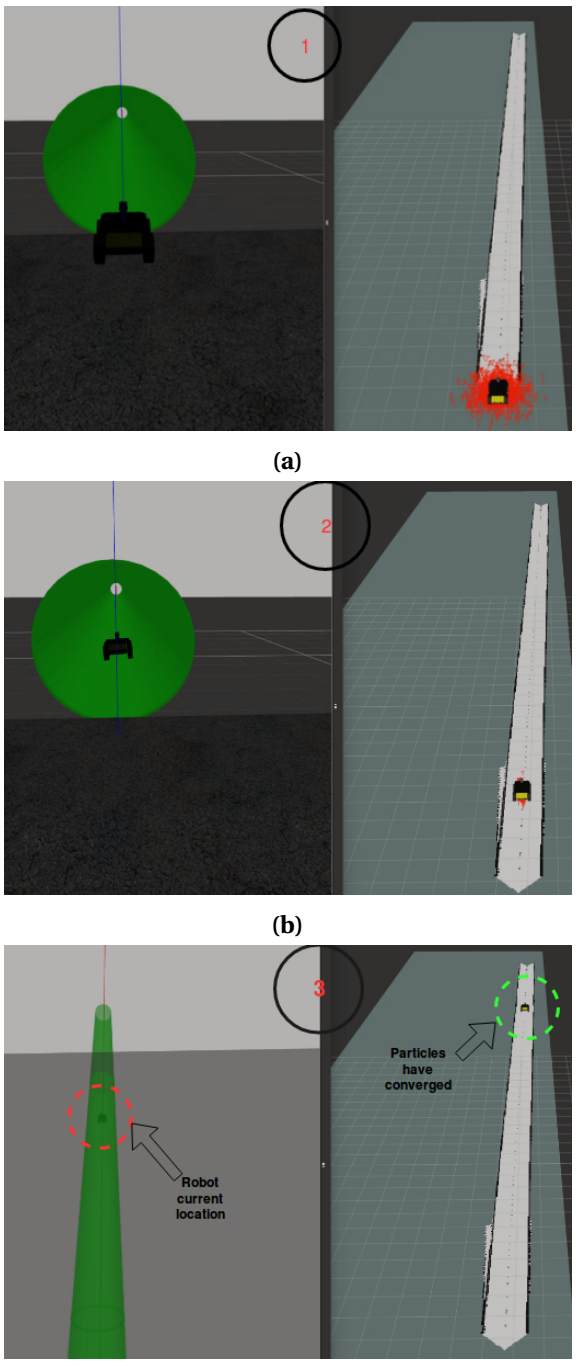


Figure 5.4: Standard deviation of Fused odometry (wheel encoders + IMU) data and Raw Odometry data. Here cw means clockwise and acw is anti-clockwise direction. It is obtained by running both the data in clockwise and counter-clockwise 3 times in 5×5m square.

In the next part, to test how fused data affects localization, AMCL was performed inside two pre-built maps, as shown in figure 4.3. The two maps are built using Gmapping: one with fused data and the other with raw odometry data. It should be noted that only the localization performance is evaluated in this experiment and not the mapping. AMCL is a particle filter based KLD sampling method that uses odometry data and LiDAR data as input for estimating the robot pose. It distributes the particles based on the difference in the odometry and particle location, and then perform scan correlation to match the best-estimated particle that aligns correctly to the given map. AMCL is first performed in a straight pipe, as seen in figure 5.5. Initially, the dispersed point cloud indicates the uncertainty in the robot's location, but as it travels, it gets more measurement information and more confidence in the robot's position. At some point, the point cloud converges, as seen in figure 5.5c, and robot pose, can be accurately known. In this case, the chances of slipping are not much, since the robot accelerates only in a linear direction, so the impact of the sensor fusion is difficult to judge. The next environment is therefore chosen as a multi-junction pipe in which AMCL is performed, as shown in figure 5.6 and 5.7. First, the AMCL is performed in the absence of sensor fusion. It is observed from figure 5.6b that the particle point cloud increases, which is due to the orientation error occurring near the junction. Again, the error in odometry leaps to a high value due to which the particles are distributed to a wide area, and eventually, the localization fails, as shown in figure 5.6c. In figure 5.6d, due to high uncertainty in the pose of the robot, the particle cloud gets distributed in two parts. At this point, AMCL is trying to get the best correlation score, but due to the similar geometric shape of the pipe, AMCL incorrectly estimates the robot pose. This is often in robotics called **False Localization**. However, once the robot is near the bend pipe, the estimates get corrected, and AMCL pulls back the robot to its correct location as shown in figure 5.6e, demonstrating the rapid convergence of AMCL.
When applying AMCL with fused odometry data, it precisely estimates the position of the robot at each junction and thus removes the False Localization issue, as shown in figure 5.7. Moreover, applying sensor fusion to AMCL, it substantially reduces the number of particles and, as a result, also reduces the computational load. The overall result shows how the sensor fusion can have an impact on the robot's localization in any environment.

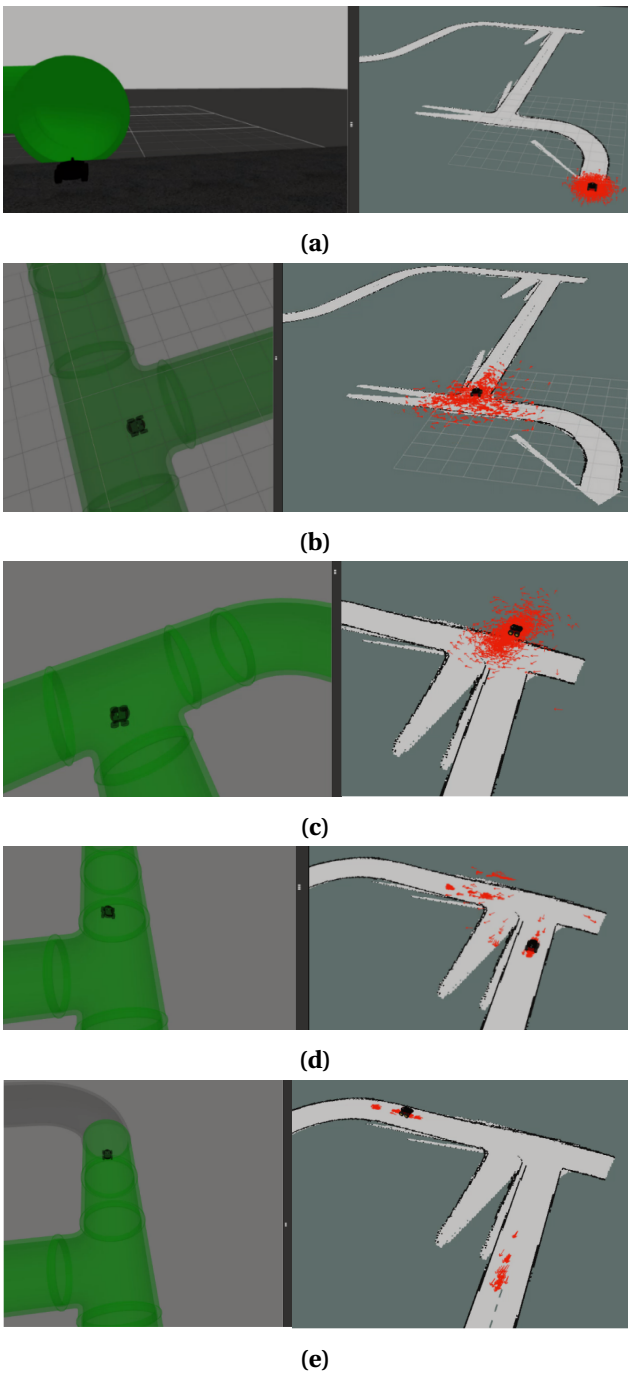


(a)

(b)

(c)

Figure 5.5: Demonstration of AMCL in a straight pipe. Figure (a) represents the dense point cloud around the robot position due to its initial uncertainty in pose. Figure (b) shows once the robot start moving inside the pipe, it gets more measurement and point cloud start converging around the robot. Figure (c) shows at the $\frac{3}{4}$th section of the pipe, particle point cloud have completely converged.



(a)

(b)

(c)

(d)

(e)

Figure 5.6: Demonstration of AMCL across different section of a curved-junction pipe **without sensor fusion**. Map is built using Gmapping without fused data. Figure (a) represents the dense point cloud around the robot position due to its initial uncertainty in pose. However, once the robot start moving inside the pipe, it gets more measurement and point cloud start converging around the robot. Figure (b) shows the point cloud get more dense due to the orientation error in the odometry. Figure (c), it can be seen that due to the error in localization, the estimated location goes out of the pipe structure. Figure (d) shows the false localization. The particles get dispersed and AMCL is uncertain about the current location of the robot, so the particle with higher density will dominate the robot pose at that moment. Hence a jump in the robot pose could be seen. Figure (e) shows after some time once it get more measurement data, it gets more certain about the robot position so again AMCL brings back to its correct position.

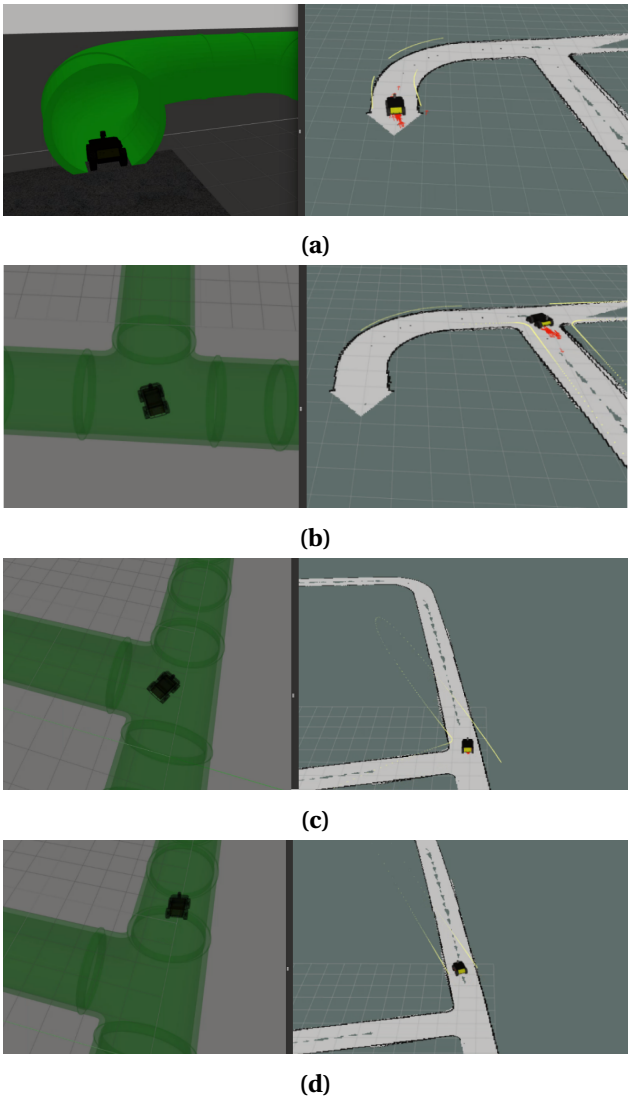


Figure 5.7: Demonstration of AMCL across different section of a curved-junction pipe **with sensor fusion**. Map is built using Gmapping with fused data. Figure (a) represents the small point cloud around the robot position due to its initial uncertainty in pose. Figure (b) shows even at the turning since the odometry data is reliable, the particle point cloud scatter in a very small region around the robot. Figure (c) shows how after sensor fusion, the false localization is eliminated. Figure (d), it can seen that the point cloud has completely converged.

5.1.3 Experiment 3 Results

As mentioned earlier, the robot was first operated manually in the simulated world without any mapping or navigation system operating. The odometry and laser scan data for this drive was recorded for evaluating different SLAM frameworks. In this experiment, the performance of the proposed Weight-Switching Gmapping algorithm is evaluated. The robot was made to run in two separate pipe environments, i.e., a 60 m long, textureless straight pipe and a 44 m long straight pipe with a T-junction at the end. On applying the proposed algorithm, the outcome of both tests is a 2D Occupancy grid map, as shown in figure 5.8 and 5.9. The white pixel is the free space inside the pipe where the robot can move, black pixels represent the pipe walls, and grey pixels are the unknown spaces outside the pipe. However, it should be noted that the width between the two parallel black pixel lines does not represent the diameter of the pipe. The LiDAR is positioned at the top of the robot chassis, so it measures the scan to its horizontal

plane. The laser rays emitted by the LiDAR are below the pipe diameter, and therefore, the length should not be interpreted as the diameter of the pipe.

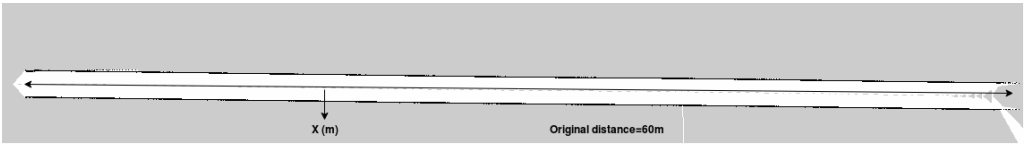


Figure 5.8: A 2D Occupancy grid map for a 60m straight textureless pipe. The length 'x' is estimated based on the outcome of each SLAM algorithm.

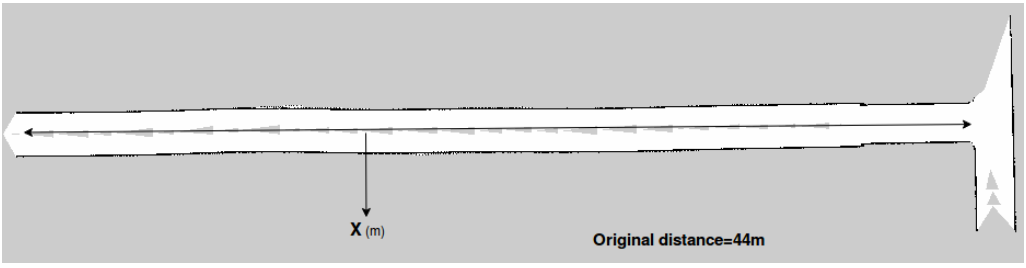


Figure 5.9: A 2D Occupancy grid map for a t-junction pipe. The length 'x' is estimated based on the outcome of each SLAM algorithm.

To benchmark the mapping performance, the Weight-Switching Gmapping algorithm is compared to the baseline Gmapping algorithm. For a quantitative evaluation between the two methods, the length of the straight pipe in both cases was measured. Eventually, the main difference in both the test cases is the estimated pipe length. Due to sensor noise, on applying the SLAM algorithm, the estimated pipe length differs each time. Hence, each test was conducted five times for both the environments, and its estimated average pipe length was calculated. From the table 5.1 and 5.2, it can be seen that the Weight-Switching algorithm performs quite well in both the environment.

Table 5.1: Estimated straight pipe length using different SLAM algorithm after 5 trials

Method	**Estimated average 'x' length(m)**
Gmapping	54.39
Weight-Switching Gmapping	57.79

Table 5.2: Estimated T-junction length using different SLAM algorithm after 5 trials

Method	**Estimated average 'x' length(m)**
Gmapping	41.35
Weight-Switching Gmapping	43.33

The existing Gmapping algorithm uses both odometry data from the motion model and LiDAR data from the measurement model for pose estimation. This remains the same throughout the map building process. In the first environment of a long straight pipe, the LiDAR data is not suitable for pose estimation since it will scan the same point at each instant, as discussed in section 3.2.1. As Gmapping uses both data equally, the error in Localization is more comparatively due to the incorrect LiDAR data, and hence it has an error of 5.61m. In the second environment, the LiDAR data will be useful only when the robot is near the T-junction. In this case, the localization error gets corrected by the LiDAR at the end, and the overall error is reduced. The performance of this Gmapping is improved by adding a threshold value and a junction-based

weighting factor α. Based on the pipe segment, this factor can be regulated during an on-going map building process. When the robot is moving in the first environment, the alpha value is kept low (0.1-0.2), making the scan matching score below the threshold value, thus forcing it to consider only the motion model for pose estimation. While in the second environment, the weighting factor takes a higher value (0.9-1) once the robot is near the junction so that only LiDAR data is used for pose estimation and thus correcting the pose. Because of the better pose estimation, the mapping performance is improved, as seen from the table. Therefore, the weight is given to the motion model when it moves in a straight pipe and to the observation model when the robot is near the junction, making a better position estimation and, as a result, creating a more accurate map of the pipe segment.

5.1.4 Experiment 4 Results

In this experiment, it is aimed to evaluate the effectiveness of proposed Weight-Switching Google Cartographer in the mapping (2D-3D) different pipe segments. Moreover, scalability and loop closure will also be qualitatively analyzed. Finally, the mapping efficiency of Octomap will be examined. This experiment was conducted in the same environment as Gmapping to test the efficiency of Cartographer. The outcome of Cartographer is also a 2D Occupancy grid map which is similar to figure 5.8 and 5.9. The main difference is the estimated pipe length for both the environments. Again each test was conducted five times, and its estimated average length of the pipe was recorded in table 5.3. Unlike Gmapping, Cartographer uses node constraints in a submap for switching. In the case of a straight pipe segment, relative motion constraint only has a translational constraint, so it keeps on building small submaps using these constraints between the nodes and simultaneously optimizing the Local SLAM for maximum pose likelihood, thus estimates the maximum pipe length. While in the case of straight pipe with a junction, the relative motion constraint has both translational and rotational constraints when the robot takes a turn at the junction. This combination can be identified using the search window, and thus suitable weight is applied to the Ceres Scan Matcher. This weight enables the Local SLAM to trust in the LiDAR measurement and thus helps in pose correction of the robot.

Table 5.3: Estimated pipe length in different environment after 5 trials

Cartographer 2D	
Environment	**Estimated average 'x' length(m)**
Straight pipe	59.05
Straight pipe with a T-junction	43.61

Next, using the same concept, it is tested in a more complex pipe, as shown in figure 4.5. A 2D Occupancy grid map is visualized along with the robot trajectory in figure 5.10. From the figure, it can be inferred that the proposed algorithm is scalable to complex pipe networks and is capable of detecting the loop closure path.
For 3D mapping, the Cartographer is first tested in a closed-loop pipe for loop closure detection and then in a multi-junction pipe network, as shown in figure 4.6. Firstly, the Cartographer is made to run in the closed-loop pipe network with a 3D LiDAR. The outcome of the algorithm is a point cloud map, as shown in figure 5.11. The purple line represents the robot trajectory. From the figure, it can be seen that the loop is closed when the robot re-visits the pipe segment, thus demonstrating the loop closure detection. While building large-scale maps, one of the major challenges is memory usage. Generally, the point cloud map requires a high demand for memory storage, which is problematic if the environment is large. For this reason, Octomap is integrated to the Cartographer. Since Octomap is only a mapping procedure, it uses Cartographer pose graph data for localization and builds map efficiently, as seen in figure 5.12. Figure 5.13 shows a significant reduction in memory consumption for an octree-based map as com-

pared to point cloud map data (closed-pipe network). Thus, Octomap can be incorporated into any 3D SLAM algorithm and used efficiently to map large areas.

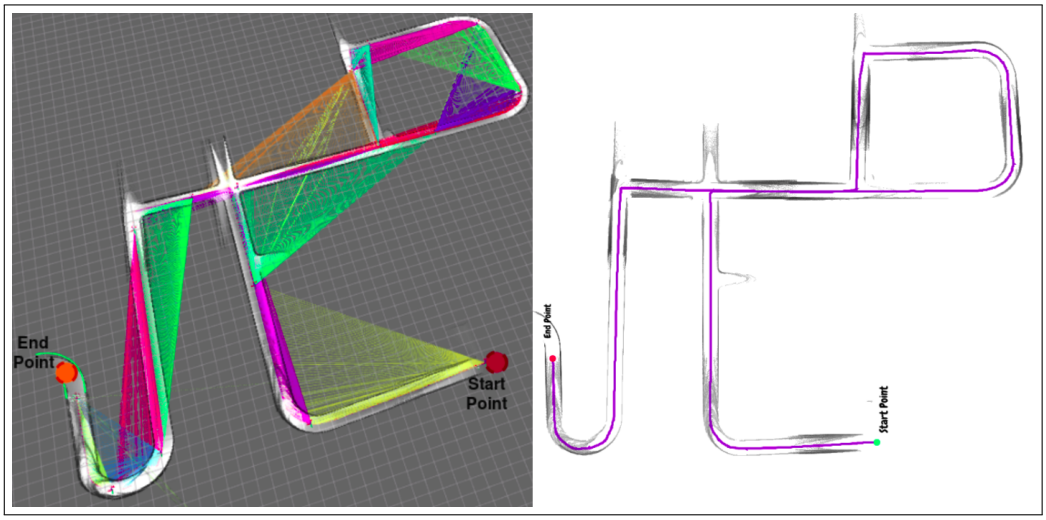


Figure 5.10: The robot was made to run across different pipe segment of the complex pipe. [Refer fig 4.5]. The left image shows the 2D Occupancy grid map obtained from Cartographer, and the right image shows its trajectory. The colored lines in the left image represent the constraints between each submaps. The different color is given to each set of submaps once the optimization is done by the Global SLAM.

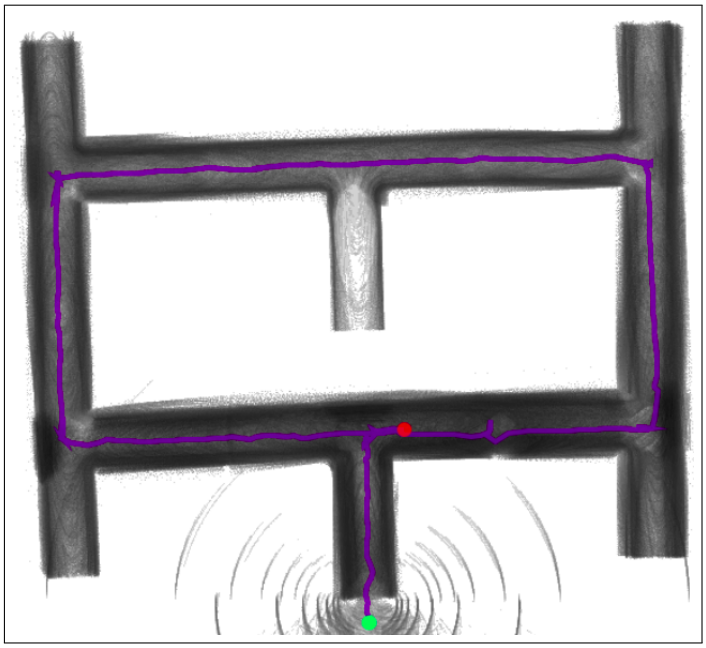


Figure 5.11: Demonstration of a loop closure using Cartographer 3D for a closed-loop pipe. [Refer fig 4.6]. The green dot is the starting point of the robot, and the red dot is the endpoint. The robot's trajectory is represented in purple color.

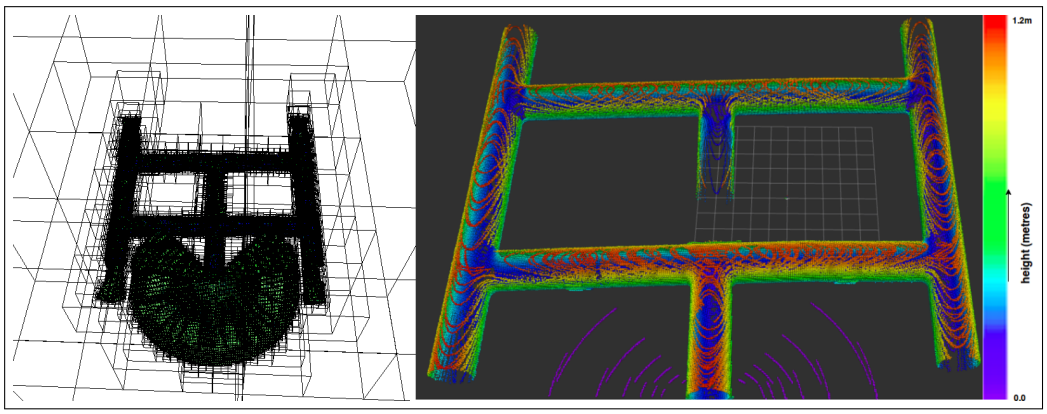


Figure 5.12: An Octomap is constructed inside a closed-loop pipe network. The left image shows how an Octomap is represented in the form of the octree structure and is stored in the voxel grids. The right image shows how an Octomap is constructed on top of Google Cartographer for memory-efficient mapping. The color bar represents the height of the pipe.

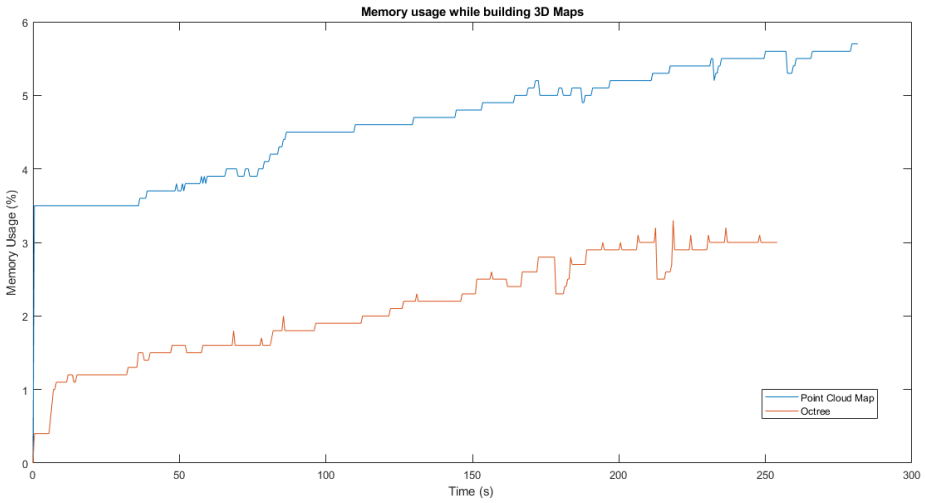


Figure 5.13: Comparison of memory utilization in constructing a point cloud-based mapping and an octree-based mapping for a closed-loop pipe network.

For mapping a multi-junction pipe, a 2D rotating LiDAR was used with the same configuration as Solid State LS02 LiDAR, which was available at RAM Lab. The purpose of using a rotating 2D LiDAR is that 3D LiDAR is far too costly and quite cumbersome to fit into the PIRATE. This test aimed to check whether a 3D map can still be created by adding a dynamixel motor to the 2D LiDAR. Figure 5.14 shows an Octomap created on top of Cartographer using a 2D rotating LiDAR. However, while traversing the robot inside the pipe, it was observed that the computation load was extremely high, especially at the turnings. As a result, the real-time factor decreased to 0.2, deteriorating the quality of the map. Overall, the performance of Cartographer is pretty good in both small and large scale maps.

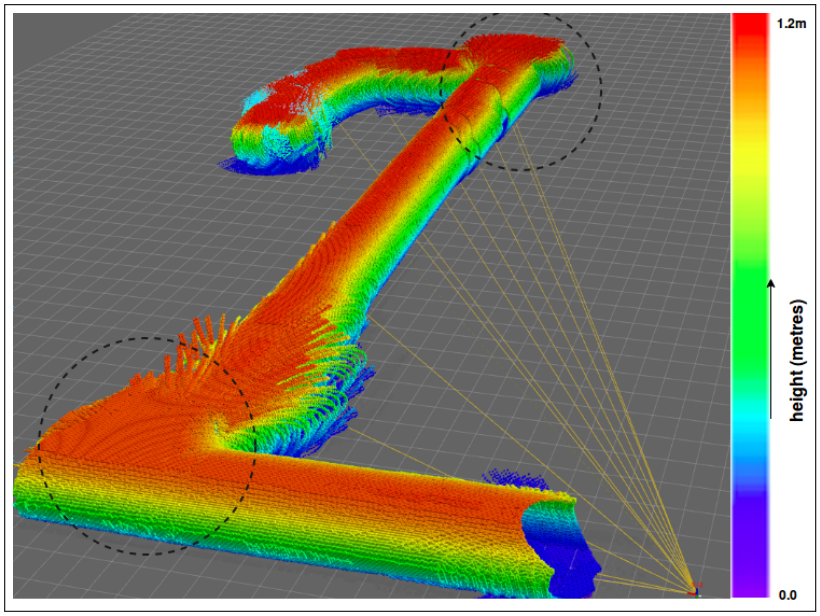


Figure 5.14: A 3D Octomap built on top of Cartographer using a 2D rotating LiDAR in a multi-configuration pipe. [Refer fig 4.6]. The black dotted circle indicates the haziness, which is due to the computation load in Gazebo, thus making the real-time factor drop to below 0.6 while making a turn. The yellow lines are the constraints developed while building submaps in Cartographer. The color bar represents the height of the pipe.

5.2 Real-time Results

In this section, the SLAM algorithm is applied to real-time. Firstly, the Jackal was tested in a 5.95m straight pipe-like environment and then in a T-junction pipe like environment. A 2D Occupancy grid map is obtained as shown in figure 5.15 and 5.16. On applying three algorithms, all are capable of mapping, but the difference is not much due to the small environment, as seen in the table 5.4 and 5.5. Because of the limited space available, it was tested in a small environment. Nonetheless, the proposed algorithm works and is capable of mapping a large-scale environment as well.

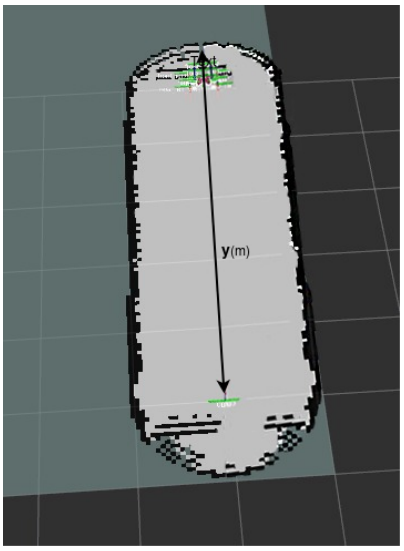


Figure 5.15: A 2D Occupancy grid map constructed in a straight pipe-like environment

Table 5.4: Estimated length 'y' for different SLAM algorithm with ground truth as 5.95m in a straight pipe like environment

Algorithm	Length 'y' (m)
Gmapping	5.88
Weight-Switching Gmapping	5.91
Weight-Switching Cartographer	5.91

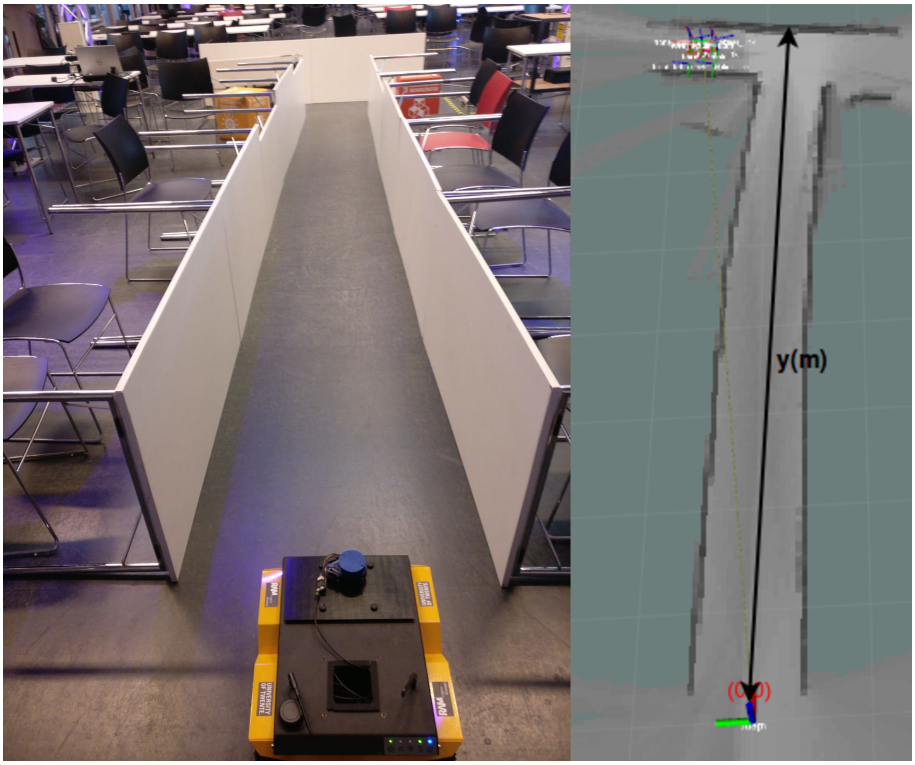


Figure 5.16: Testing of Jackal in T-pipe like environment. The left image shows the environment set-up while the right images shows the construction of Occupancy grid map.

Table 5.5: Estimated length 'y' for different SLAM algorithm with ground truth as 7.7m in a T-pipe like environment

Algorithm	Length 'y' (m)
Gmapping	7.389
Weight-Switching Gmapping	7.538
Weight-Switching Cartographer	7.650

Next, before integrating the 2D solid-state LiDAR to the PIRATE, the LiDAR was tested inside a 120mm pipe, as seen in figure 5.17. It is capable of detecting the curve as seen from the figure, but the number of samples point is quite less for constructing a good quality map. Also, the LiDAR needs to be placed at an angle to get the maximum number of scan points. Firstly, it was tested in a 120mm curved pipe, as seen in figure 5.18. It was observed that on moving the PIRATE inside the pipe, the LiDAR could detect only one side of the pipe and fails to detect the other side due to its minimum range and, therefore, fails to build a map. To overcome the minimum range issue, it was further tested in a 196mm pipe, as shown in figure 5.19. In this case, it can detect both sides of the pipe, but the map quality is poor due to the limited sample point of the 2D LiDAR. The number of points can be increased by adding a servo motor, but

overall it would increase the weight of the entire sensing mechanism that the PIRATE might not be able to handle. However, it can be resolved by using other better LiDAR.

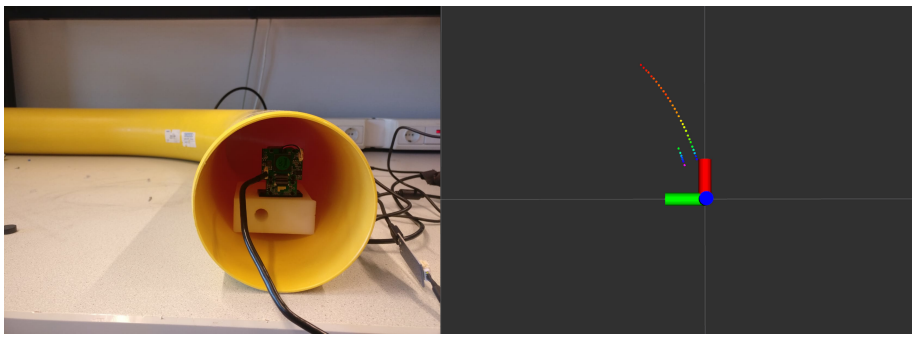


Figure 5.17: Visualization of a laser scan from a 2D solid-state LiDAR inside a curve pipe. The left image shows the LiDAR kept inside a 120mm pipe while in the right figure, its corresponding scan could be seen in Rviz.

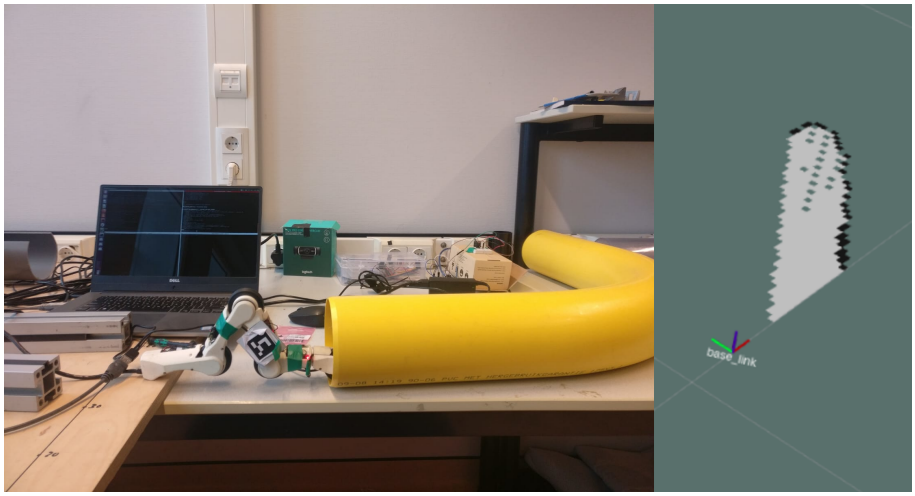


Figure 5.18: Testing of PIRATE in a curved pipe of diameter 120mm. The left image shows the environment set-up while the right images shows the construction of Occupancy grid map. It fails to produce the map since the LiDAR is unable to detect the surface of the pipe due to its minimum range.

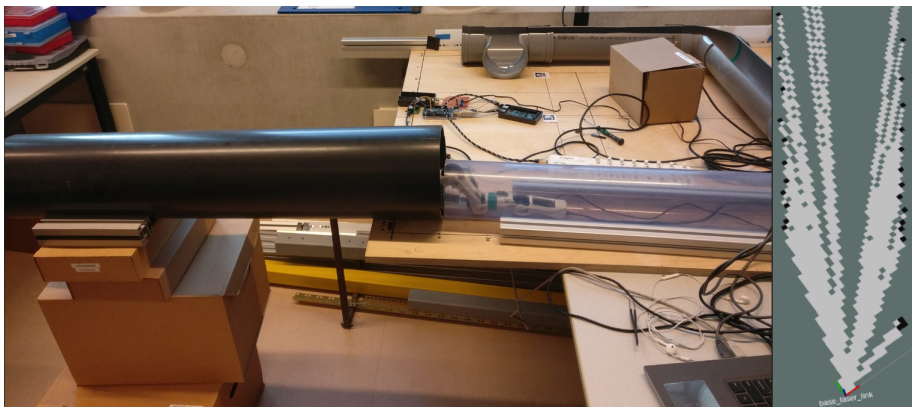


Figure 5.19: Testing of a PIRATE in a straight pipe of diameter 196mm. It is able to detect both sides of the pipe; however, the PIRATE cannot clamp in a pipe with a diameter >150mm.

Overall, it has been noted that the SLAM framework can be constructed for PIRATE in a larger pipe with the available sensor, but a bigger version of PIRATE is required since the current version cannot clamp inside a pipe with diameter > 150mm. For smaller pipe, LiDAR with minimum range < 5cm is recommended to be able to build quality maps.

5.3 Comparative study on Particle filter and Graph-based algorithm

This section presents a comparative analysis of the proposed methods based on the outcome of the experiments. Particle filter-based Gmapping has been one of the most commonly used SLAM frameworks due to its simple integration and operation. Graph-based Cartographer is mostly used for large-scale maps and loop closure detection. When the two proposed algorithms are compared, both tend to have comparable accuracy in short trajectories. However, constraint-based Weight-Switching Cartographer outperforms the junction-based Weight-Switching Gmapping algorithm as seen from the table 5.1, 5.2 and 5.3. Also, because of its low computation and memory consumption, Cartographer is an ideal candidate for the SLAM system within pipes as compare to Gmapping, as shown in figure 5.20. Both CPU load and memory usage is higher for Gmapping. Furthermore, because of its parameter versatility, Cartographer is more robust. Nevertheless, the integration of Cartographer to any application takes a considerable amount of time due to its complex architecture. It is also challenging to tune several parameters at once because each parameter can affect the quality of the map. But once implemented and tuned successfully, it can certainly produce accurate results in any environment.

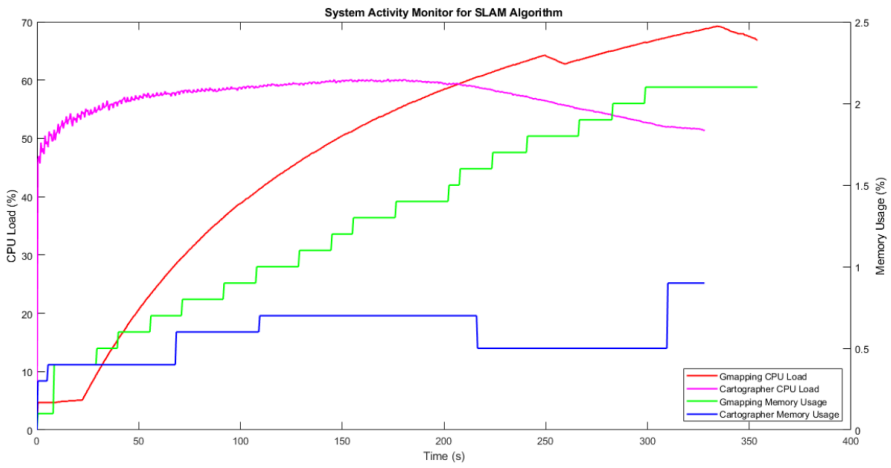


Figure 5.20: Evolution of CPU load and memory usage for different SLAM algorithm inside a straight pipe. Here memory usage represents the percentage of Random Access Memory (RAM) used by the process.

Apart from providing SLAM framework, Cartographer can also be used as a self-localization technique in a given map similar to AMCL. The two localization methods are compared in two different environments, i.e., a long straight pipe and a T-junction pipe. From the figure 5.22a, 5.22, it can be concluded that AMCL performs better than the Cartographer. This result is due to the fact the AMCL resamples the particles based on the difference in odometry and previous particle location. Once the odometry data is good, the particle point cloud region converges very quickly as compared to the pose graph optimization technique of Cartographer, which happens in a batch process only for the completed submaps. Based on the correlation score, the resampling process occurs at a faster rate, thus improving the overall localization. The only downside of using AMCL is that a pre-built map is required for localization, while Cartographer itself can act as a SLAM framework and self-localization technique.

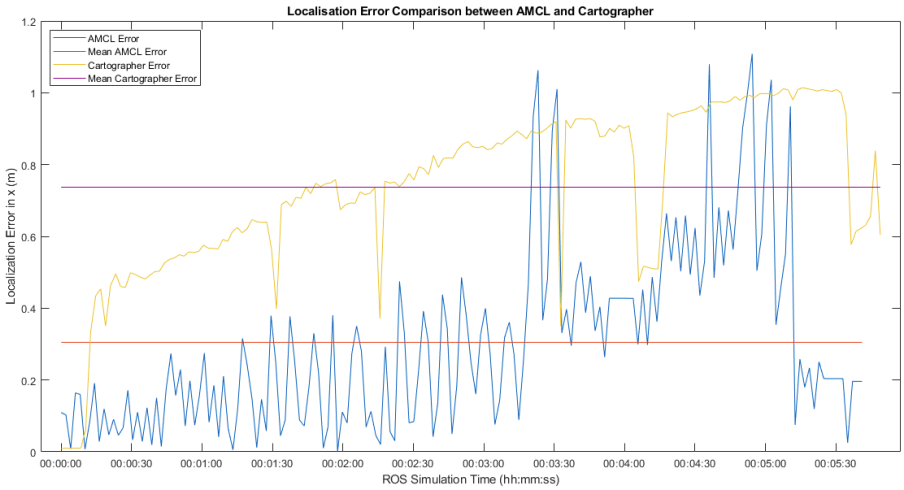
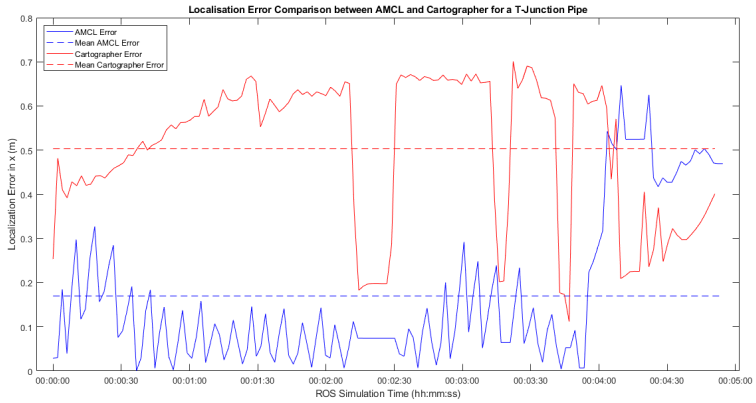
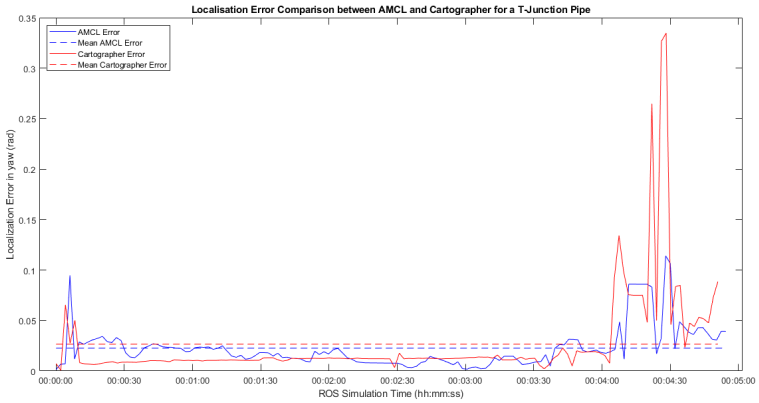


Figure 5.21: Comparison of Absolute Localization error in x between AMCL and Cartographer inside a straight pipe. The mean error for AMCL is less than that of Cartographer.



(a)



(b)

Figure 5.22: Comparison of Absolute Localization error in x and yaw between AMCL and Cartographer inside a straight pipe with a T-junction. In the bottom image, the peaks appear once the robot starts turning within the T-junction. The mean error in x and yaw for AMCL is less than that of Cartographer.

5.4 Discussion

To improve the critical assessment of the work carried out in this project, the strength and weaknesses of the suggested SLAM algorithm are thoroughly discussed in this section.

Strengths

- *Independent of pipe environment:-* The proposed algorithms are independent of the pipe environment. The environment inside the pipes may be textureless, or it may have cracks, a corroded portion, or any other textures that can be easily perceived. In such cases, generalizability is necessary for pipe inspecting robots, especially when they are used for map construction or exploration inside the pipes. Both the algorithms are capable of building the map for any in-pipe environment.

- *No dependency on Lighting condition:-* Lack of sufficient illumination can greatly affect navigation task, especially inside pipes and tunnels. The use of range-based sensors gives an advantage over a visual system that they can easily operate in an environment having inadequate lighting conditions. The 2D LiDAR makes it much reliable in such scenarios.

- *Repeatability:-* The experiments are repeated to observe the performance of the individual algorithm. Overall, it can produce similar results with a slightly different estimation of pipe length, which is due to sensor noise.

- *Scalability:-* The proposed algorithms are scalable to wider and complex pipe environments, especially graph-based SLAM.

- *Cost-effective:-* The proposed SLAM framework only requires a 2D LiDAR and odometry data. It can work with low-resolution LiDAR as well, making it a low-cost solution. The 2D mapping can be further expanded to 3D by simply adding a small motor to the 2D LiDAR, thus eliminating the very costly 3D LiDAR requirement.

Weaknesses

- *Risk of Slippage:-* Wheel odometry is susceptible to systematic and non-systematic errors. The most significant being wheel slippage which is due to the smooth pipe surface or due to low wheel traction. The error induced by the slippage can affect the localization and may degrade the quality of the map.

- *Absence of PIRATE model in simulation:-* The simulated test were performed on a differential drive robot instead of the PIRATE model. Hence, the exact dynamics were not modeled for testing purposes.

6 Conclusion and Future Recommendations

6.1 Conclusion

In this project, the aim was to investigate various Simultaneous Localization and Mapping solutions for PIRATE inside a pipe network. The use of inspection robots in the pipe network is very beneficial: their small size allows them to move through most of the pipes, eliminating the need for further excavation. This project started with a set of research questions, i.e.,

Is it possible to build a map and localize inside the pipe network by only using a range or vision sensor on the robot?

To answer this question, two different pipe environments were constructed, i.e., a long straight textureless pipe and a pipe with a T-junction to check the effectiveness of the two sensors. It was observed that both of them failed to localize inside the straight pipe. This is due to the lack of salient features but were successful in localizing inside a T-junction pipe. Hence, it proves that they are ineffective if the pipe has no textures or features and will likely fail in such an environment. Nonetheless, LiDAR is more effective than the camera because it can work in low visibility, with less memory consumption, and can also detect low geometric areas (small changes in pipe diameter), but cannot be used as a stand-alone mapping and localization sensor. The probable solution to this problem is using wheel encoder data in such an environment. Then a further research question resulted from this study, i.e.,

How can a low-cost IMU (Inertial Measurement Unit) sensor measurement help to improve the localization of a robot in a pre-built map of the pipe network?

This is achieved by employing sensor fusion between wheel encoders and IMU measurements. To validate the obtained fused data, the UMBARK benchmark strategy was followed. It was noticed that the fused data is much reliable and can be used for the localization of the mobile robot. To test the applicability of fused data in Localization, AMCL was performed in two pre-built maps of the pipe environment. From the results, it was found that sensor fusion significantly improved the overall localization technique and thus can further be used for the map building process as well. The third research question was

How can the mapping accuracy of the Gmapping algorithm be improved for different pipe segments?

To improve the mapping accuracy of the existing Gmapping algorithm, a pipe segment-based weighting parameter was introduced for the scan measurement, which enabled it to switch the weights between the LiDAR measurement and odometry measurement for pose estimation. To validate the proposed approach, it was compared to the existing Gmapping approach inside two pipe environments. The result showed that the estimated pipe length for the proposed method was much closer to the ground truth, thus validating the effectiveness of the proposed design. Finally, the last research question was

How can Google Cartographer be exploited effectively for building a 2D-3D map inside the pipe environment?

To apply Cartographer effectively inside the pipe network, a constraint-based Weight-Switching strategy is adopted. The relative motion constraint governs this switch. A combination of translation and rotational constraints can be identified near the pipe junctions

using a search window within a submap, which enables the Local SLAM to trust in the LiDAR measurement. To validate the proposed algorithm, it was tested in the same two pipe environments in which the Gmapping experiment was conducted. The result showed that the Cartographer estimated the pipe length better than the Weight-Switching Gmapping algorithm. Further, to test the scalability and loop closure detection, complex environments were constructed. It was noted that it was successful in detecting a re-visited path in a complex environment, thus validating loop closure as well as scalability. Finally, to increase the efficiency of the 3D mapping of Cartographer, an Octomap was integrated into the framework. It was further compared with a point cloud map to evaluate the mapping efficiency. The result highlighted that memory usage for Octomap in building large scale maps was significantly less than in contrast to the point cloud map.

From the obtained results, it can be observed that both the SLAM approaches work quite well in different pipe segments, but there is still some error in the localization as no sensors are noise-free. However, in the pipe inspection application, a couple of meters of inaccuracy can always be acceptable rather than excavating a huge area for the replacement of the faulty pipe. To conclude from the experimental results, the best combination for 2D can be Google Cartographer for mapping and AMCL for localization. And for 3D, Octomap for mapping and Cartographer for localization is the evident combination. For the real-time PIRATE experiment, although it was unable to build the map properly due to sensor limitation, however with a better sensing mechanism, it is possible to build the SLAM in both smaller and bigger pipes. Overall, the proposed SLAM framework in this thesis for the PIRATE set up a baseline for achieving higher autonomy levels like autonomous navigation and exploration inside the pipe network.

6.2 Future Recommendations

In this section, recommendations are made for future work. In general, the main reason for this study was to achieve the next step of autonomy for the PIRATE by introducing the SLAM platform for the pipe network. This SLAM framework presented in this thesis offers a good base for future development in a multidisciplinary field. The future scope can be listed as:

1. **Further development in SLAM and autonomous navigation of PIRATE**

    - Based on the results obtained so far, there is still a scope of development in the proposed Gmapping algorithm. Currently, there is no separate algorithm for junction detection, change in pipe diameter. Integrating it to Gmapping can optimize the weighing parameter α, and as a result, performance can be much improved with respect to mapping and localization.
    - In pipes with a diameter ranging in between 110mm-150mm, a 1D LiDAR can be useful since the minimum range of 1D LiDAR is 1-2cm and is quite small, which can be easily integrated into PIRATE. Adding a small servo motor to it can produce the required 2D maps.
    - Google cartographer offers a variety of parameters. Further analysis can be performed in order to tune the parameters to improve map quality and guarantee higher flexibility.
    - A 2D LIDAR with a minimum range less than of 10cm with a sufficient number of samples can be integrated into the PIRATE to improve the quality of the built-map further.
    - Adding a small servo motor to the PIRATE can rotate the 2D LiDAR in the z-axis, which can help in constructing a 3D map using the proposed 3D SLAM algorithm in this thesis.

- Having the SLAM framework, the next step of autonomy for the PIRATE, i.e., autonomous path-planning, can be achieved. There are many path-planning algorithms like Dijkstra, Carrot planner, which can be integrated into the existing ROS framework for PIRATE. Introducing these planner modules will help the PIRATE to navigate inside the pipe autonomously.

2. **Future direction in Inspection**

   - A small camera along with few LEDs or a night-vision camera could be integrated into the PIRATE for only inspection purposes so that the requirement for recording video for building a map is not required. A deep learning method or common computer vision algorithm can be applied more efficiently using the camera data for crack and leakage detection, as shown in figure 6.1. Since the SLAM framework is already available, locating these cracks could be much easier.
   - Similarly, an array of cheap ultrasonic sensors can be added to the PIRATE for detecting minute deformations and cracks inside the pipe by analyzing the guided ultrasonic wave.

3. **Alternative Simulator**

   - The physics engine of Gazebo was unable to handle the simulated PIRATE model. It caused frequent vibrations when the PIRATE clamps to the pipe wall, making it unstable and thus not suitable for simulation. However, other physics simulators like V-REP or ARGoS could be tried for the simulated PIRATE model. If the problem still remains, a new custom simulator should be designed to incorporate the physics of the PIRATE model.

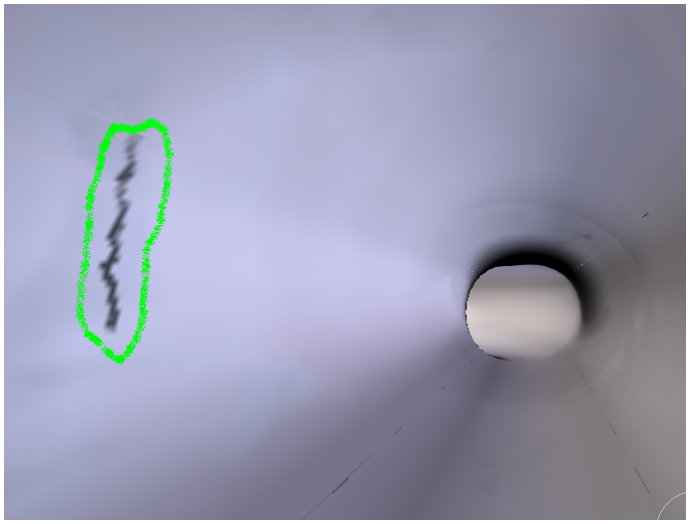


Figure 6.1: Crack Detection using Canny Edge detector algorithm

A Appendix 1: PIRATE Odometry motion model and Transformations

An important aspect of SLAM is defining the motion model for the robot. The motion model corresponds to the robot's movement between two positions based on the control command. Here the position of the robot can be expressed as s=(x, y, θ) in a 2D environment. As discussed in chapter 2, the motion model is defined as

$$p(s_t|s_{t-1}, u_t) \tag{A.1}$$

where, s_t is the current pose, s_{t-1} is the previous pose and u_t is the control command.
In equation A.1, it shows the current pose of the robot is the probability distribution of its previous pose and the control command. Using the current pose of the robot, the map can be constructed. The odometry data is retrieved using the rotary wheel encoders. PIRATE has in total of 6 wheels, which can be used for determining the position. However, after analyzing the wheel encoders data, it was observed that only three wheels (Wheel 2, 3, and 5), as shown in figure A.1, that provide reliable angular position and twist data which can be used for further calculation. For x-position, only the angular displacement data is used for calculation of the distance traveled by each wheel. The odometry calculation should be linked to the base link of the robot, and since wheel 2 is the closest to the base link, hence odometry data from wheel 2 should suffice the requirement of the motion model. To calculate the odometry, the change in the angular displacement by the wheels are used.

$$\Delta\theta = \theta_f - \theta_i \tag{A.2}$$

$$x = x_{initial} + R * \Delta\theta \tag{A.3}$$

where $\Delta\theta$ is the angular displacement, R is the radius of the wheel.
It should be noted that this odometry value needs to be transformed to the base link so that for the appropriate tf transformation in ROS.

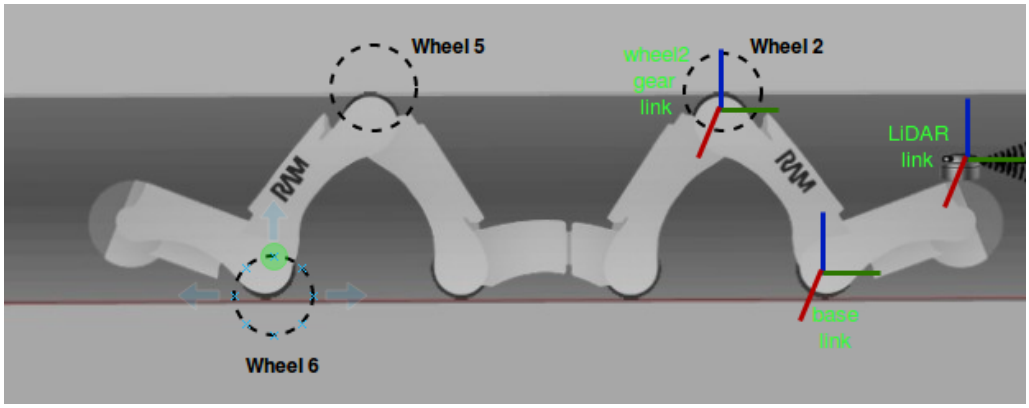


Figure A.1: PIRATE link transformation. Red corresponds to x-axis, green as y-axis and blue as z-axis

In order to transform the odometry data from wheel 2 to the base link, the pitch angle (θ) was used, as shown in figure A.2. Since the pitch angle remains constant once the PIRATE clamps, a static transformation of $0.09 * cos(90 - \frac{\theta}{2})$ was added to the calculated odometry using wheel 2 in the x-direction.
Also, for the LiDAR, a static transformation was added from the base link for all the measurements. For the orientation measurement, IMU embedded in the PICO board was used. The

IMU was a 6 DOF with 3DOF of accelerometer and 3DOF of a magnetometer. Roll and pitch angles can be easily calculated from an accelerometer; however, for yaw calculation, the data from the magnetometer was not reliable. Hence, for yaw calculation, an IMU with a 3DOF gyroscope can be handy.

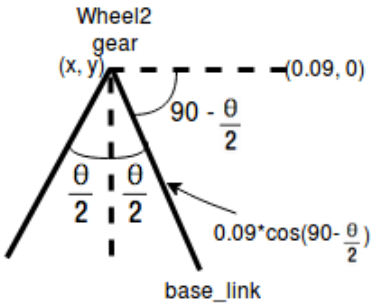
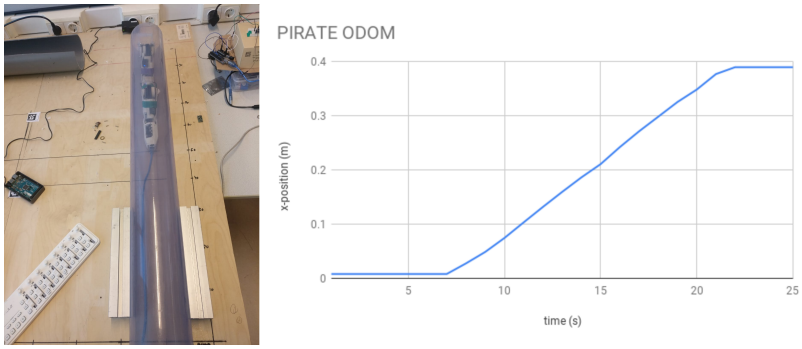
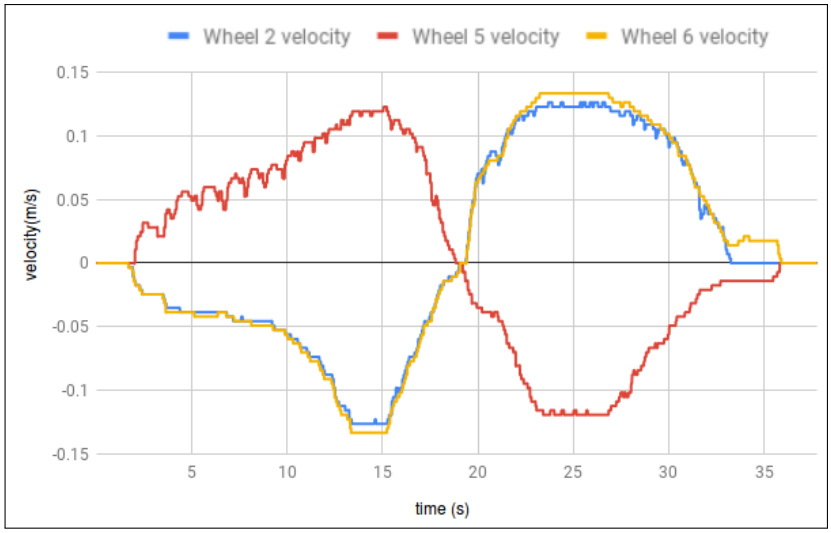


Figure A.2: Transformation from wheel 2 gear link to base link



(a) Odometry of PIRATE transformed from wheel 2 to base link

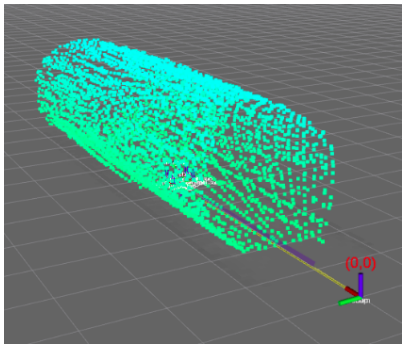


(b) Twist from the wheels 2, 5 and 6

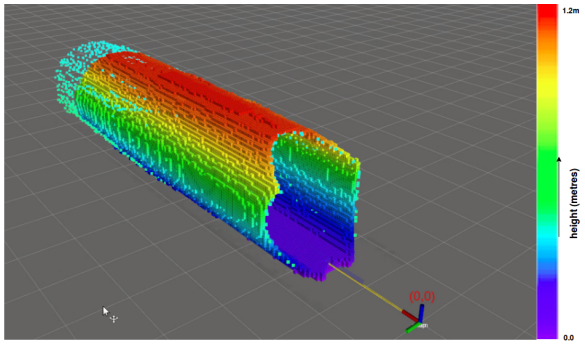
Figure A.3: PIRATE sensor data visualization

The advantage of PIRATE is that it offers more traction due to its 6-wheel segment and thus chances of less slippage. Most importantly, due to its two V-shaped clamping action, the chances of yaw error (θ) is quite less when it is moving in a straight pipe as compared to mobile robots.

B Appendix 2: Other 3D Mapping Plots



(a) Point cloud map of a straight pipe obtained from Cartographer



(b) Building an Octomap on top of Cartographer since Octomap is a mapping algorithm

Figure B.1: Illustrates of 3D map built using Cartographer and Octomap

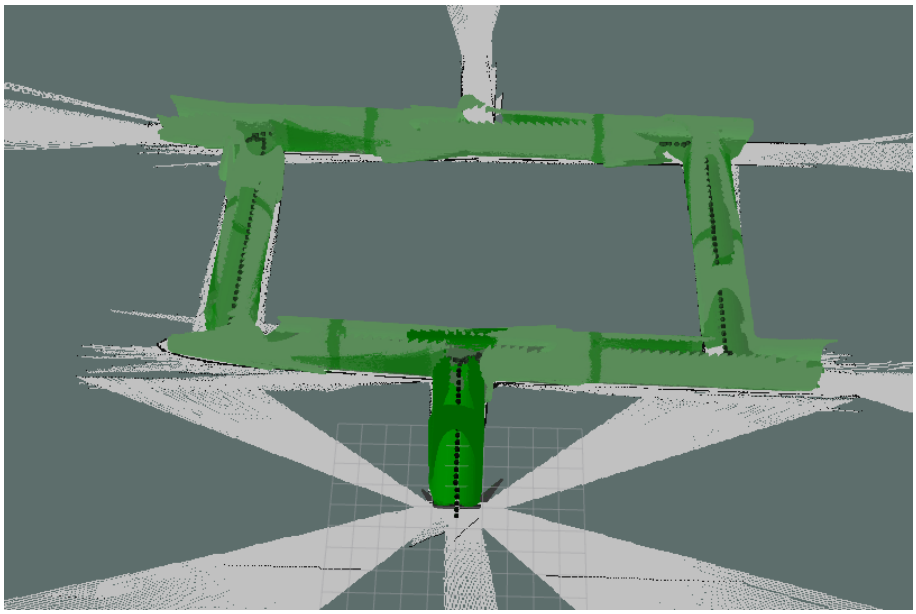


Figure B.2: A 3D map obtained from RTAB-Map using Microsoft Kinect demonstrating appearance-based loop closure detection. The black dot inside the map represents the robot trajectory inside the pipe.

From the table B.1, it is clear that the memory consumption to build the same set of map from a vision-system is significantly high. Moreover, to get the required map as shown in figure B.2, it requires tuning of a bunch of parameters which takes considerable amount of time and effort.

Table B.1: Representation of memory requirement for running 3D-MAP using Kinect and 3D LiDAR

3D-MAP	
Method	**ROSBAG Memory size(GB)**
MAP with Kinect	43.4
MAP with 3D LiDAR	4.1

C Appendix 3: Pseudocode for Algorithms

In the pseudocode, each sample (particle) s is defined as a tuple $s = (x, w, M)$, where x represents the pose vector, scalar weight w and map matrix M. Since it is a 2d mapping algorithm, map matrix M corresponds to $M \in \mathbb{R}^{m \times n}$, where m and n correspond to the grid cells defining whether a particular grid cell is occupied or not. Measurement from laser scan sensor is defined by the vector z, where its index defines the scanning angle, and its value tells the measured distance at that angle. The motion command i.e. odometry u (x, y, θ) changes relative to the local coordinate frame of the robot.

Algorithm 2 Implementation of ROS Gmapping SLAM algorithm, adapted from [30]

Require: : **(Initialized Once)**
$T_{resample}$, threshold value
N, number of particles
Require: : **(Every cycle)**
S_{t-1}, sample set of the previous time step
z_t, most recent laser scan
u_t, most recent odometry measurement (x, y, θ)
Ensure: S_t, new sample set (It contains all the particles, and thus estimated map and pose)
$S_t = \{\}$
for all $s_{t-1}^{(i)} \in S_{t-1}$ **do**
$<x_{t-1}^{(i)}, w_{t-1}^{(i)}, M_{t-1}^{(i)}> \leftarrow s_{t-1}^{(i)}$
// scan-matching
$x_t'^{(i)} = x_{t-1}^{(i)} \oplus u_t$
$\hat{x}_t^{(i)} = \text{argmax}_x \, \text{p(x}| \, M_{t-1}^{(i)}, z_t, x_t'^{(i)})$
if $\hat{x}_t^{(i)}$ = failure **then**
$x_t^{(i)} \sim \text{p}(x_t | \, x_{t-1}^{(i)}, u_t)$
$w_t^{(i)} = w_{t-1}^{(i)} \cdot \text{p}(z_t | \, M_{t-1}^{(i)}, x_t^{(i)})$
else
$x_t^{(i)} \sim \text{p}(x_t | \, M_{t-1}^{(i)}, x_{t-1}^{(i)}, z_t, u_t)$
$w_t^{(i)} = w_{t-1}^{(i)} \cdot \text{p}(z_t | \, M_{t-1}^{(i)}, x_{t-1}^{(i)}, u_t)$
end if
// update map
$M_t^{(i)} = \text{integrateScan}(M_{t-1}^{(i)}, x_t^{(i)}, z_t)$
// update sample set
$s_t^{(i)} \leftarrow < x_t^{(i)}, w_t^{(i)}, M_t^{(i)} >$
$S_t = S_t \cup s_t^{(i)}$
end for
// resample if required
$N_{eff} = \frac{1}{\sum_{i=1}^{N} (\tilde{w}_t^{(i)})^2}$
if $N_{eff} < T_{resample}$ **then**
$S_t = resample(S_t)$
end if

The pose compounding operator $\oplus$ [48], performs addition of the odometry vector to the previous pose. To limit the risk of sufficient particle depletion, resampling is done in a more efficient manner. Resampling is only performed when N_{eff} drops below the threshold value $T_{resample}$ (generally $T_{resample} = N/2$), where N represents the total number of particles.

1. The algorithm is initialized with the initial guess of the robot's pose $x_t'^{(i)}$ represented by particle i and it starts from $[0, 0, 0]^T$

2. Using the initial guess of the robot's pose, scan-matching is performed on $M_{t-1}^{(i)}$ based on the motion model, hence limiting the search region. Based on matching score of the scan-matching, a new estimate $\hat{x}_t^{(i)}$ is generated.

3. A new pose estimate is derived using the motion model, observations, prior pose of the robot and map built till then. This distribution This distribution of the particles involves the generating of Gaussian distribution based on the particles drawn near the scan matching estimate $\hat{x}_t^{(i)}$. The weights of the particle $w_t^{(i)}$ are updated accordingly.

4. Using the function integrateScan, new map is calculated based on $M_{t-1}^{(i)}$, $x_t^{(i)}$, z_t.

5. Resampling is performed when $N_{eff} < T_{resample}$ (threshold). For this, the weights are normalized first $\tilde{w}_t^{(i)}$.

D Appendix 4: Dynamixel motor integration for rotating LiDAR

A dynamixel motor is one of the most commonly used actuators designed for multi-joint robot configurations. Here this motor is used for rotating a 2D LiDAR along the z-axis to get a 3D point cloud for building 3D SLAM framework. There are a few steps required to integrate the motor in Gazebo:

1. The first step is to integrate the motor to the LiDAR by editing the URDF file in ROS, where the robot-LiDAR joint has been defined. Since it is rotating, the joint type is selected as 'continuous,' and the axis of rotation is chosen to be z-axis.

2. Secondly, the motor hardware interface needs to be defined, which is acting in between the robot base frame and the LiDAR link. Here an effort joint interface is chosen from the control action point of view since it is easier to apply position control on such joint in ROS. The hardware configuration is given by

```
<transmission name="base_to_laser_trans">
    <type>transmission_interface/SimpleTransmission</type>
    <actuator name="motor1">
      <mechanicalReduction>1</mechanicalReduction>
    </actuator>
    <joint name="base_to_laser_joint">
      <hardwareInterface>EffortJointInterface</hardwareInterface>
    </joint>
</transmission>
```

3. The third step is to define the joint controller configuration using which the position of the motor can be controlled. A Proportional Integral (PD) control is applied to achieve this. The PI control values are tuned and chosen to be: P=1, I=1.05. In this joint position controller, a set point (position) is given by the user, in this case, a small value of 0.007 is given so that the motor rotation is smooth and the LiDAR can perceive the environment easily.

4. While running the ROS node, make sure the following node and the configuration parameter defined in previous step is added in the launch file

```
<node name="controller_spawner" pkg="controller_manager"
args="base_to_laser_joint_position_controller
joint_state_controller --shutdown-timeout 3"/>
```

However, it should be kept in mind that the ROS GAZEBO control plugin has been added in the URDF file of the robot because all the controllers applied in Gazebo uses this plugin for proper functioning of the system. In addition, if the rotating LiDAR scan is not smooth, increase the LiDAR inertia.

Table D.1: LiDAR inertia Value

Inertia parameter	Value
i_{xx}	0.0002835
i_{yy}	0.0002835
i_{zz}	0.000324

E Appendix 5: Feature Extraction Algorithms

Features / Algorithms	Corner detector	Blob detector	Rotation invariant	Scale invariant	Affine invariant	Repeatability	Localization accuracy	Robustness	Efficiency
Harris	x		x			+++	+++	++	++
Shi-Tomasi	x		x			+++	+++	++	++
Harris-Laplacian	x	x	x	x		+++	+++	++	+
Harris-Affine	x	x	x	x	x	+++	+++	++	++
SUSAN	x		x			++	++	++	+++
FAST	x		x			++	++	++	++++
SIFT		x	x	x	x	+++	++	+++	+
MSER		x	x	x	x	+++	+	+++	+++
SURF		x	x	x	x	++	++	++	++

Figure E.1: Chart of different feature detector available [6]

F Appendix 5: Communication

F.1 PIRATE Connection

To be able to communicate with the PIRATE through your laptop, follow the following steps:

1. Clone the GitLab PIRATE repository to your local machine
2. Plug in the adapter and wait till you see a white light on the MIDI panel
3. After the white light appears, insert the USB to your PC
4. Go to the workspace PIRATE/catkin_ws–> Type 'source devel/setup.bash'
5. Use the command 'sudo chmod 777 /dev/ttyACM0' to allow the connection between Arduino and to your pc
6. Launch the file 'roslaunch pirate_seq seq.launch' to activate all the nodes for the PIRATE and visualize in Rviz. The LiDAR transformation is included in the launch file. In case there is an error, then probably rosserial is missing. To solve this issue, two libraries need to be installed. In the command window type:
   - sudo apt-get install ros-kinetic-rosserial-arduino
   - sudo apt-get install ros-kinetic-rosserial
7. Once the connection is established, and all the ROS nodes are launched, to echo any ROS Topic data, first press the Play button followed by the Cycle button. On pressing the Cycle button, blue LEDs connected to the PICO board starts blinking. These blue LEDs tell about the data communication from the lower level to the higher-level architecture of the PIRATE. Finally, the data can be visualized in the respective ROS Topics.

F.2 NVIDIA Jetson TX2

This hardware was used on the Jackal robot as the main processing unit. In order to operate wirelessly, move the Jackal in an environment, a connection is required from your laptop, which can be achieved via ssh.
To do ssh Nvidia Jetson from laptop

1. Connect eduroam with same credentials and wifi settings as done in your local pc.
2. Type ifconfig in the jetson terminal to check the ip address —-> check the inet address for wlan0 since you are connecting wirelessly.
3. Type: 'sudo nvidia@wlan inetAddress' in your pc
4. If it does not connect, run 'sudo ssh service start' in the Jetson terminal and then repeat step 2 and 3. To check status type: 'sudo ssh service status' and to stop, type: 'sudo ssh service stop'.

Bibliography

[1] C. Pulles, Edwin Dertien, Pol J, and R. Nispeling. Pirate, the development of an autonomous gas distribution system inspection robot. *Journal of Membrane Science - J MEMBRANE SCI*, 01 2008.

[2] M. Montemerlo and S. Thrun. Simultaneous localization and mapping with unknown data association using fastslam. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 2, pages 1985–1991 vol.2, Sep. 2003.

[3] Mathworks. Path Planning in Environments. http://tiny.cc/stj1cz.

[4] H. Rapp W. Hess, D. Kohler and D. Andor. ROS Cartographer Integration. https://github.com/googlecartographer/cartographer/blob/master/docs/source, 2016. [Online; accessed 19-July-2019].

[5] Alexei A. Makarenko, Stefan B. Williams, Frederic Bourgault, and Hugh F. Durrant-Whyte. An experiment in integrated exploration. In *IEEE/RSJ International Conference on Intelligent Robots and System*, pages 534–539, 2002.

[6] F. Fraundorfer and D. Scaramuzza. Visual odometry : Part ii: Matching, robustness, optimization, and applications. *IEEE Robotics Automation Magazine*, 19(2):78–90, June 2012.

[7] Edwin Dertien, Stefano Stramigioli, and Kees Pulles. Development of an inspection robot for small diameter gas distribution mains. pages 5044 – 5049, 06 2011.

[8] N.M. Geerlings. Design of a high-level control layer and wheel contact estimation and compensation for the pipe inspection robot pirate, 2018.

[9] Mark Reiling. Implementation of a monocular structured light vision system for pipe inspection robot pirate. 2014.

[10] R. Minichan, R. Fogle, and A. Marzolf. H-canyon air exhaust tunnel inspection vehicle development.

[11] Zhixiang Li, Jin Zhu, Changzhong He, and Wei Wang. A new pipe cleaning and inspection robot with active pipe-diameter adaptability based on atmega64. pages 2–616, 09 2009.

[12] Erich Rome, Joachim Hertzberg, Frank Kirchner, Ulrich Licht, and Thomas Christaller. Towards autonomous sewer robots: the makro project. *Urban Water*, 1(1):57 – 70, 1999.

[13] Ivan Virgala, Alexander Gmiterko, and Michal Kelemen. Motion analysis of in-pipe robot based on sma spring actuator. 2013.

[14] Giancarlo Canavese, Luciano Scaltrito, Sergio Ferrero, C.F. Pirri, Matteo Cocuzza, Marco Pirola, S Corbellini, G Ghione, Chiara Ramella, Francesca Verga, Andrea Tasso, and Alberto Di Lullo. A novel smart caliper foam pig for low-cost pipeline inspection—part a: Design and laboratory characterization. *Journal of Petroleum Science and Engineering*, 127, 01 2015.

[15] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose Neira, Ian D. Reid, and John J. Leonard. Simultaneous localization and mapping: Present, future, and the robust-perception age. *IEEE Transactions on Robotics*, 32, 06 2016.

[16] D. Krys and H. Najjaran. Development of visual simultaneous localization and mapping (vslam) for a pipe inspection robot. In *2007 International Symposium on Computational Intelligence in Robotics and Automation*, pages 344–349, June 2007.

[17] H. Habibi Aghdam, H. A. Kadir, , and M. Zaman. Localizing pipe inspection robot using visual odometry. In *2014 IEEE International Conference on Control System, Computing and Engineering (ICCSCE 2014)*, pages 245–250, Nov 2014.

[18] A. C. Murtra and J. M. Mirats Tur. Imu and cable encoder data fusion for in-pipe mobile robot localization. In *2013 IEEE Conference on Technologies for Practical Robot Applica-*

tions (TePRA), pages 1–6, April 2013.

[19] Y. Bando, H. Suhara, M. Tanaka, T. Kamegawa, K. Itoyama, K. Yoshii, F. Matsuno, and H. G. Okuno. Sound-based online localization for an in-pipe snake robot. In *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 207–213, Oct 2016.

[20] K. Ma, M. Schirru, A. H. Zahraee, R. Dwyer-Joyce, J. Boxall, T. J. Dodd, R. Collins, and S. R. Anderson. Pipeslam: Simultaneous localisation and mapping in feature sparse water pipes using the rao-blackwellised particle filter. In *2017 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 1459–1464, July 2017.

[21] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, June 2006.

[22] Cyrill Stachniss. *Robotic Mapping and Exploration*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[23] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[24] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees, 2012.

[25] Aarne Halme. Sensors for mobile robots: Theory and application [book review]. *Robotics and Automation, IEEE Transactions on*, 12:922–, 01 1997.

[26] D. L. Hall and J. Llinas. An introduction to multisensor data fusion. *Proceedings of the IEEE*, 85(1):6–23, Jan 1997.

[27] Thomas Moore and Daniel Stouch. A generalized extended kalman filter implementation for the robot operating system. 302:335–348, 01 2016.

[28] D. Barbosa, A. Lopes, and R. E. Araújo. Sensor fusion algorithm based on extended kalman filter for estimation of ground vehicle dynamics. In *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 1049–1054, Oct 2016.

[29] E. B. Olson. Real-time correlative scan matching. In *2009 IEEE International Conference on Robotics and Automation*, pages 4387–4393, May 2009.

[30] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, Feb 2007.

[31] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. *Proc. IJCAI Int. Joint Conf. Artif. Intell.*, 06 2003.

[32] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Transactions on Intelligent Transportation Systems Magazine*, 2:31–43, 12 2010.

[33] G. Bresson, Z. Alsayed, L. Yu, and S. Glaser. Simultaneous localization and mapping: A survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 2(3):194–220, Sep. 2017.

[34] W. Hess, D. Kohler, H. Rapp, and D. Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278, May 2016.

[35] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. pages 343–349, 01 1999.

[36] Dieter Fox. Kld-sampling: Adaptive particle filters and mobile robot localization. 10 2001.

[37] G.A. Garza Morales. Increasing the autonomy of the pipe inspection robot pirate, August 2016.

[38] Jens Thielemann, Gøril Breivik, and Asbjørn Berge. Robot navigation and obstacle detection in pipelines using time-of-flight imagery. *Proceedings of SPIE - The International Society for Optical Engineering*, 7526, 02 2010.

[39] Robert Zlot and Michael Bosse. Efficient large-scale 3d mobile mapping and surface reconstruction of an underground mine. volume 92, 07 2012.

[40] Robert Zlot and Michael Bosse. Three-dimensional mobile mapping of caves. *Journal of Cave and Karst Studies*, 76:191–206, 12 2014.

[41] Nicholas J. Weidner. Underwater cave mapping and reconstruction using stereo vision. 2017.

[42] Mohammad O A Aqel, Mohammad H Marhaban, M Iqbal Saripan, and Napsiah Bt Ismail. Review of visual odometry: types, approaches, challenges, and applications. *SpringerPlus*, 5(1):1897, 2016.

[43] Jharna Majumdar Sandeep Kumar Malu. Kinematics, localization and control of differential drive mobile robot. *Global Journal of Research In Engineering*, 2014.

[44] Edwin Christian Dertien. *Design of an inspection robot for small diameter gas distribution mains*. PhD thesis, University of Twente, Netherlands, 7 2014.

[45] Carlos Rizzo, Francisco Lera, and Jose Villarroel. A methodology for localization in tunnels based on periodic rf signal fadings. *Proceedings - IEEE Military Communications Conference MILCOM*, pages 317–324, 11 2014.

[46] Johann Borenstein and Liqiang Feng. Umbmark : a method for measuring, comparing, and correcting dead-reckoning errors in mobile robots. 1994.

[47] Y. Abdelrasoul, A. B. S. H. Saman, and P. Sebastian. A quantitative study of tuning ros gmapping parameters and their effect on performing indoor 2d slam. In *2016 2nd IEEE International Symposium on Robotics and Manufacturing Automation (ROMA)*, pages 1–6, Sep. 2016.

[48] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4(4):333–349, Oct 1997.