University of Windsor
Scholarship at UWindsor

Electronic Theses and Dissertations | Theses, Dissertations, and Major Papers

8-31-2022

Study and Analysis of Mirror-like Object Detection for Autonomous Indoor Navigation using a 2D LiDAR

Deeptha Damodaran
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

 Part of the Robotics Commons

Recommended Citation

Damodaran, Deeptha, "Study and Analysis of Mirror-like Object Detection for Autonomous Indoor Navigation using a 2D LiDAR" (2022). *Electronic Theses and Dissertations*. 9598.
https://scholar.uwindsor.ca/etd/9598

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

Study and Analysis of Mirror-like object detection for autonomous indoor navigation using a 2D LiDAR

By

Deeptha Damodaran

A Thesis

Submitted to the Faculty of Graduate Studies

through the Department of Mechanical, Automotive and Materials Engineering

in Partial Fulfillment of the Requirements for

the Degree of Master of Applied Science

at the University of Windsor

Windsor, Ontario, Canada

2022

© 2022 Deeptha Damodaran

Study and Analysis of Mirror-like object detection for autonomous indoor navigation using a 2D LiDAR

APPROVED BY:

S. Alirezaee

Department of Electrical and Computer Engineering

D. Ting

Department of Mechanical, Automotive and Materials Engineering

---

J. Ahamed, Advisor

Department of Mechanical, Automotive and Materials Engineering

August 4th, 2022

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Simultaneous Localization and Mapping (SLAM) is the process of representing the spatial environment (mapping) while keeping track of position (localization) within the built map. SLAM is widely used in indoor navigation, where several types of objects and obstacles need to be mapped. Objects that pose an issue for laser or light-assisted indoor navigation include specular surfaces such as mirrors that cause light reflection.

This thesis aims to understand the characteristics of mirror-like objects in various arrangements. Experiments were conducted using a lidar-mounted mobile robot navigating with respect to one or more mirrors and mapping the environment. Several observations were made to understand the inaccuracies of laser scans in mirrored environments. The outcomes of these observations suggest that laser scans may be fully reflected off mirrors, causing no range or intensity data to be provided back to the robot and causing the map to develop areas of negative space. Objects or boundaries within the range of the lidar may be mapped behind the surface of the mirror, and self-detection may occur on the surface of the mirror. Some uncertainties may occur when more than one mirror is present in the environment.

As many different observations were noted, a potential solution approach is outlined, advising the use of clustering algorithms to identify and remove inconsistencies before building a map of an indoor environment. This research has applications in several industries. These could include autonomous robots navigating through the environment to perform specific tasks; mapping of malls, museums or office buildings where specular surfaces are common. Future work would include implementation of the solution approach and extension to different types of nondiffuse surfaces, including transparent and translucent surfaces.

This research could be most practical when one or more mirrors are present. applied in the mapping of indoor spaces with high amounts of optically unique surfaces, including modern office buildings, hospitals, and museums

DEDICATION

So many people have supported me through this long journey. I would like to dedicate this thesis to my incredible family and friends.

Appa, thank you for being such an inspiration and for always motivating me and everyone around you to be better. Amma, thank you, for always being so resilient, compassionate, and caring. You both make me so proud to be your daughter. To the rest of my family - brother and grandparents, thank you for always rooting for me.

Gregory, thank you for being a pillar of strength. Your encouragement and support through this journey have meant everything to me. Debra, thank you for the motivation and your support in the completion of my thesis.

To my greatest friends, Namratha and Rachan, thank you for being there for me always and through everything. Narayani, thank you for all the academic and emotional support.

I appreciate you all so much for everything.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. J. Ahamed, for his patience and guidance during my degree.

Thank you, for your continuous encouragement, and support. I would not have been able to make it this far without your understanding and expertise.

I would also like my committee members, Dr. D. Ting and Dr. S. Alirezaee, for agreeing to serve on my committee and providing the time and feedback to support my thesis. Dr. Ting, thank you, for your advice and guidance through the last several years. Thanks to Dr. Alirezaee, and his group for their time and for providing me with contacts and resources to assist ROS programming aspects with my research.

I would also like to acknowledge and thank the University of Windsor MAME Graduate Department Staff, Ms. Haskell, and Ms. Jibrail, for assisting and accommodating me through this time.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

This thesis aims to study multiple mirrors and mirror-like object detection for indoor navigation applications using 2D LiDARs. This chapter introduces several important topics relevant to indoor navigation, including autonomous navigation, path planning, localization, and mapping.

1.1. Autonomous Navigation

Autonomous navigation is the ability for mobile objects to find their course to a specific destination without human interference [1]. There are numerous applications of this technology. For example, there is a large amount of research in the area of manned and unmanned aerial vehicles, spacecraft, marine vehicles, agriculture, trains, road vehicles and robotics [2]. Applications can be widespread, from education and transportation to surveillance [3] and customer support.

Strategies for autonomous navigation range from aided through strategies such as teleoperation with some autonomous features to fully autonomous navigation. Some applications of indoor autonomous navigation include domestic robots such as vacuum or mopping robots that use simpler strategies such as turning a certain angle when detecting an obstacle to cover the entire area [4]. These types of strategies, although not necessarily the best way to cover an area, can be used in any type of area. A similar concept can be implemented for maze-solving robots. More strategized navigation technologies use environmental data to make safer and better navigation in circumstances where the surroundings are subject to change (for example, moving objects). These technologies can be used in emergency robots that need to navigate unsafe environments such as fires [5]; in semiautonomous and driverless automobiles in real-world situations [6]; in robots used in industrial or manufacturing environments, etc.

To have a functioning autonomous navigation system, it is required to have hardware and software related to the physical propulsion of the vehicle; the ability to understand the position of

the robot with respect to its environment using information acquired using sensing and perception using various sensors; and the robot should have the ability to navigate itself in the environment using tracking, following a path or creating its own path. This comes with several challenges, including perception and object detection; localization and mapping; and navigation and path planning [7].

Localization is the process of estimating the position of a target in a known or unknown environment [8]. Mapping is the process of constructing a representation of the physical environment around the target. Navigation refers to controlling the course of a concerned object. Path Planning is the process of finding the best path between two points

1.2. Path Planning and Navigation

Path planning is a computational problem to find the shortest or the best path from source to destination that requires the minimum amount of work. Path planning has several problems associated with it, including self-localization and obstacle avoidance [9] [10]. Environmental information can be used to classify path planning methods into static and dynamic methods. Static path planning is used in static environments with static obstacles that do not change their position with respect to time [11], such as walls in indoor environments and buildings in outdoor environments [12]. Static path planning is best used when the map of the environment is known.

The most classic static path planning algorithm is called Dijkstra's algorithm, published in 1959 [13]. This and other algorithms based on it are generally used in the telecommunication domain for telephone networks and navigation applications such as Google Maps [14] [15]. This algorithm uses the vertices and edges of a weighted connected graph to calculate the shortest path between the origin and destination. It does so by creating a web of weighted points from the origin to the destination. The points in the graph can be represented as nodes and edges, nodes as groups of data and edges as links that connect the nodes [16]. Consider an arbitrary node as the origin node and assign it a distance value of zero ($S_O = 0$) with the remaining distance values equal to infinity. Each directly connected node will then be visited (v), providing it with a measured distance $S_{i(v)}$, with all indirectly connected unvisited (u) nodes remaining at infinity ($S_{i(u)} = \infty$). The

distance for all unvisited nodes will be calculated from the origin node through a visited node with only the shortest distance considered. Through this iterative method, the shortest distance is determined. The Dijkstra algorithm is reliable but computationally expensive. There have been several modified and enhanced algorithms based on it [15] [17] [18]. Another algorithm for static path planning is the A* algorithm published shortly after [19], which uses information on the objects in the environment through a grid-based system using a heuristic function (h(n)) that estimates the most cost-effective path from any node (n) to the destination, minimizing the following formula, where g(n) is the cost of the current node from the origin [20].

$$f(n) = h(n) + g(n) \quad (1)$$

Dynamic path planning is a much more complex problem used in dynamic environments where obstacles have the possibility of changing their position and orientation with respect to time [12]. It is important in these environments to have the ability to continuously collect and compute environmental data [21]. The D* method is a classic dynamic path planning algorithm. It maintains an open list of valid locations/states that are used to transfer location and cost information, including the cost of moving between locations in the space [22]. The algorithm has two main functions: the first to compute the best path cost to the destination and the other to enter affected states into the list. The nodes in the open list are iteratively evaluated, and changes are applied to the neighboring states depending on the dynamic environment. In a mobile application, this allows the target to update its map and replan its path to the destination [22]. The D* lite algorithm is similar to this but decreases the computational load needed. The D* value, using the same terms as in A*, can be described by the following formula [23].

$$f(n) = h(n) \quad (2)$$

The environmental information available to the target can be used to classify the navigation and path planning strategies into local and global. Global navigation strategies are used when the environment is known. Local navigation strategies are used when the immediate environmental data can be collected through sensor readings and are used to control the motion of the target to avoid collision with obstacles that may change their position with respect to time [24]. Global and local navigation strategies must be combined for adequate path planning.

There are several navigation approaches, and they can be classified in many ways, including geometric, topological, semantic [25]; deterministic, stochastic, evolutionary [26]; classical, and reactive [27]. The following are the most commonly documented and used navigation strategies in the literature.

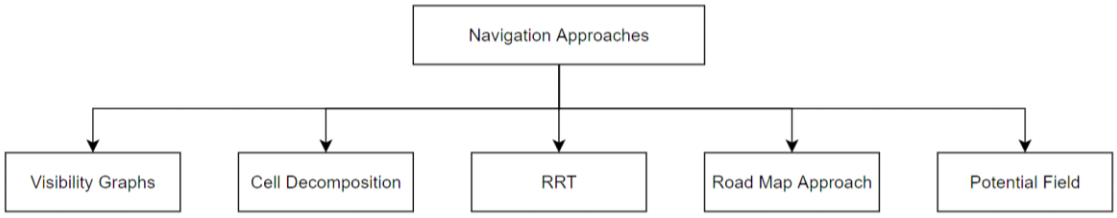

Figure 1: A list of some commonly documented navigation strategies in the literature

The visibility graph method and Voronoi diagram method. Cell decomposition methods are graph search methods [28]. In the first two methods, a graph is constructed, and then a path is computed. The configuration space including all obstacles is divided into edges and vertices. The start and destination are vertices, and all connected vertices are joined as the shortest distance between any two vertices. The Voronoi diagram method improves performance by removing unnecessary turns. In cell decomposition, the configuration space is divided into cells and is classified depending on their level of occupancy, after which connectivity graphs are created [27, 28].

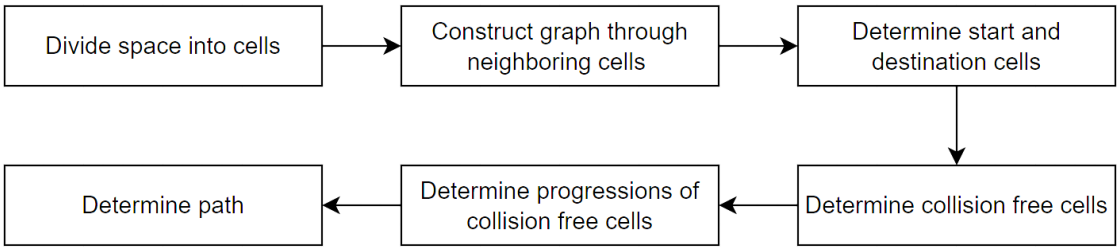

Figure 2: Outline of the process of navigation through cell decomposition

The road map approach uses the visibility graph method and the Voronoi diagram method to develop a set of curves also known as a roadmap [27] that could span the entirety of the free space, allowing for the solving of path planning problems.

The potential field approach uses the idea that the entire environment is filled with an artificial potential field, where the destination attracts the target, and the obstacles repel the target at a potential that is inversely proportional to the distance [27]. This approach can be relayed as in the following equation, where 'U' is the potential force.

$$U = U_{attract} + U_{repel} \tag{3}$$

The Rapidly-Exploring Random Tree (RRT) algorithm [29] is one of the most efficient path planning algorithms and works for high-dimensional dynamic environments [12]. This algorithm builds an expanding tree of points from the origin into the unexplored space [30] to the destination, increasing the node density to find the shortest path.

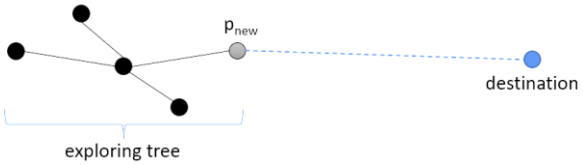

Figure 3: Pictorial representation of Rapidly Exploring Rapid Trees where a tree of points expands to into unexplored space in search of the destination.

Modifications and combinations of these traditional path planning and navigation methods are now commonly investigated in the literature [31] [32] [33] [34] [35].

1.3. Navigation and Localization

Navigation can be defined as orienting a target object such that it would be able to control its course through a predetermined path or a route to the desired destination [36]. Navigation can also be further classified on the basis of the environment, as indoor and outdoor. Navigation techniques generally have different localization strategies associated with them. Localization or positioning allows the body to understand its position with respect to the environment using data collected through sensors. It is in and allows it to have the information required to perform further activities such as noise cancellation, mapping, and path planning.

In outdoor environments, straightforward localization and navigation methods can be used. Global Navigation Satellite Systems (GNSS) is a navigation technology that uses satellite networks. Receivers measure transmitted signals with orbit and position-related information that are broadcasted by satellites and use it to estimate position [37]. There are several positioning technologies that are encompassed by GNSS, including Global Positioning System (GPS), which is generally used by the US, and Galileo, which is used in the EU, among several others, on global and regional scales.

The positioning technique used by GNSS is trilateration. Here, the known positions of multiple satellites are used. Considering each satellite, a point in a cartesian space, they will each produce a spherical locus of position [38]. The position of the receiver at the target can then be estimated by solving for the intersection of the position loci. As the receiver moves, the distance from each satellite will change, allowing for a new position estimate.

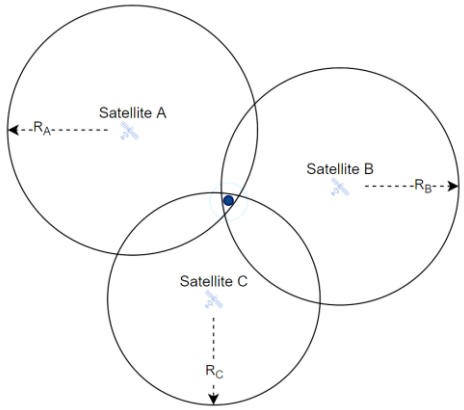

Figure 4: A pictorial representation of how GNSS uses trilateration for localization

A similar positioning technique is triangulation, which uses a congruency of triangles to estimate the position of the target and is most commonly used in surveying applications.

Outdoor applications generally use a combination of GNSS and INS (inertial navigation system) to increase position estimation accuracy [39] [40]. INS uses various sensor data to compute information such as position, velocity and attitude. The sensor data are collected by an inertial measurement unit (IMU). This unit is generally a microelectromechanical (MEMS) device consisting of some combination of gyroscopes, accelerometers and magnetometers; most commonly, three gyroscopes and three accelerometers. Gyroscopes measure rotation and work on

the principle of the conservation of angular momentum: L is angular momentum, I is the moment of inertia, and ω is angular speed.

$$L = I \times \omega \tag{4}$$

There are now several types of gyroscopes. The most common are Gimballed, Vibrating Gyroscopes, MEMS-based gyroscopes, Fiber Optic Gyroscopes (FOG) and Ring Laser Gyroscopes (RLG), among others. A MEMS gyro generally contains a mounted proof mass made of silicone and uses Coriolis force to measure rotation. FOGs and RLGs generally use laser beams traveling in opposite directions and use the Sagnac effect to measure rotation.

Accelerometers are used to measure linear movement by measuring the forces acting on a system. They generally work as a spring-mass-damper system and can be calculated as a sum of the inertial, damping and spring forces, where m is the mass, c is the damping coefficient, k is the spring coefficient, and x is the position.

$$F = m\ddot{x} + c\dot{x} + kx \tag{5}$$

Magnetometers are sensors that are used to measure magnetic induction. In outdoor applications, they can be used to measure the intensity and direction of the magnetic field (for example, the Earth's magnetic field) that the target is passing through. They are generally used in high-performance inertial navigation systems.

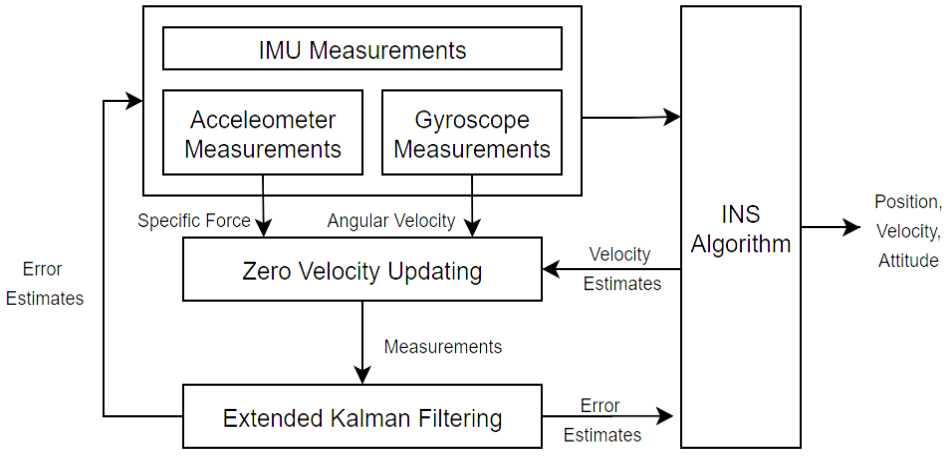

Figure 5: A simplified algorithm for an inertial navigation system

INS can be categorized into two main types – Gimballed systems and Strapdown systems [41]. Gimballed systems are generally used in areal and marine applications. These suspended gyrostabilized systems maintain the vehicle's equilibrium in unstable environments by allowing the vehicle to rotate around it, measuring yaw, pitch and roll at the gimbals [42]. Strapdown systems generally have the IMU rigidly attached to the vehicle, which is assumed to travel in a single plane of motion [43]. Strapdown applications are generally used on road applications as well as indoor and pedestrian applications [44].

INS is generally closely linked to dead reckoning. This method is a more straightforward localization method that can be used when the starting position and orientation of the vehicle or target are known. Dead reckoning uses the starting position and known velocity to approximate the new position based on the time taken at that velocity to the newly updated position. It has several applications; for example, it can be used for short ranges when the GPS signal is lost while driving through a tunnel [39] [45].

It can also be used for pedestrian navigation [44] [46]. Dead reckoning and INS are prone to errors and use concepts such as zero velocity updating for error correction [44], where the algorithm checks for zero velocity and resets the positions. INS generally will also use algorithms for state estimation, such as a variation of a Kalman filter [44] [45]. These algorithms will help the system estimate the bias and minimize error.

Navigation techniques are also used for indoor applications or areas where GNSSs cannot be used, such as parking structures, airports, office buildings, etc. For example, applications could include indoor drones, mobile robots for vacuuming or mopping and pedestrian navigations using mobile phones.

As previously stated, navigation methods generally need localization strategies associated with them. Localization strategies can be classified on several bases. Classification can be performed based on range-based positioning techniques for these applications. It can be classified based on the signal property [47] [48] and broadly into angle-based, distance-based and signal-based.

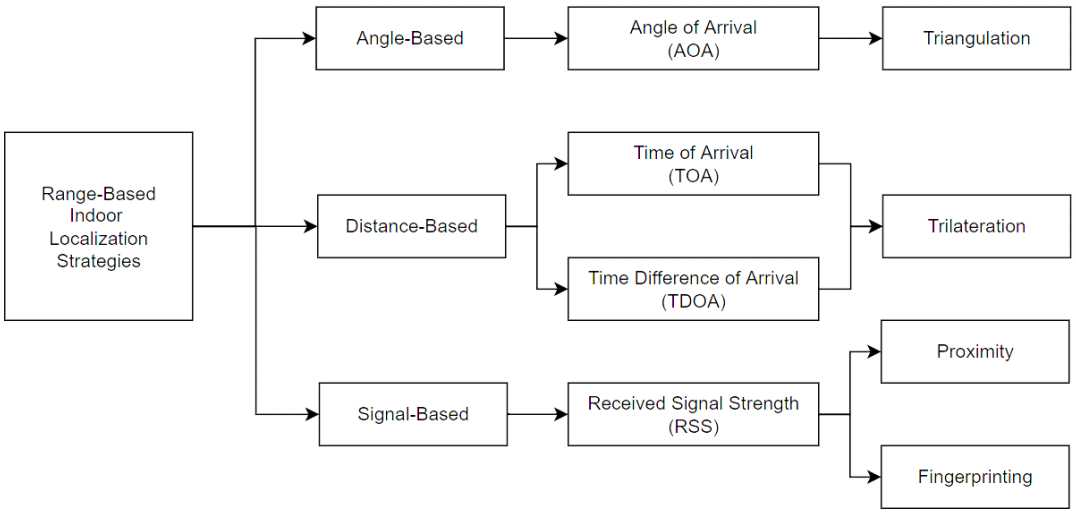

Figure 6: Classification of range-based localization based on three characteristics: angle-based, ranging-based and signal-based

The time of arrival (ToA) and time difference of arrival (TdoA) are distance-based localization techniques. They both use a time of flight (ToF) algorithm, which calculates the time taken for a measured signal to travel between the transmitter and the receiver. Trilateration, which was touched upon earlier for its use in GNSS, uses this type of localization algorithm and can also be used for indoor applications.

$$d = \frac{c}{4\pi f}\varphi \tag{6}$$

While solving for distance, d; c is the speed of light at 3 x 10^8 m/s, and f is the modulation frequency of the signal [48] TOA generally uses the data from the node and the source and requires some synchronization between the two. TDOA uses a pair of nodes and restricts the synchronization between nodes to be high; hence, it does not need the node and the source to be synchronized. Due to this, it is more efficient and less prone to errors compared to TOA [49]. The angle of arrival (AOA) generally uses angle information from two or more nodes to draw intersecting lines to understand position information. There is generally a high computational cost associated with AOA algorithms. Triangulation generally uses AOA by using triangle properties and a few reference points.

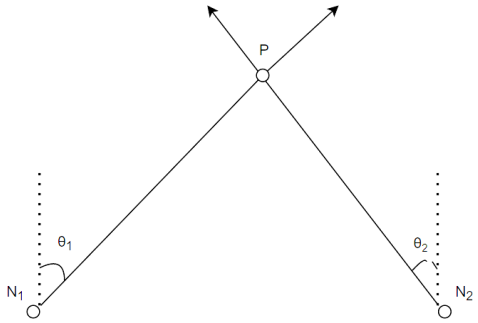

Figure 7: This figure shows a pictorial representation of using triangulation for localization using the angle of arrival (AOA)

Received signal strength (RSS) generally uses Wifi- or Bluetooth-enabled environments and uses the power of the signal received by the receiver to determine the position of the target considering the attenuation losses of the signal. where P is the reference power, which is the power of the signal in dB at one meter from the transmitter; d is the distance measured between the receiver and the transmitter; and α is the exponent of loss of power with respect to the path γ [49] [48].

$$\mathrm{RSS} = \mathrm{P} - 10\alpha \log_{10} \mathrm{d} - \gamma \qquad (7)$$

RSS is generally low in computational cost but is high in error accumulation. Two positioning techniques that commonly use RSS are proximity algorithms and fingerprinting. Proximity algorithms use a grid of nodes with known locations in the environment to deduce the connectivity [36] [48]. Similarly, fingerprinting can be used in a WiFi- or RF-enabled environment and uses a network of RSS values to determine the position of the target [36].

1.4. Localization and Mapping

Mapping is the process of building a virtual map of the physical environment around the target. Mapping is generally done with no prior knowledge of the location and can be used for navigation. Simultaneous localization and mapping (SLAM) is the representation of the spatial

environment (mapping) while keeping track of position (localization) within the built map. The SLAM framework involves odometry, landmark prediction, landmark extraction, data association and matching, pose estimation, and continuous map updating. Conceptually, the location of the target can be simply determined based on prior knowledge of the position of landmarks in a global map and range and bearing measurement from the robot to the landmarks [50]. However, in practical implementation, this process is not trivial. Many issues must be overcome to obtain reliable and accurate localizations, such as identifying good landmarks, data association, effective calculation processes and many others [51].

Typical technologies that have been used for indoor mobile robot localization include infrared (IR), light detection and ranging (LIDAR), radio detection and ranging (RADAR), radio frequency (RF), sound navigation and ranging (SONAR) and visual sensors [36]. These technologies need to be able to deal with noisy observations and generate not only an estimate of the robot's location but also a measure of the uncertainty of the location estimate. SLAM technologies are explicitly used to collect data to create a map.

Most SLAM frameworks use probabilistic localization to build a map of the environment around the target. When a target changes position in an environment, probabilistic localization uses the information from all the previous position estimates along the path taken to approximate the target's current position [52]. This concept is Markov's assumption. In this process, they continuously assign probabilities to possible positions the target could move to.

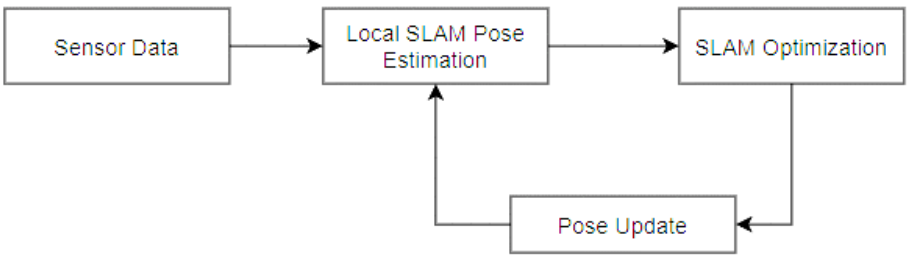

Figure 8: A simplified algorithm of simultaneous localization and mapping

The Bayesian approach is a basic method to deal with conditional probability; more precisely, it relates the conditional probability of more than two events. Bayesian filtering is used

to estimate the state of a dynamic target given noisy observations. Considering Markov's assumption, at each point in time, t, u_t is the motion command to the target; x_t is the state of the target; and z_t is the measurement taken at the current state. Using the Bayes theorem, the probability distribution over the state, Bel(x_t), can be illustrated as follows:

$$Bel(x_t) = \eta P(z_t|x_t) \int P(x_t|u_{t-1}, x_{t-1}) Bel(x_{t-1}) dx_{t-1} \quad (8)$$

State estimation methods are used to ascertain the state of an anticipated system that is continuously changing, given some observations or measurements. Bayesian filtering implementations can be categorized based on belief distributions into discrete and continuous. (Probabilistic representations are included in Appendix A)

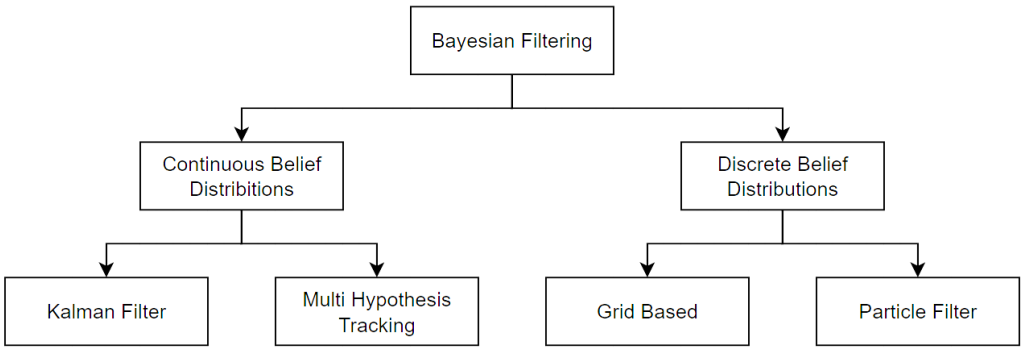

Figure 9: A classification of localization techniques using a Bayes filter

For continuous Bayes filters, belief distributions are generally represented as Gaussian curves. The linear Kalman filter method is a commonly adapted method of filtering and can remove redundancy in the system and predict the system's state. The Kalman filter is an algorithm that estimates the state of a discrete time-controlled process described by the linear stochastic equation.

The Kalman filter has a similar representation to a unimodal Gaussian model, where given a Gaussian, μ_t is the mean at a given time t, and $\sum_t$ is the uncertainty at the state characterized by a dxd covariance matrix [53]. Here, belief probability can be represented as follows:

$$Bel(x_t) = \frac{1}{(2\pi)^{\frac{d}{2}}|\sum_t|^{\frac{1}{2}}} e^{\left[-\frac{1}{2}(x_t-\mu_t)^T \sum_t^{-1}(x_t-\mu_t)\right]} = P(x_t;\mu_t;\sum_t) \quad (9)$$

Kalman filtering is advantageous because it has a lower computational cost. This is mainly because they are only considered uncertainty for unimodel Gaussian distribution preventing Kalman filtering from being adequate for global localization problems [53] [54].

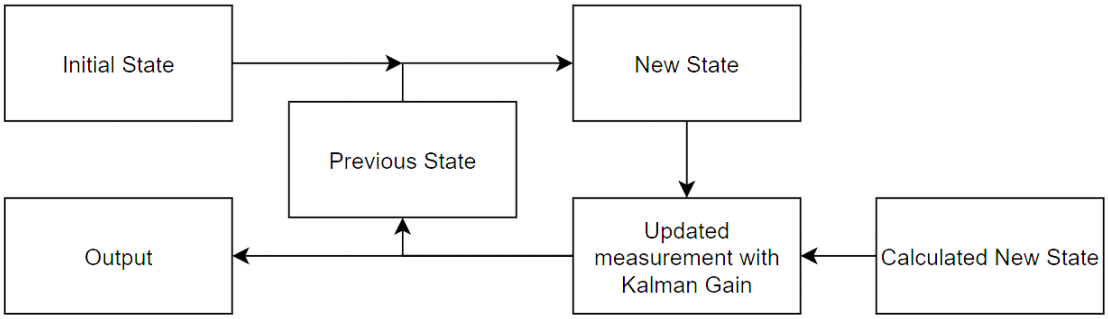

Figure 10: A simplified algorithm of Kalman Filtering used for position estimation

Extended Kalman filtering (EKF) is an extension of Kalman filtering. EFK accounts for nonlinear states by estimating nonlinear through linearization by approximating the mean to predict covariance and compute Kalman gain. Other enhancements to Kalman filtering include unscented Kalman filtering (UKF) [54]. UKF provides slightly better results than EKF by considering several sigma points in its approximation of nonlinear systems [55]. Limitations related to Kalman filtering can be mitigated through multihypothesis tracking (MHT). MHT represents the probability belief as a mixture of Gaussian models, where each Gaussian can be solved by EKF [53] [54]. Weights $w_t^{(i)}$ are set proportional to the likelihood of the measurements.

$$Bel(x_t) = \sum w_t^{(i)} P(x_t^{(i)};\mu_t^{(i)};\sum_t^{(i)}) \quad (10)$$

In a discrete belief distribution, the belief distribution is often represented as a finite set of possible values. Grid-based approaches perform indoor localization by dividing the environment into small areas and reinforcing the belief probability that the target lies within the region. These approaches are advantageous because they have good noise reduction capabilities and can provide accurate results. However, this comes at a high computational cost.

The particle filter (PF) method can approximate probability density functions. The sequential Monte Carlo (SMC) technique uses the sequential importance sampling (SIS) algorithm, including a resampling step at each instant. This method builds the consequent density function using several random samples called particles. Particles are propagated over time with the integration of sampling and resampling steps. At each iteration, the sampling step is employed to reject some particles, increasing the significance of regions with advanced posterior probability. The belief is represented by the following sets:

$$Bel(x_t) = \left\{ \left(w_t^{(i)}; x_t^{(i)} \right) \middle| \, i = 1, \dots, n \right\} \tag{11}$$

Although Kalman filtering is one of the most efficient localization methods due to its low computational cost, particle filtering methods are more reliable than Kalman filtering methods because they can be used in multimodal and dynamic environments without prior information.

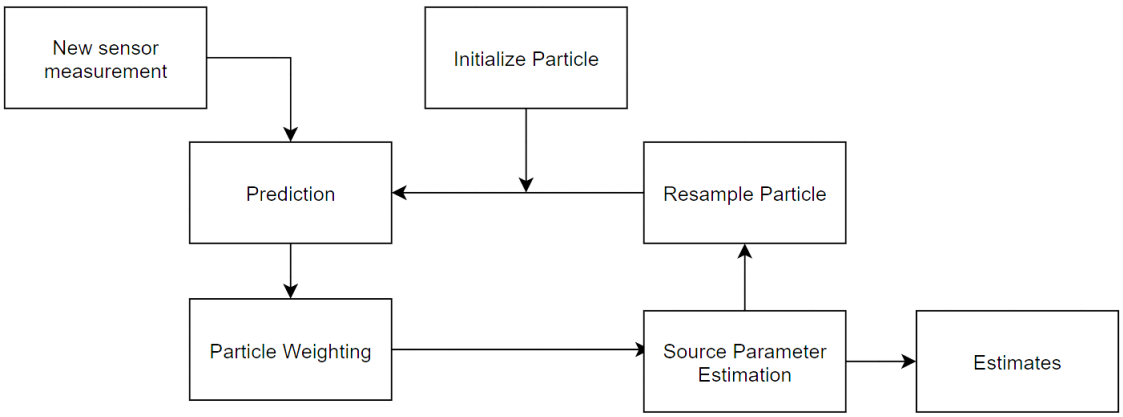

Figure 11: A simplified algorithm for particle filtering used for position estimation

These filters can be used to solve probabilistic SLAM problems. Some examples of this include HectorSLAM, FastSLAM, GraphSLAM, EKF-SLAM, Cartographer and GMapping, among others.

Mapping generally applies some type of odometry. Odometry is the concept of retrieving state or position, orientation, and the change in position over time using sensor data and wheel motions from an encoder [50]. Here, the position can be incrementally calculated through the distance rolled by the wheels of the robot: Distance (dw) traveled is proportional to the wheel

radius (rw) and angle of rotation (φ). If rotation sensors are available on both wheels, the odometrical reading can be calculated from the center of the robot and calculated accordingly. As error generally accumulates over time, encoder data can be coupled with other sensor data to provide more accurate results.

$$\Delta d_w = r_w \Delta \varphi \tag{12}$$

To create a representation of the environment, occupancy grid maps provide a discretized representation of an environment where each of the grid cells is classified into two categories: occupied or free. Considering each cell to be a binary variable, the map will be able to estimate the location of an obstacle in the space by computing a posterior approximation for any given cell within the range of the sensor collecting data [56].

SLAM frameworks can also be classified based on the sensor technology used for data acquisition, for example, acoustic and visual SLAM. Ranging sensors such as LiDAR and SONAR sensors containing only one scan plane can be used. They are beneficial, as they work well in a wide range of weather and ambient conditions. Cameras can also be used to record the image and the depth of information in every pixel, hence having the advantage of both vision and ranging sensors.

A laser range finder or LiDAR is such a sensor. These sensors consist of a laser beam that rotates at a relatively high speed on the order of tens of revolutions per second and measures the distance to the obstacle that reflects it. If there is no obstacle within the sensor range, a reflection is not received, and the sensor typically is at a nominal maximum distance. Given an estimate of the robot location and the grid map, the expected value of a range measurement can be numerically obtained using ray casting. This makes it possible to evaluate the likelihood of a given pose, which is sufficient in some localization approaches, such as a particle filter.

CHAPTER 2

Objectives and Motivation

In the previous chapter, topics related to indoor navigation were introduced. In this chapter, the objectives and motivation behind the problem statement to characterize the behavior of 2D LiDARs in mirrored environments are discussed. Information about the effects of light reflection on various surfaces, the operation of laser-based sensors, limitations, and a summary of the literature addressing some optical challenges are included in the section.

2.1. LiDARs for Mapping

LiDAR (light detection and ranging) sensors are ranging sensors similar to SONAR (sound detection and ranging) and RADAR (radio detection and ranging). LiDARs work on the principle of light. A LiDAR uses laser light to measure the distance and reflective properties of the environment. This light is most commonly infrared but can also be in the visible or ultraviolet range of the spectrum. The LiDAR generally transmits a laser beam and is able to detect a barrier by using a sensor to catch the reflected beam.

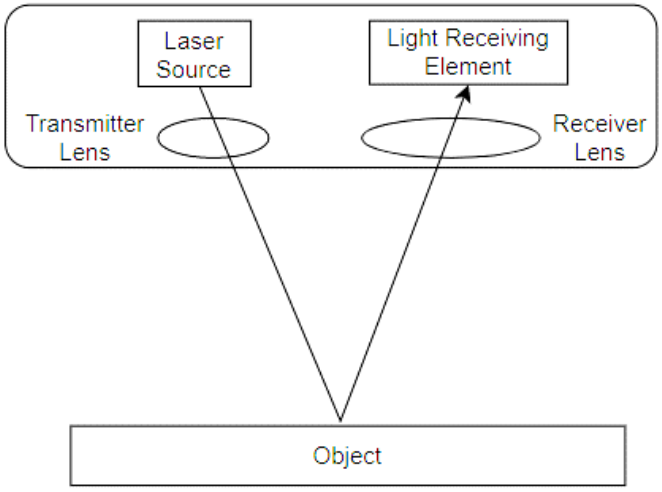

Figure 12: A schematic of the working principle of a laser-based sensor

$$distance = \frac{time \times speed\ of\ light}{2} \quad (13)$$

The distance between the obstacle detected and the receiver and the target is calculated using time-of-flight (ToF). This calculates the distance using the amount of time taken for the emitted pulse to reach the object, reflects some portion of the emitted ray and returns to the receiver lens of the LiDAR, considering the emitted and reflected rays to be traveling at 3 x 10^8 m/s.

LiDARs can take two or three-dimensional scans. A 2D LiDAR will generally spin around an axis, repeatedly emitting a single beam and taking 360° angle scans of the surroundings in a single plane. A 3D LiDAR will emit multiple beams while spinning around an axis, allowing it to take more detailed three-dimensional scans of the surrounding environment. These LiDARs used different types of beams for different outdoor applications. Topographical and terrestrial LiDARs are infrared beams for the mapping of land; bathymetric LiDARs are used for marine applications and green light.

Both 2D and 3D LiDARs have several applications; however, due to the nature of data collected by 3D LiDARs, they are generally used in outdoor environments, while 2D LiDARs are more commonly used in indoor applications. 3D LiDARs are bulkier, priced significantly higher and computationally expensive compared to 2D LiDARs.

LiDARs are advantageous in industry, as the data collected can be converted into 2D or 3D point clouds and easily integrated with other sensor data. Multiple consecutive LiDAR readings allow for complex applications such as obstacle detection, localization and mapping. The width of the emitted beam can be made smaller, increasing the distance it can travel and allowing LiDARs to be capable of detecting obstacles over long ranges. Depending on the nature of the emitted beam, measurements could potentially be less sensitive to ambient conditions and could be used in a wide range of illuminations. Based on the application, LiDARs could be a cost-effective solution.

There are also limitations associated with LiDARs. For small-scale applications, high operating costs are incurred. External sources that change visual properties will affect LiDAR data. Adverse weather effects, such as low-hanging clouds and heavy rain or fog, can have a negative

impact on LiDAR data collection. One study was performed on ten different LiDARs in different weather conditions, including simulated fog, rain and intense sunlight [57]. The findings determined that in fog, obstacles were only partially visible due to reflected light being scattered and attenuated by fog. Rain was detected as solid vertical pillars, while intense sunlight caused the experimental targets to not be visible through LiDAR data [57]. This can be explained by understanding the surface properties.

Surface properties affect the way that reflected light will be scattered, which in turn will affect the way that LiDAR will receive light information. There are four key ways that surface properties affect incident light: specular reflection, diffuse reflection, absorption and transmission.

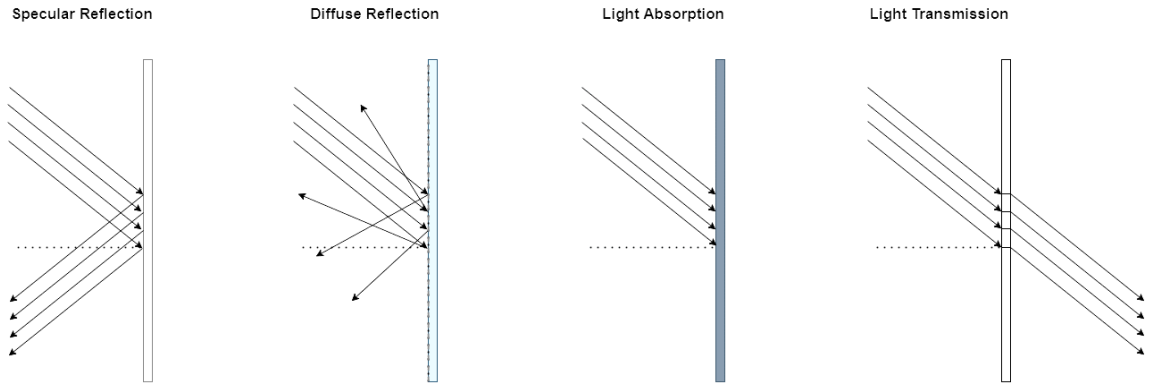

Figure 13: Behavior of Light on various surfaces for Specular and Diffuse Reflection, Light Absorption and Light Transmission

Specular reflection occurs on very smooth surfaces where light is reflected at predictable and consistent angles as it moves along the surface, which creates a mirror effect. Diffuse reflection occurs on rough surfaces due to inconsistent angles of reflection as the surface moves along the rough surface. Different material properties and surface characteristics have an impact on the absorption and transmission of light on or through a surface. A dark black wall would absorb most of the light as it meets the surface, whereas a glass wall would allow most of the light coming in contact with the wall to transmit through the surface. Most surfaces allow for a combination of reflective behaviors. Diffuse objects can be detected most accurately through LiDAR sensors. Some examples of how different surfaces allow for light reflection are listed below.

Table 1: Reflective behaviors for surface properties

Surface	Example	Primary Light Characteristic	Secondary Light Characteristic
Reflective Surface	Mirror	Specular	Absorption, Diffuse
Diffuse Surface	Concrete	Diffuse	Absorption, Specular
Transparent Surface	Glass	Transmission	Specular, Absorption, Diffuse
Dark Surface	Black	Absorption, Diffuse	Specular
Light Surface	White	Specular, Diffuse	Absorption

2.2. Problem Statement

As seen in the previous sections, LiDAR sensors work on the principle of light. Therefore, external sources that change visual properties will affect LiDAR data. As seen in table 1, specular and transparent surfaces that allow for large amounts of specular reflection and light transmission will distort LiDAR data. In the problem of simultaneous localization and mapping for autonomous navigation, current occupancy grid mapping algorithms assume that objects in the environment are detectable from all angles.

The objective of this thesis is -

- To study and characterize the obstacle detection problem for reflective/mirror-like objects using a 2D LiDAR for indoor navigation applications through running experiments in multimirror environments
- To propose a potential method for mirror-like obstacle detection using data collected by a 2D LiDAR while navigating through a multiple-mirror environment.

2.3. Summary of Literature

There have been several attempts to solve the problem of specular and transparent object detection using light emitting sensors such as laser range finders and LiDAR sensors. The oldest and most common attempts to solve for this problem are through sensor fusion. Sensor or data

fusion is the combination of data from various sensors to minimize error in retrieved information compared to retrieving information from individual sources.

Diosi and Kleeman [58] used a sensor fusion strategy. They combined the sensor data of a sonar sensor and a laser. Sonar sensors are ranging sensors that pulse soundwaves and use the reflected echo to compute distance. Both sensors are used to increase the accuracy of their measurements. All surfaces are tracked by taking sonar readings six seconds before and after each laser scan, and data segmentation is used to create map features for SLAM implementation. Kalman filter-based sensor fusion was applied to the data collected by both sensors. Since ultrasonic sensors are not affected by light properties, all able objects that were not diffuse objects were able to be tracked while also minimizing the overall error of the data they collected.

Yang and Wang [59] also detailed sensor fusion in attempts to deal with laser scan failure. An ultrasonic sensor was used to create two separate occupancy grid maps accounting for all types of surfaces. Assuming that mirrors are planar, the property of mirror symmetry is used to restore missing scan data, and an iterative closest point (ICP) algorithm is used for mirror prediction. Extended Kalman filtering is used to integrate mirror predictions at different time steps.

In an extension of this research [60], an attempt was made to solve the mirror reflection problem using only LiDAR data. Assumptions were made based on detecting mirrors that were planar as well as framed. Data association is performed, and a Bayesian filter is used to estimate the endpoints of the mirror by detecting the change in properties while the LiDAR moves from a framed surface to a reflective surface and is related to mirror symmetry. They used this concept and integrated a Bayesian filter into an occupancy grid map algorithm.

Peasley and Birchfield [61] developed an algorithm to detect specular objects resting on the floor using an IR camera as part of an obstacle detection algorithm. This algorithm consists of acquiring a point cloud, using segmentation to remove the ground plane from the point cloud and using the points to create an occupancy map to use. The approach to detect specular objects considers all pixels with an invalid depth reading. Using 8 neighbors, all adjacent pixels are examined, and if any point is a floor point, a vertical line is assumed at that floor point. Using this technique, they were able to create boundaries around standing specular objects.

Foster et al [62] developed an algorithm to detect transparent surfaces. It was found that when light travels through a transparent surface, there exists a subset of diffuse-like angles approximately 2 degrees from normal. An algorithm was developed to sweep through the widest range of angles at every cell of the occupancy grid map and match it to the LiDAR sweeps. Matched cells create a subset to map all surfaces now visible from any angle.

Koch et al. [63] set up a prefiltering module that detected discrepancies in reflective scans of a 3D multiecho LiDAR by searching for mismatches in the measurements between corresponding echos. At these mismatches, the points behind the potential obstacle are also considered. The outliers identified were fitted using an ICP algorithm and classified into the type of surface. After the surface was identified, the points behind the surface were removed to provide a clean representation of the environment.

This research was furthered [64] to include a postfiltering algorithm that could be used to identify and differentiate between different types of surfaces. Multiple experiments were carried out on a shiny surface (aluminum), glass and mirrors. The two echoes of the surface and the background of the surface were compared. For aluminum, it was found that the intensity of the first echo was greater than the second echo. For glass, it was found that the background in the second echo impacted the resulting curves more than the first echo, which showed low intensity. Echoes on mirrored surfaces showed a combination of both behaviors. Experiments were also performed on other transparent, translucent, and reflective objects, and it was concluded that it would be possible to map but would require a special algorithm for each type of surface.

Zhao et al. [65] studied the reflection problem of glass by analyzing point cloud 3D LiDAR for mapping. Observations were made using a lidar with three modes of beam pulse processing, comparing diffuse objects, glass intensity and diffuse objects placed behind glass. It was observed here that perpendicular to the glass, the lidar will return the strongest intensity. This problem was addressed by processing each ask by detecting the intensity peak and dual return and identifying boundaries before integrating into a SLAM framework.

Tibebu et al. [66] also studied the reflection problem of glass with no frames using LiDAR mounted on a mobile robot. They also made observations taking into consideration glass, direct diffuse obstacles, diffuse obstacles behind glass, free space and area outside the field of vision of

the robot. An approach for solving this problem was proposed using range data comparisons between neighboring point clouds. Using this information, the difference in standard deviation of the data, change in distance and intensity is used to create a filter to estimate the width profile of glass before incorporation into a map.

Summarizing the literature review, it can be seen that there has been extensive research done on the mirror detection problem through solution approaches such as sensor fusion of 2D LiDARs and ultrasonic sensors to protect unpredictable light properties [58] [59]. Sensors that work on the principle of light were used for the detection and mapping of transparent obstacles [62] [65] [66] and specular objects resting on the ground [61]. Mirror detection for 2D mirrors has been conducted under the assumption that mirrors are planar and framed [60]. 3D LiDARs have been used to analyze, detect and map specular surfaces and transparent surfaces and collect intensity vs distance data for multiple surfaces [63] [64]. This derives the research potential that there is no detailed study of mirror location and path direction on LiDAR data. Research done with 3D LiDARs cannot be easily translated to more cost-effective 2D LiDAR applications. No intuitive method exists for applying 2D LiDARs in an environment containing multiple specular and transparent surfaces.

CHAPTER 3

Approach and Implementation

In the previous section, the limitations of LiDARs and how limitations with respect to optical surfaces have been solved in the literature for mapping and navigation are outlined. In this section, the tools used to characterize the effect of mirrors on 2D LiDARs, the hardware used to run experiments and the configuration between the tools and the hardware are outlined.

3.1. Tools

ROS stands for Robot Operating System. Although it has the words in its name, ROS is not an actual operating system; rather, it acts as middleware between the operating system and robot programs. It is an open-source framework that can be used to build and reuse code between robotic applications. ROS has been maintained by the Open-Source Robotics Foundation (OSRF) since 2013 [67]. OSRF also releases a new version of ROS every year. As of 2020, ROS still runs primarily on Unix-based platforms, specifically Ubuntu. However, experimental versions have been released for other operating systems, including MacOS and Windows.

Table 2: ROS Distributions [68]

ROS Distribution	Release Year	Version	Compatible Ubuntu Release
ROS Kinetic Kame	2016	ROS 1	16.04 – Xenail Xerus
ROS Melodic Morenia	2018	ROS 1	18.04 – Bionic Beaver
Dashing Diademata	2019	ROS 2	18.04 – Bionic Beaver
ROS Noetic Ninjemys	2020	ROS 1	20.04 – Focal Fossa
Foxy Fitzroy	2020	ROS 2	20.04 – Focal Fossa

ROS has many versions that have been released over the years. Some of the stable distributions and their compatible ubuntu versions are listed below. For the purpose of this thesis, we will use the ROS Kinetic Kame on Ubuntu 16.04 Xenial Xerus.

The current open-source design philosophy of ROS involves four sections: plumbing, tools, capabilities, and ecosystems [69]. Plumbing refers to process management and communication between processes through a publisher-subscriber messaging infrastructure. This method allows for easy access to device drivers and the construction of complex systems. ROS provides many tools, including graphical user interfaces (GUIs); tools for simulation, such as Gazebo; tools for plotting and visualization, such as RViz and graphical visualization diagnostics; and libraries, language support (C, C++, Python, etc.), and Linux tools, including compilers, debuggers, data loggers, etc. [69] [70]

ROS not only broaden the current industry-standard capabilities but also provide a large range of algorithms for control, planning, perception, navigation, and manipulation, making it popular among research communities. Ecosystems refer to the large community supporting, developing, and collaborating within ROS. ROS is very well documented and organized. All available resources can be found at a single location, including the software distributions, package organization and tutorials [69] [71].

The following concepts are integral in understanding the inter-process communication within ROS.

Nodes - Nodes can be defined as software processes that perform certain functions after registering with an ROS Master. These functions can include processing data, commanding hardware, and executing algorithms. Nodes are usually written in C++ or Python and allow for a modular approach while developing robotic projects [72]. Nodes can be represented as ellipses. Publisher nodes generate information, including sensor data, publish the processed data to a topic and use ROS topics to communicate with other nodes. Subscriber nodes monitor system states. They receive data by subscribing to the information in a topic and use a topic callback to process the information [72].

Service - A ROS Service is defined using two message types: a request message type and a response message type. ROS services generally block the flow of the program and provide event-based execution. They are, hence, useful when designing sequential behaviors. ROS services are made available to other ROS nodes via a service server. Service clients can send requests once they are available [72].

Actions - Actions can be defined using three message types: the goal, which is the request message; the result, which is the response message; and the feedback, which provides continuous feedback regarding processing the goal. An action server allows all nodes to request action goals to be processed, and an active client sends the goal requests to the action server [72].

Topics - Topics can be defined as buses that transport information between nodes. The information passing through a topic is organized in a data structure, which could include multiple datatypes. Each topic is identified by a name and a standard or custom type. The type of topic, however, depends only on the information that the node requires it to carry. Topics are graphically represented as rectangles [72].

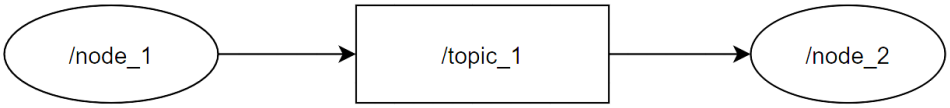

Figure 14: Representation of Communication between ROS nodes and topics

Catkin is a build tool that compiles source files into executable binaries. A catkin workspace is an ROS workspace where catkin is used as a build tool. There are four major sections of the workspace: Build, which is used by the build tool to store intermediate files; Devel, which contains all the setup and executable files for the project; Logs, which contains the debug information; and Src, which contains the code. There are two files that turn a normal file into an ROS package CMakeLists.txt and pakckage.xml, which contain the functionality of the script [72].

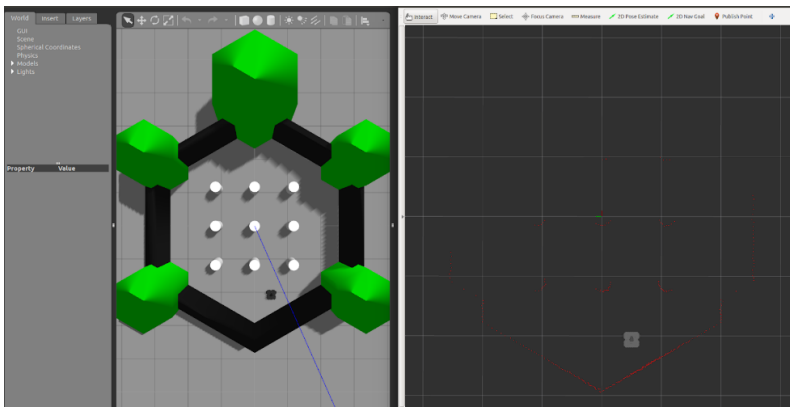

Figure 15: ROS tools for simulation and visualization: Gazebo and RViz

Gazebo is a physics-based robot simulation tool that can be set up with an ROS plugin to simulate projects. It can be thought of as a node that acts as both a publisher and a subscriber. It can also support the integration of several sensors and test them through simulation before real-world implementation [73] [74]. Gazebo provides many robot models and can generate sensor data that can be used by RViz.

RViz is a 3D data visualization tool that is used to analyze robot transforms. It can visualize data from both simulations and real-world robots. It can capture data individually from all sensors present on the robot or robot simulation [73] [75].

3.2. Hardware

The Turtlebot is a standard ROS Platform robot. It is an open-source and low-cost research robot with the capabilities of teleoperation, Simultaneous Localization and Mapping (SLAM), Manipulation and Navigation, Artificial Intelligence and Autonomous research. As of 2020 there have been three versions of the Turtlebot.

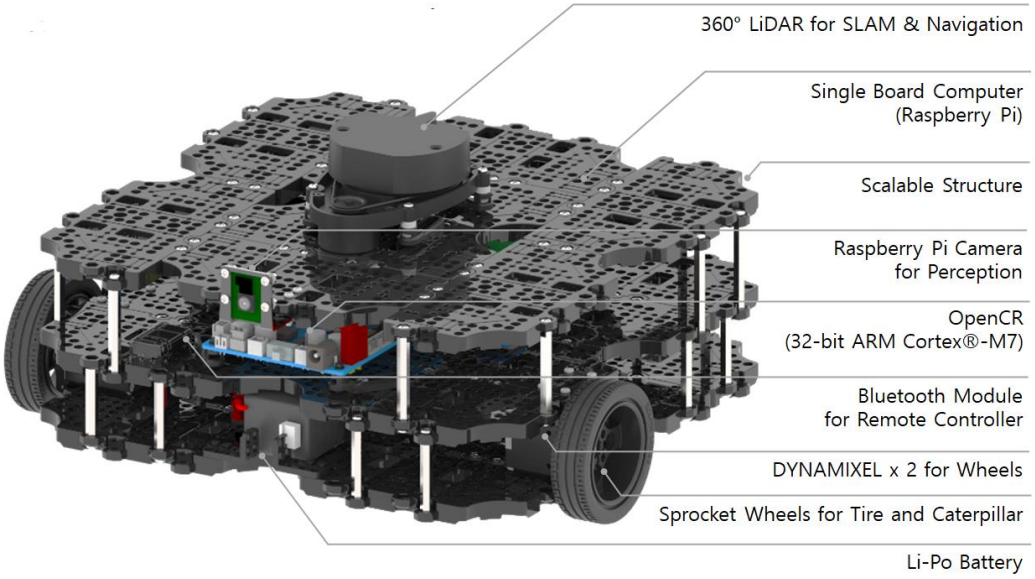

Figure 16: Indoor Mobile Robot - Turtlebot3 Waffle-Pi [76]

Turtlebot1, developed on top of a Roomba-based mobile robot, using an onboard computer, in use since 2011. The Turtlebot2 developed over a Yujin Robot base using an onboard computer

has been in use since 2012. The Turtlebot3 released in 2017 used Dynamixel smart actuators and an onboard SBC (Single Board Computer). It has three variations: burger, waffle and waffle-pi. Turtlebot 4 is now released [77]. Here, a Turtlebot3 Waffle pi is used.

Table 3: Turtlebot3 Specifications [78]

Turtlebot 3 – Waffle Pi: Specifications

SBC	Raspberry Pi 3
Embedded Controller	OpenCR
Sensors	Raspberry Pi 3 Camera
	360 LiDAR
	IMU (3-axis gyroscope, accelerometer, magnetometer)

The Main Controller board is an OpenCR, which is an open-source control module made specifically for ROS applications. It also contains an inbuilt IMU (Inertial Measurement Unit) with a 3-axis accelerometer, gyroscope and magnetometer.

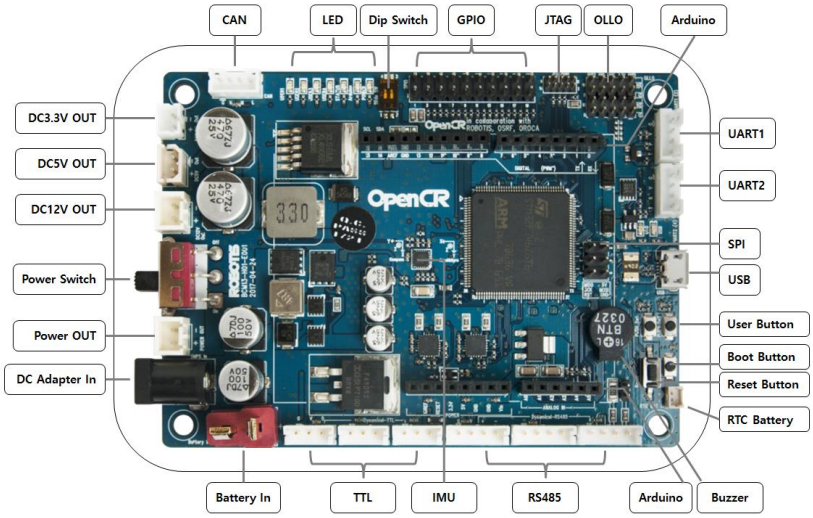

Figure 17: Micrrocontroller - OpenCR [78]

Accelerometer – Sensors that measure linear, nongravitational acceleration (specific force) by detecting a change in capacity due to the displacement of a hinged proof mass or the change in frequency of a vibrating element (principle of a mass-damper system)

Gyroscope – Sensors that measure the rate of rotation of an object by sensing the Coriolis force induced by rotation. This causes energy transfer to the sense mode proportional to the angular input (principle of a mass damper system)

Magnetometer – Sensors that measure the direction, strength, or relative change of a magnetic field at a particular location

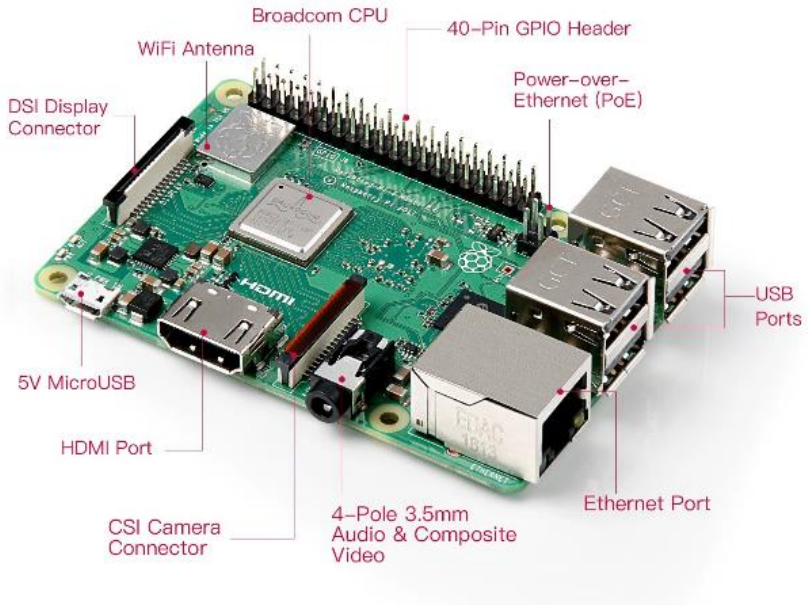

Figure 18: SBC - Raspberry Pi 3 [79]

The SBC present on Turtlebot3 is a Raspberry Pi 3. The Waffle Pi also contains a 360° LiDAR and a Raspberry Pi camera. The raspberry pi has Quad Core 1.2 GHz Broadcom BCM2837 64-bit CPU; 1 GB RAM, BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board; 100 Base Ethernet; 40-pin extended GPIO; composite video port; CSI camera port for connecting a Raspberry Pi camera; DSI display port for connecting a touchscreen display; a Micro SD port for loading the operating system and storing data and Micro USB power source up to 2.5A [79].

Figure 19: Raspberry Pi Camera [79]

LiDAR - LiDAR uses laser light (typically infrared light) to measure the distance and reflective properties of the environment. It is also less sensitive to ambient conditions and hence can work in poor illumination. 2D LiDAR sensors (2D laser scanners as well) are suitable for performing detection and ranging tasks on surfaces. Regardless of the angle of installation, it can be used both indoors and outdoors.

Figure 20: 2D Laser Scanner - LDS – 01 [80]

For navigation, detection, or measurement: 2D LiDAR sensors supply reliable measurement data for a whole host of tasks. The LDS-01 is a 2D laser scanner capable of sensing 360 degrees that collects a set of data around the robot to use for SLAM (Simultaneous Localization and Mapping

Table 4: LDS Specifications [80]

LDS Specifications	
Detection Distance	120 ~ 3,500 mm
Distance Precision	± 15 mm ± 5.0%
Distance Accuracy	± 10 mm ± 3.5%
Scan Rate	300 ± 10 rpm

Turtlebot 3 also has two smart servo motors from the Dynamixel X-series (XM430-W210-T). The servos are at the wheels and have an absolute encoder and use a PID control algorithm. They also have several control modes, including current, velocity, position and voltage (PWM) control [81].

3.3. Setup

The Turtlebot is an ROS platform robot that interfaces as shown in the figure below. As mentioned in the previous sections, the Turtlebot has both an SBC (raspberry pi) and a controller (OpenCR). The microcontroller is used for communication with simpler sensors and actuators. In this case, the OpenCR interacts with the IMU and the Dynamixel Servo motors. This setup is particularly useful for odometry. SBCs are essentially small-scale, fully functioning computers that can run an operating system. This is desirable for use with ROS, as it needs an operating system to run on, so using a raspberry pi allows for ROS to be used directly on the robot.

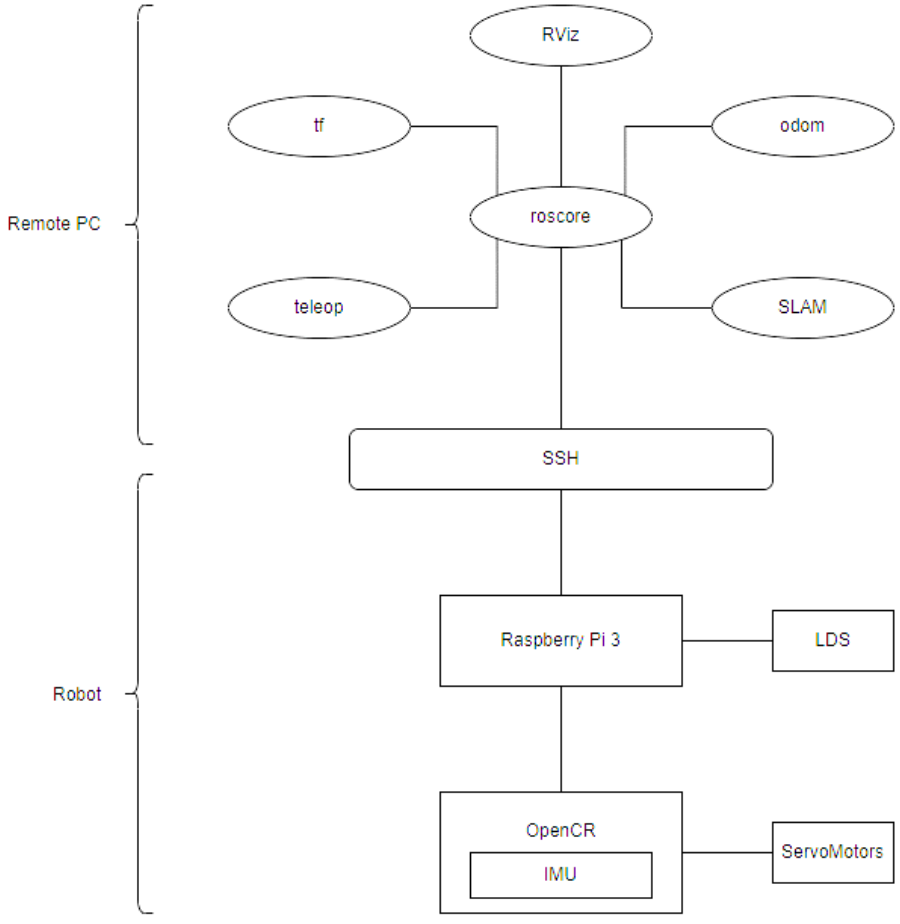

Figure 21: Setup of communication between the mobile robot and remote PC

Secure Shell Protocol (SSH) is used to communicate with the SBC of the mobile robot from a PC using remote access. It uses a client-server architecture to allow communication between the two entities by providing a secure connection over a nonsecure network such as Wi-fi [82]. In this setup, a Linux command used to establish the connection between the remote PC and the

Once the robot is accessible remotely, ROS-related packages can be brought up through roscore. These packages are a collection of programs and nodes, including an ROS Master and ROS parameter server [83]. Enabling the ROS master is a prerequisite to use the system, as it allows nodes to be located and locate other nodes [84]. Roscore is initiated on the remote PC. Before initiating Roscore, all network settings should be configured correctly such that the remote PC is able to communicate with the robot. The package that keeps track of and maintains relationships between different coordinate frames is called tf and is short for the word “transform” [85]. An rqt plug in can be used to visualize the tf relations as a tree using the following command. (An image of this is included in Appendix C)

After ROS packages are brought up, to interface with the turtlebot, the turtlebot packages need to be brought up as well. These packages are provided through Robotis. Teleop, which is short for “teleoperation”, provides the ability to control movement of the robot through the keyboard of the remote PC. With respect to the Turtlebot, this node allows a general range of motion in 4 directions and allows for the robot to move with a range of linear and angular speeds. More information about the setup of the remote PC (master) and the turtlebot (host) is included in Appendix B.

To understand how LiDARs work in multiple-mirror environments, the robot travels through the environment running a SLAM node. The figure above shows the experimental process for the robot to do so. The SLAM node can only run after SSH is set up and roscore is enabled. It then uses IMU and LiDAR data to build a continuous map of the environment. GMapping is a grid-based SLAM algorithm that uses particle filter-based adaptive Monte Carlo localization and local pose estimation to create a grid-based map of the environment.

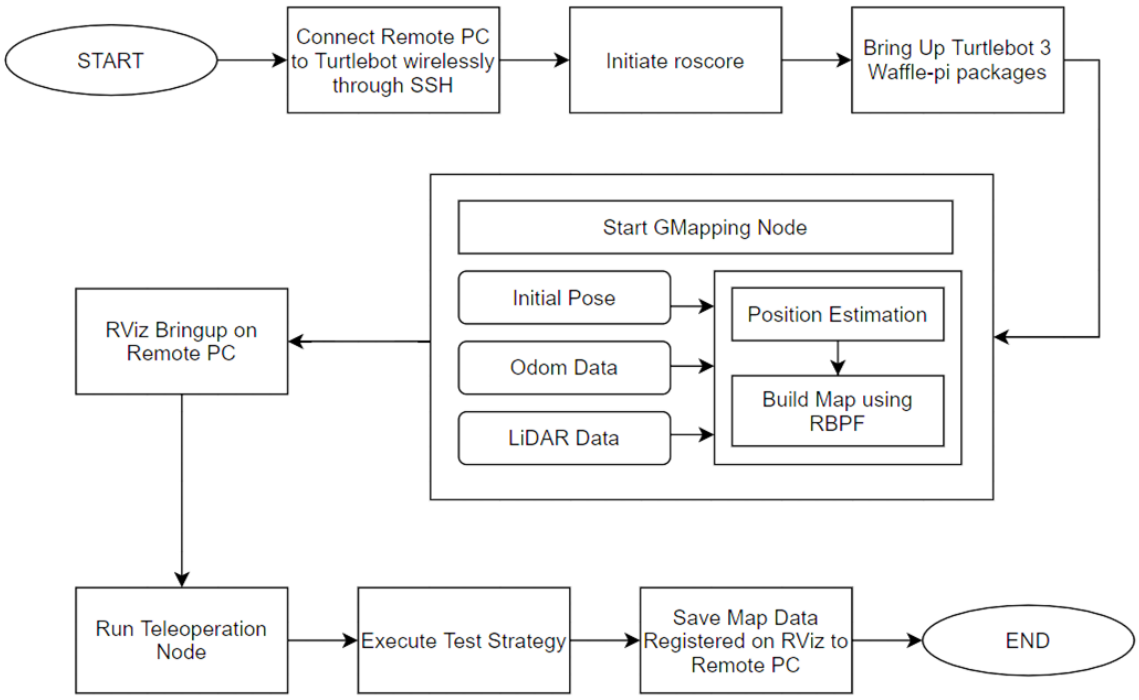

Figure 22: Process of data collection by the mobile robot to build a map using ROS tools

The particle filter used in this algorithm is a Rao Blackwell particle filter. This filter sets the initial belief that in a set of uniform Gaussian distributions, every sample has a weight associated with it. Random weights will decide which states are evaluated, where higher weights are more probable and lower weights are less likely. The joint posterior probability of the position and map will be estimated. Sampling will occur to generate new particles, where weighting will be recalculated and then resampled, and a map update will occur [86].

In Gmapping, the system subscribes to the sensor "msgs/LaserScan" message and publishes the "nav msgs/OccupancyGrid", tf transformation as shown below in the topic. While running a SLAM node, to obtain the most accurate data, the robot should be run slowly to obtain as many laser samples as the LiDAR can pick up in a particular area. It is also not recommended to drive over the same area more than once while mapping as doing so will increase the noise in the map.

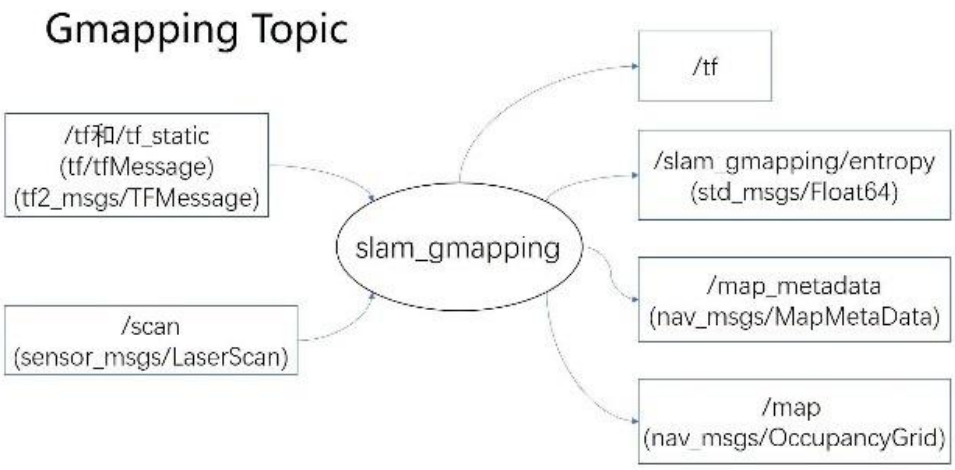

Figure 23: RQT graph of the SLAM - GMapping Topic

To visualize the LiDAR data, Rviz is initiated, and the robot is moved through the system through a teleoperation node. It can alternatively be done through remote control or an object detection and automation strategy. After executing the test strategy, the built map is saved for further analysis. To test the functioning of the node along with mirror detection problems, the robot was placed in a new environment with irregularly shaped boundaries along with one obstacle and one mirror, as seen in figure 24.

Figure 24: SLAM behavioral-check environment

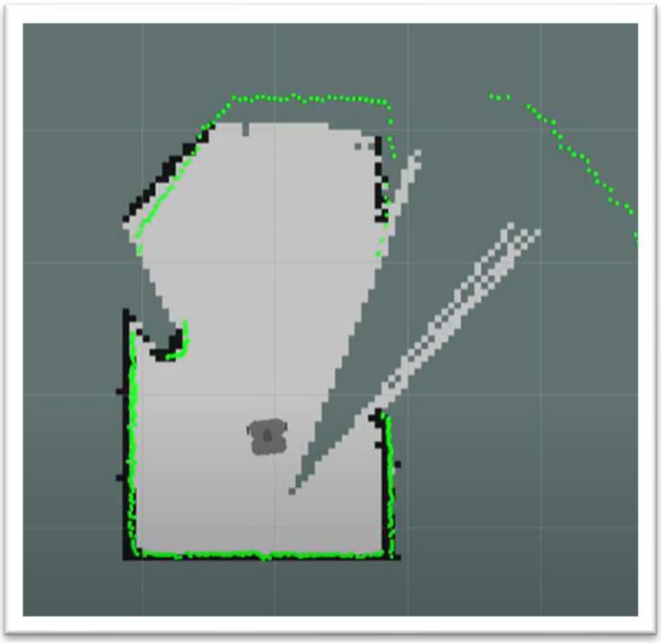

Figure 25: RViz Visualization for SLAM behavioral-check environment

As seen in figure 25, the robot is able to detect all the boundaries and the obstacle. It is also able to estimate its own position in the space while building a map of the environment. Here, it can be seen that the mirror placed on the right side of the robot provides irregular readings and creates an inconsistent map of the boundary.

CHAPTER 4

Experimentation and characterization

In the previous section, the hardware for the mobile robot, ROS and related tools such as Gazebo and RViz, the setup used to facilitate the communication, and the process of data collection were outlined. In this chapter, the experimental environment, experiments run, and observations to characterize this behavior are mentioned.

4.1. Experiments

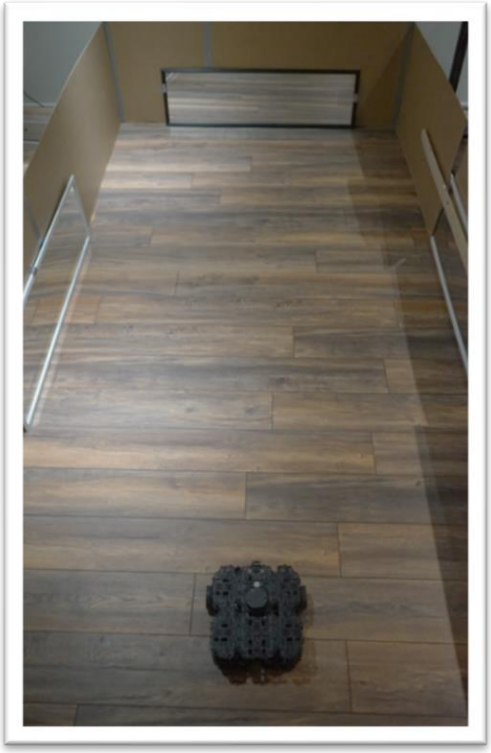

Figure 26: Constructed Experimental Setup

To test the behavior in a more standardized setup, an environment was constructed as a 12 ft x 6 ft rectangular box with three potential mirror locations.

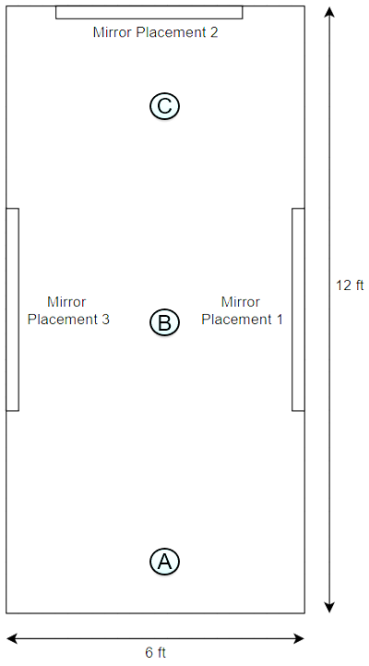

Figure 27: Representation of the experimental setup with mobile robots A, B and C

The experimental runs had four separate tests with three mirrors that were four feet in length. These mirrors were placed in one or more locations, as seen in figure 27, and the robot was driven in the same direction from one end to the other end of the setup from the start position "A" to the end position C.

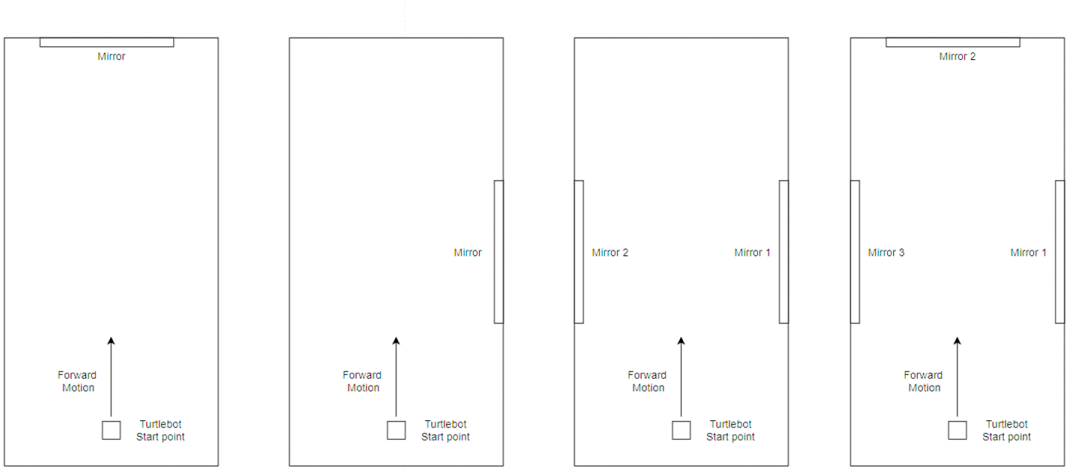

Figure 28: Representation of Tests 1-4 with single or multiple mirrors

Table 5: List of Experiments

Test Number	Procedure
Test 1	Mirror Placement 2 Turtlebot from A →C
Test 2	Mirror Placement 1 Turtlebot from A →C
Test 3	Mirror Placement 1; 3 Turtlebot from A →C
Test 4	Mirror Placement 1; 2; 3 Turtlebot from A →C

The first two tests use only one mirror. The third and fourth methods use multiple mirrors. This strategy was used to understand how LiDAR data collection would be impacted by the direction of travel of the LiDAR.

4.1.1. Test #1

In the first test, we drive the robot from position A to C toward a single mirror.

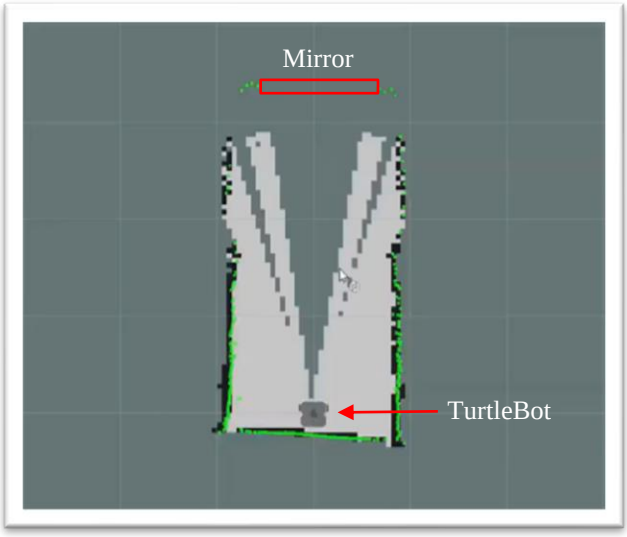

Figure 29: RViz map of Test 1 – Driving toward a mirror; Position A – Bottom of the experimental setup

At position A, the top borders of the environment have not yet been formed. This is due to the LiDAR range being approximately 10-11 feet. The top boundary that is not yet formed is still out of bounds. There is a large empty space seen in the area of the LiDAR where the mirror is located.

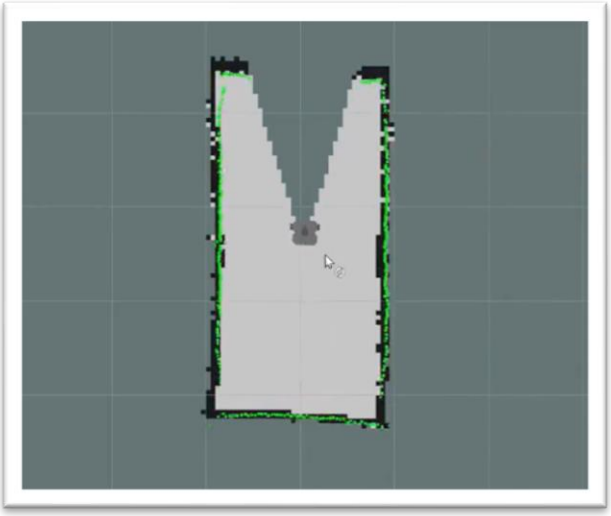

Figure 30 RViz Map of Test 1 – Driving toward a mirror; Position B – Middle of the experimental setup

At position B, it can be seen that the upper boundary is now in range, and the entire experimental area should be mapped. However, the mirror location is still not detected, and the area within the field of view between the mirror location tapered to the LiDAR is still unaccounted for, showing up as an unmapped area in the RViz visualization tool.

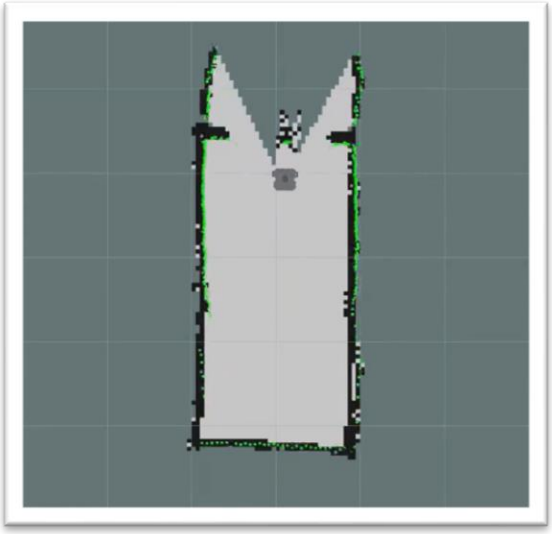

Figure 31: RViz Map of Test 1 – Driving toward a mirror, Position C – Top of the experimental setup

At position C, the robot has completed its course. Although the mirror should be a solid boundary, the LiDAR has mapped past the region of the mirror. This is likely through the detection of the reflection of the test environment. As the mirror is a plane mirror, it can be seen that the reflections of some parts of the boundaries are equidistant from the surface of the mirror as the physical boundary. Another interesting observation is that the turtlebot itself was detected on the surface of the mirror when the robot was in the range normal to the mirror. Some parts of the mirror remained undetected.

To summarize test 1, when the Turtlebot moved toward a plane mirror, the mirror was not detected. Some boundary reflections were observed. The Turtlebot itself was detected when it moved within the detection range of the LiDAR normal to the mirror.

4.1.2. Test #2

In the second test, the robot travels alongside a single mirror from A to C

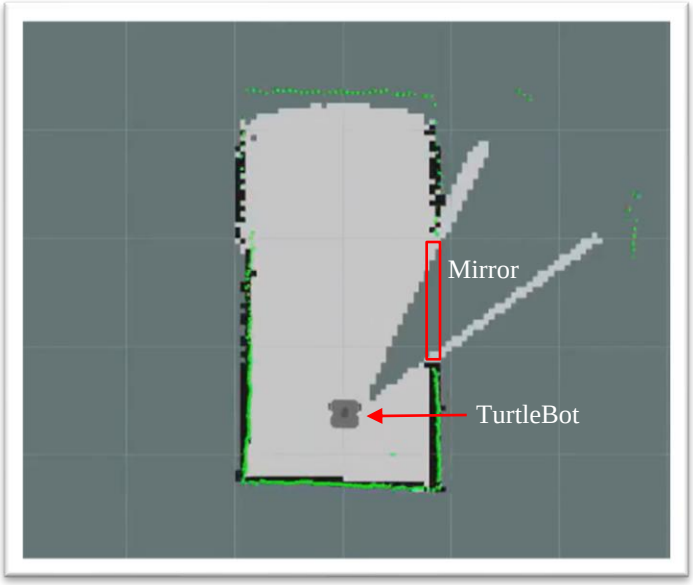

Figure 32: RViz Map of Test 2 – Driving alongside a mirror, Position A – Bottom of the experimental setup

At position A, it can be seen that the top of the boundary is still out of range, and some of the areas between the LiDAR and the mirror within the field of view of the LiDAR are undetected. A small amount of boundary reflection is seen on the other side of the mirror.

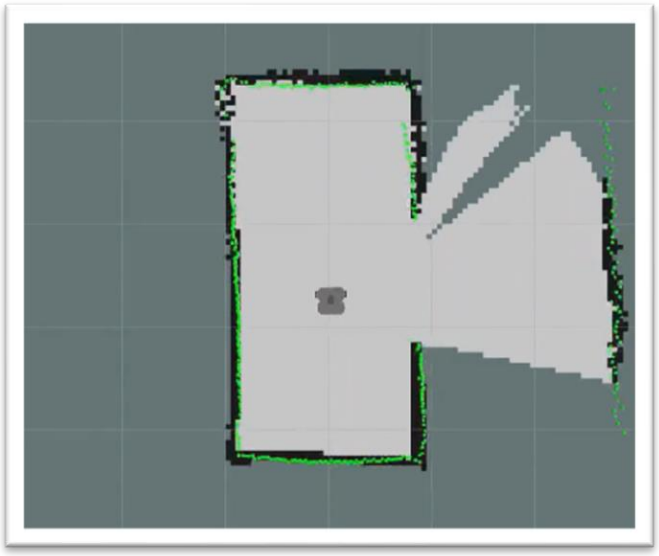

Figure 33: RViz Map of Test 2 – Driving alongside a mirror, Position B – Middle of the experimental setup

At position B, there is no more negative space within the test boundary. A significant amount of the reflection of the left boundary wall is now detected behind the surface of the mirror at the same distance from the mirror as the physical wall. An expected observation would have been to see the robot detected on the surface of the mirror as it passed. This observation was seen in some runs of this test.

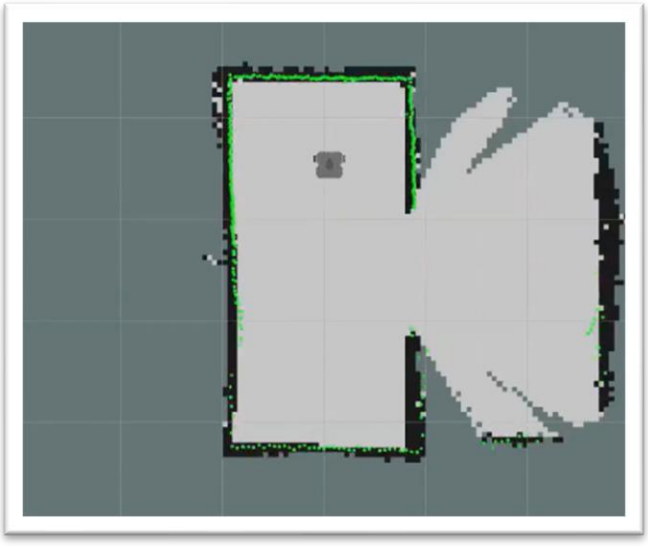

Figure 34: RViz Map of Test 2 – Driving alongside a mirror, Position C – Top of the experimental setup

At position C, the reflection of the left wall boundary is detected behind the mirror and is well defined on the RViz visualization tool. Some of the bottom boundary reflection is also seen. The robot does travel slowly to increase the number of LiDAR scans it is able to take per distance traveled. However, observing the amount of boundary reflection detected, it is possible that if the robot was slowed down further, a complete boundary reflection would have been seen.

To summarize test 3, when the Turtlebot moved alongside a plane mirror, it was observed that the SLAM node mapped false boundaries behind the mirror. Reruns of the test showed that the robot was continuously mapped on the surface of the mirror when traveling normally to the mirror. This is likely due to inconsistent lighting conditions in the test environment.

4.1.3. Test #3

In the third test, the mobile robot travels between two plane mirrors placed parallel to each other while traveling from point A to point C.

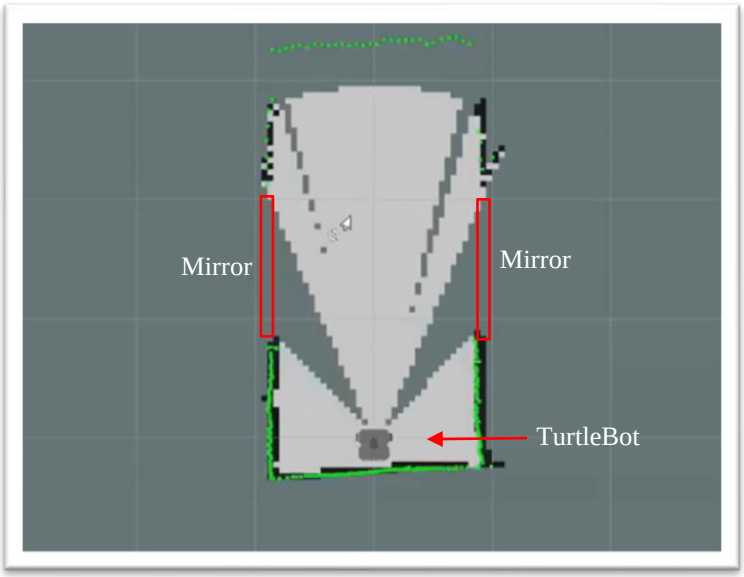

Figure 35: RViz Map of Test 3 – Driving between mirrors, Position A – Bottom of the experimental setup

At position A, it can be seen that the top of the boundary is out of the range of the LiDAR, and some of the areas between the LiDAR and the mirrors within the field of view of the LiDAR are undetected on both sides.

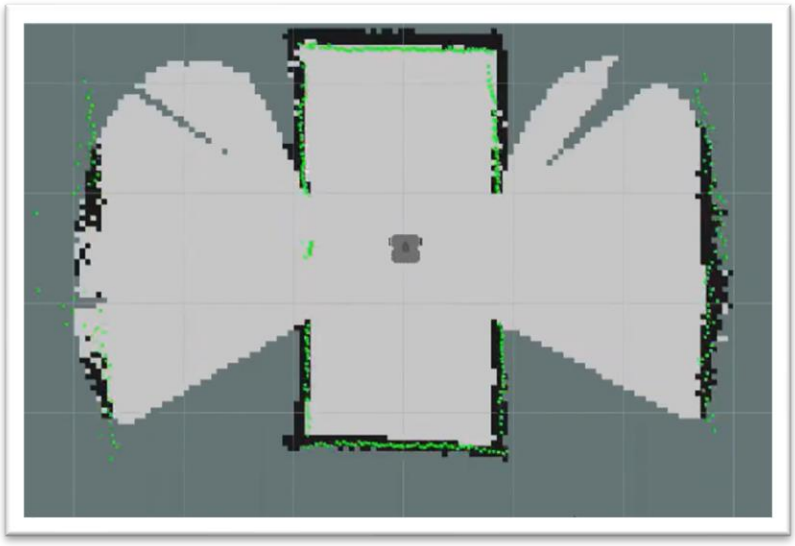

Figure 36: RViz Map of Test 3 – Driving between mirrors, Position B – Middle of the experimental setup

At position B, it is seen that the reflection of the boundary walls can be seen on both sides behind both mirrors. It can be assumed that both mirrors map the reflection of the opposite walls. Another interesting observation is that the robot is detected on only one of the mirrors. This was continuously corrected as the robot traveled along the mirror and did not leave a map boundary after crossing the mirror.

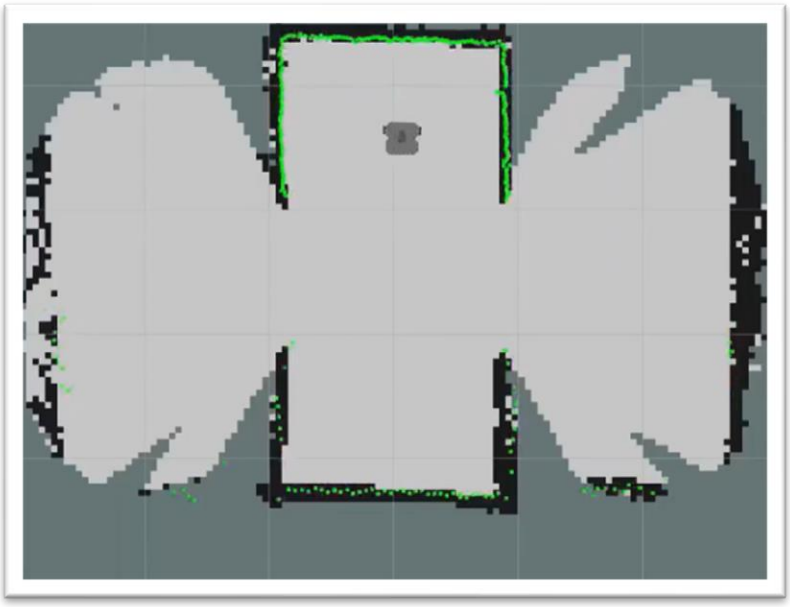

Figure 37: RViz Map of Test 3 – Driving between mirrors, Position C – Top of the experimental setup

At position C, solid boundaries can be seen behind both mirrors. The mirror reflections of the bottom boundaries are partially formed as well. The left boundary reflection map is slightly less prominently mapped than the right side. A potential cause for this could be the disturbance caused by the moving turtlebot map. Inconsistent lighting could also cause the difference in mapping between the two boundary reflection maps.

To summarize test 3, when the Turtlebot traveled in a straight line between two parallel mirrors, it was observed that the maps of the boundary wall were seen on both sides behind both parallel mirrors; however, one mapped reflection was lower in density of map points than the other. The Turtlebot was also continuously detected on the surface of one of the mirrors normal to the target.

4.1.4. Test #4

In the fourth test, all three mirrors are placed in the environment, and the robot is driven from point A to point C.

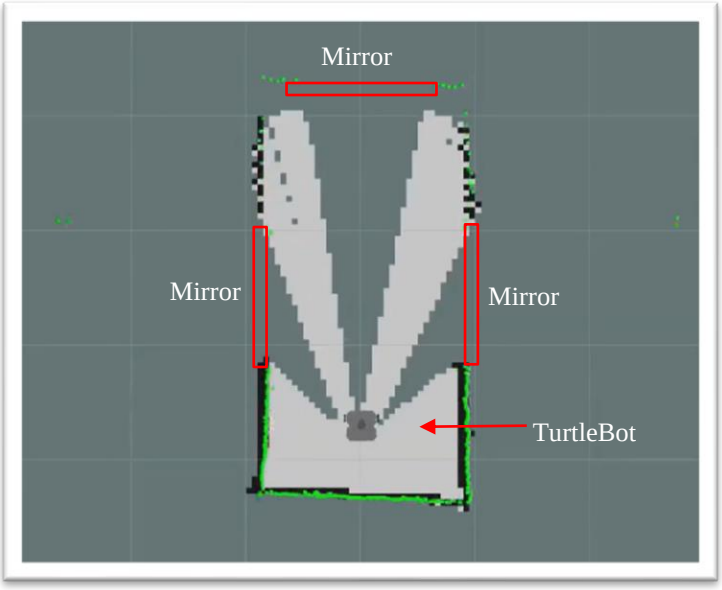

Figure 38: RViz Map of Test 4 – Multiple mirror environment, Position A – Bottom of the experimental setup

At point A, similar to the first three experiments, it was seen that there was negative space in the field of view between the LiDAR and the mirror locations.

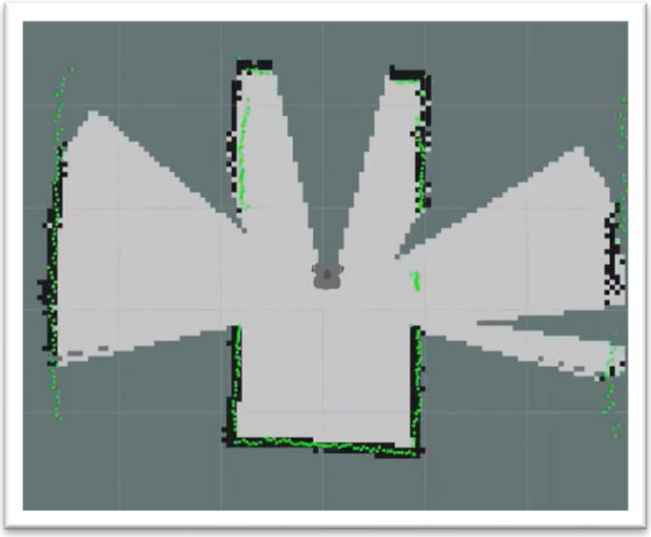

Figure 39: RViz Map of Test 4 – Multiple mirror environment, Position B – Middle of the experimental setup

At point B, it was seen that the reflections of the boundaries of the 12-foot sides of the environment were mapped behind the mirrors as in previous tests. With respect to the mirror at the top of the environment, there is only negative space in the field of view between the LiDAR and the mirror. Additionally, unlike test 3, where the turtlebot was continuously mapped in the left mirror, in test 4, the turtlebot was continuously mapped in the right mirror.

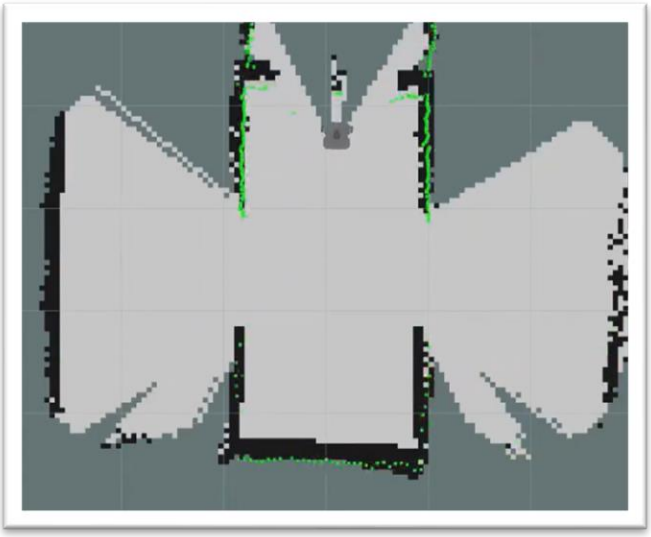

Figure 40: RViz Map of Test 4 – Multiple mirror environment, Position C – Top of the experimental setup

At Position C, it can be seen that the top mirror also allows for some of the reflections of the side boundaries to be mapped while also mapping the robot reflection on the surface of the mirror, normal to the robot. It also leaves some negative space where no boundaries are in range. There is also a solid reflection of the left and right boundaries mapped behind both mirrors. The right reflected boundary map is less dense than the left. This is likely due to disturbance caused by the robot mapping and correction as it passed through the parallel mirrors.

To summarize, this test reinforces the observations of the first three experiments where negative space is seen where no boundary reflections are in range. When boundary reflections are in range, they map behind the mirror at the distance of the boundary from the surface of the mirror. The robot is detected on the surface of the mirror.

4.1.5. Test #5

To reinforce the understanding of detecting the robot on the surface of the mirror, a stationary test was performed by placing the robot in front of a single mirror, running the SLAM algorithm and viewing the map on the Rviz visualization tool.

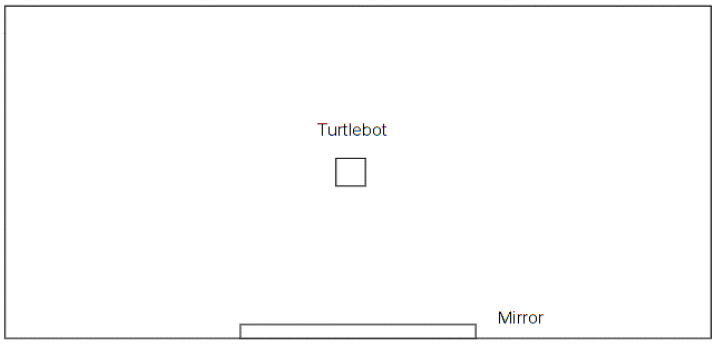

Figure 41: Single mirror stationary test representation

As seen in figure 42, a solid collection of map points is mapped on the surface of the mirror. The remaining length of the mirror is not mapped, but the reflected portion of the opposite boundary is mapped behind the mirror.

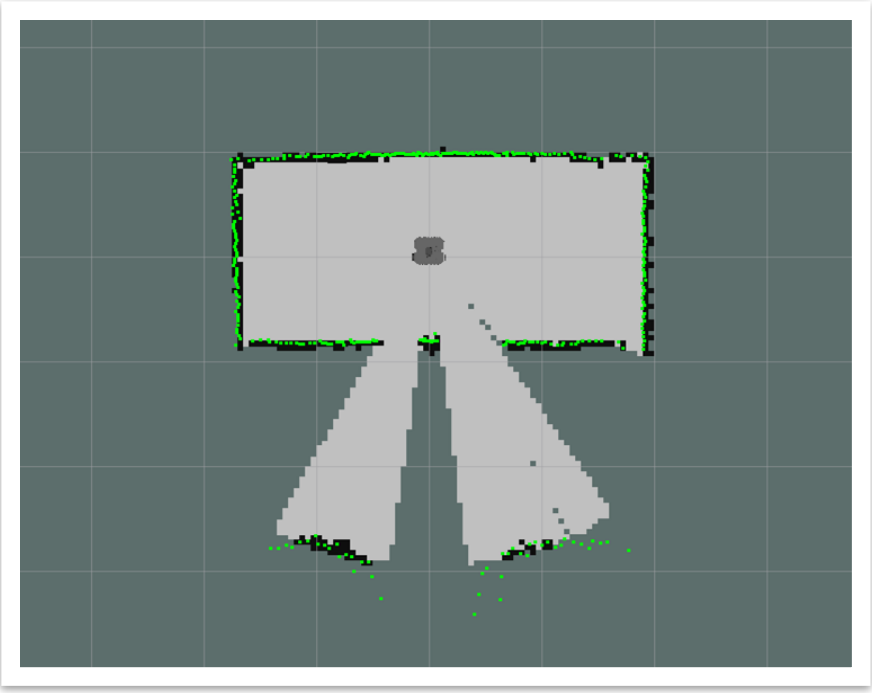

Figure 42: RViz Map of the stationary single mirror test

4.2. Characterization of Results

The experimental tests conducted with mirrors located at various locations along the navigation path of the robot showed that mirrors in the environment cause inconsistencies in mapping using LiDAR data. Several important observations were noted.

4.2.1. Observation #1

Detection of negative space where mirrors or present was seen consistently over multiple experiments both before and after mirror a fully in the range of the mirrors. This could be consistent with the uniform reflection of all LiDAR scan points from the start to end length of the mirror. It is possible that the total reflection causes any boundary reflections that could potentially be detected to be out of the LiDAR range. Alternatively, perhaps the LiDAR was unable to receive any intensity from any other reflected points.

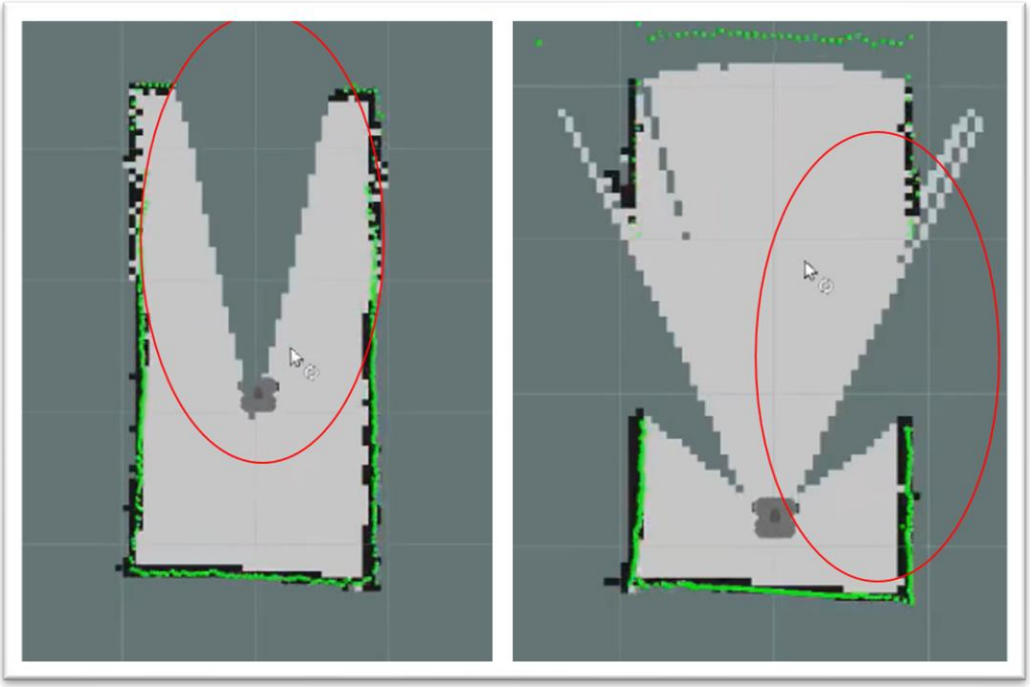

Figure 43: Observation of negative space in mirrored environments

Figure 44 shows the classification of the LiDAR scan datapoints when the LiDAR rotates within the range of a plane mirror. All scans that lie outside the length of the mirror are unaffected by having a mirror in the environment. The points between the mirror and the LiDAR and the mirror are affected. Beyond the mirror, all LiDAR readings are false.

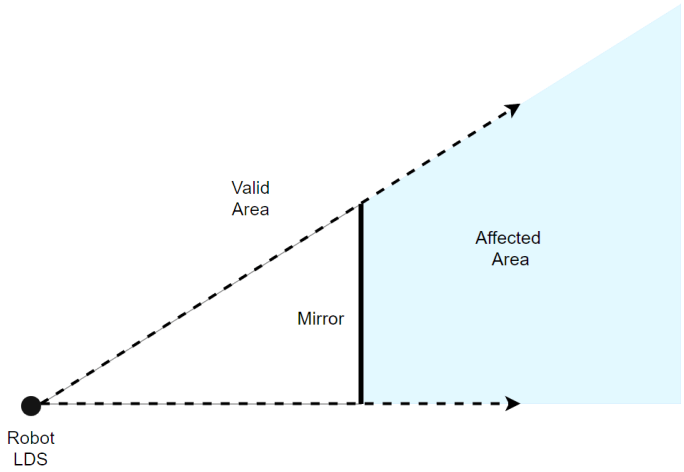

Figure 44: Affected area of LDS Scan in the presence of a mirror

4.2.2. Observation #2

False detection within the mirror was seen consistently when the opposite boundary was in range. Observing this behavior, it can be assumed that any other diffuse object placed at a distance from the mirror where the reflection of the object would be within the range of the LiDAR would also be mapped.

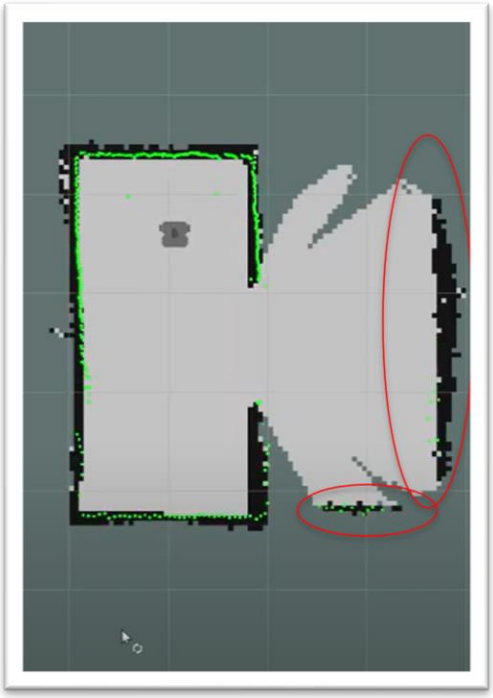

Figure 45: Observation of the reflected boundary detected behind the surface of the mirror

This behavior seen in figure 2 is consistent with the law of reflection for a plane mirror. The law of reflection states that for reflection off a plane reflective surface, the angle of incidence is equal to the angle of reflection.

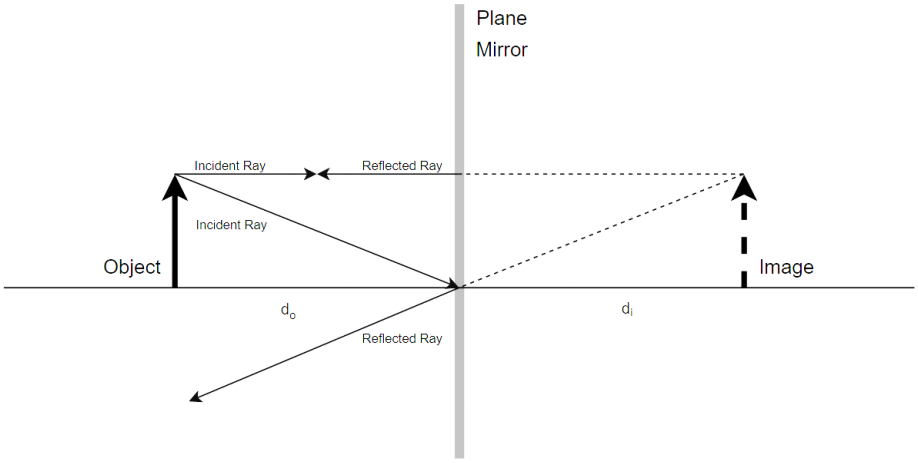

Figure 46: Schematic of object reflection in a plane mirror

The reflection off a plane mirror is shown in figure 46. For a plane mirror, this would further infer that the distance of the object from the mirror (d_o) is equal to the distance of the image from the mirror (d_i)

$$d_i = d_o \tag{14}$$

$$d_{lid/i} = d_{lid/mir} + d_o \tag{15}$$

Therefore, the distance of the image from the LiDAR ($d_{lid/i}$) is equal to the sum of the distance between the LiDAR and the mirror ($d_{lid/m}$) and the object from the mirror.

4.2.3. Observation #3

The robot detected itself on the surface of the mirror. According to the previous observation, all obstacles detected have been detected at the correct location of the mirror image; however, in the case of the robot itself, from the data collected, it can be inferred that the robot is being detected on the surface of the mirror rather than a few feet behind the mirror where their mirror image should have been.

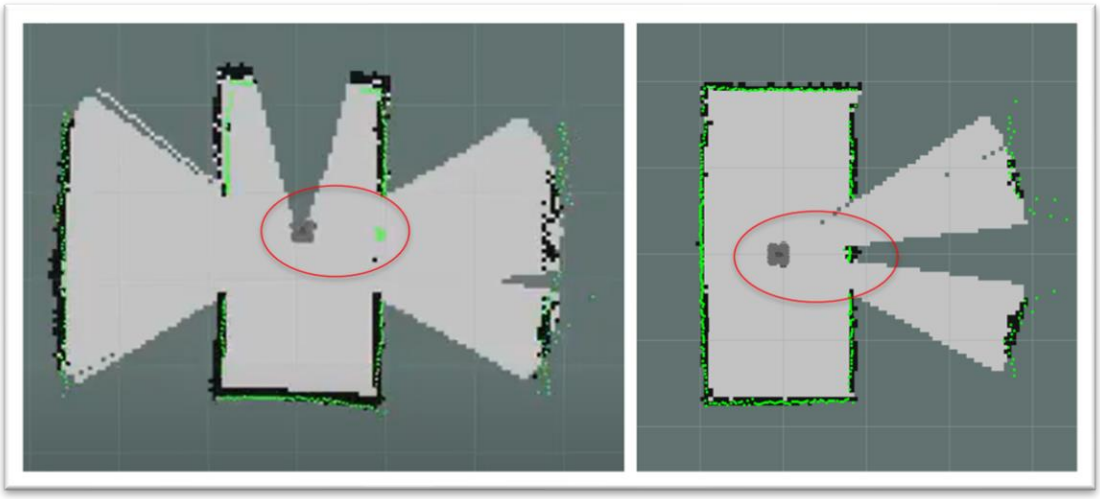

Figure 47: Observation of the robot detecting itself on the surface of the mirror

From this information, it can be inferred that the mirror acts as a diffuse surface when the location of the robot LiDAR scans is normal to the mirror. Although this behavior is inconsistent with observation 2, this observation is consistent with [62], where a similar behavior was found for glass surfaces. However, this may not be the only observation to rely on, as although it was consistently seen in at least one of the mirrors in all the tests run; there were some inconsistencies found during different runs of the robot through parallel mirrors. The diffuse behavior was only seen normal to the mirror (figure 48).

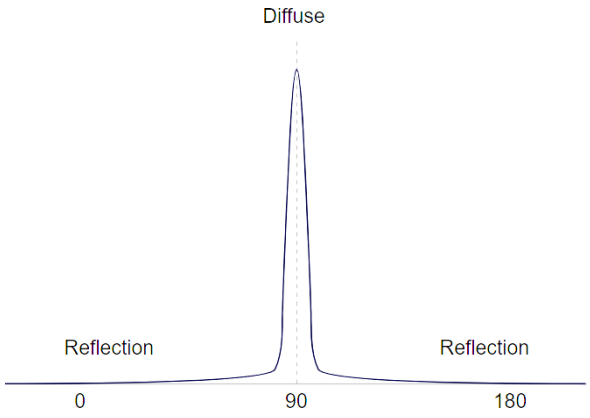

Figure 48: Representation of specular surface acting as a diffuse surface at normal

CHAPTER 5

Future work and Conclusions

In the previous section, experiments were run, and observations were made related to the mirror detection problem. This thesis studied and characterized the effects of 2D LiDAR scan data for mapping indoor environments. An introduction to indoor navigation was provided, which included covering concepts of autonomous navigation, path planning, localization, and mapping. The scope of the problem was outlined, providing a review of the literature, experiments were run to understand the problem and observations were made. In this chapter, potential solution approach to the problem has been discussed, potential future work and the conclusion will be provided.

5.1. Data Collection

Shown in Figures 49, 51, and 53, LiDAR scan points are plotted on an XY plot as seen in experiment three. LDS-01 has a 1° resolution. Each rotation of the laser will thus take 360 scan points. As seen in the figures, several of those scan points are unique with respect to range. The robot receives both range and intensity data to use for the process of map building.

$$Intensity_{\ received} = \frac{Intensity_{sent}}{Distance^2} \tag{16}$$

The unit of range is metres (m) and the unit of intensity is Watts/metre (W/m). In Figure 50, 52, 54 the intensity has been scaled down by a fraction of 600 to compare the plot size. These intuitively show that the range is inversely proportional to the received intensity back to the receiving element after detecting an element.

From this information, there are two entire sets of data to categorize and work with. Based on the experimental observations, the next step is to develop a modification to the SLAM algorithm for mirror detection using a combination of the four distinct categories of types of data collected.

The following figures show some data collected for Test 2. This is a good example to show some normal boundary conditions as well as some negative space when the robot position is close to the start position of the test. Normal detection for intensity depends on the distance of the LiDAR from the boundary. This can be in the range of 3000 to 4000 W/m when the range is around 0.5m from the boundary.

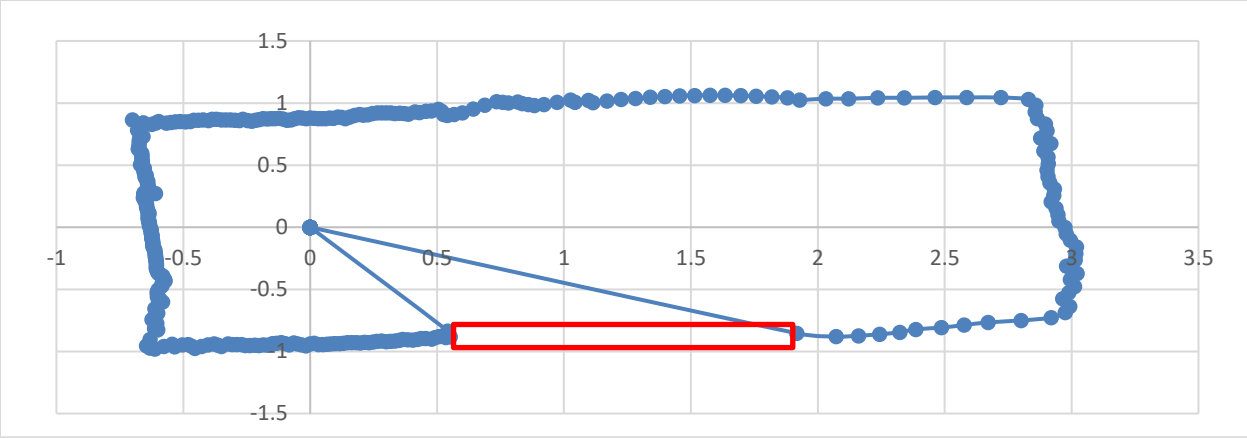

Figure 49: 360 laser scan points for range mapping radially around the robot (Test #2)

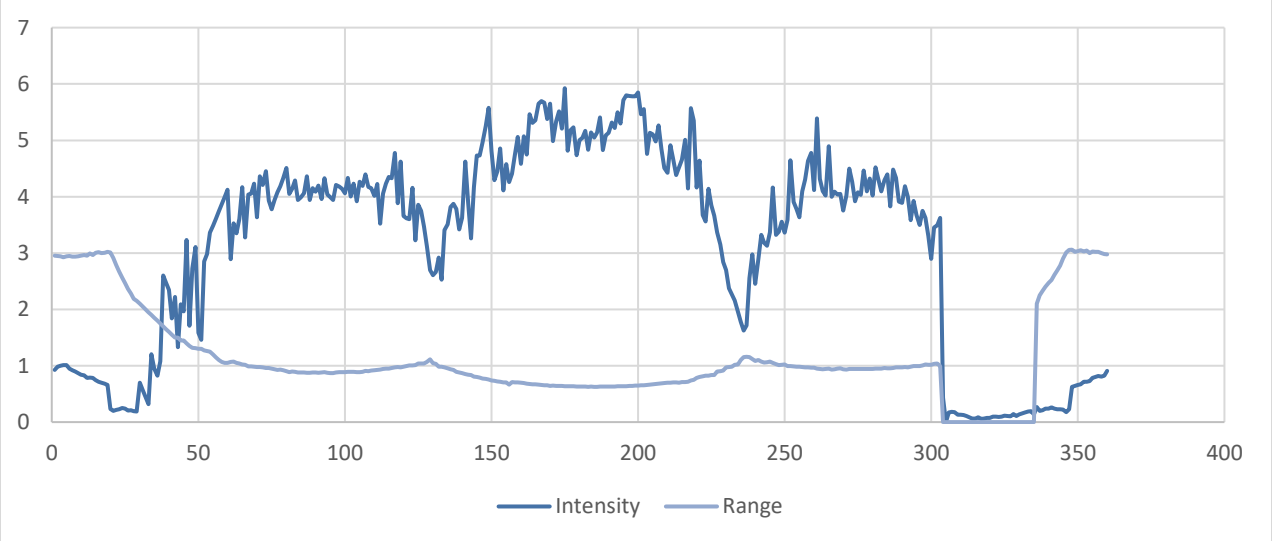

Figure 50: 360 laser scan points for received intensity mapped linearly (Test #2)

In the case of a single mirror and the travel of the robot in a single direction, a very clear distribution of data is seen where there is no range data. However, some intensity data is still collected, which is inconsistent with equation 16.

The following figures show some data collected for Test 3. This is a good example to show some normal boundary conditions, some negative space as well as some false detection when the robot position is close to the start position of the test.

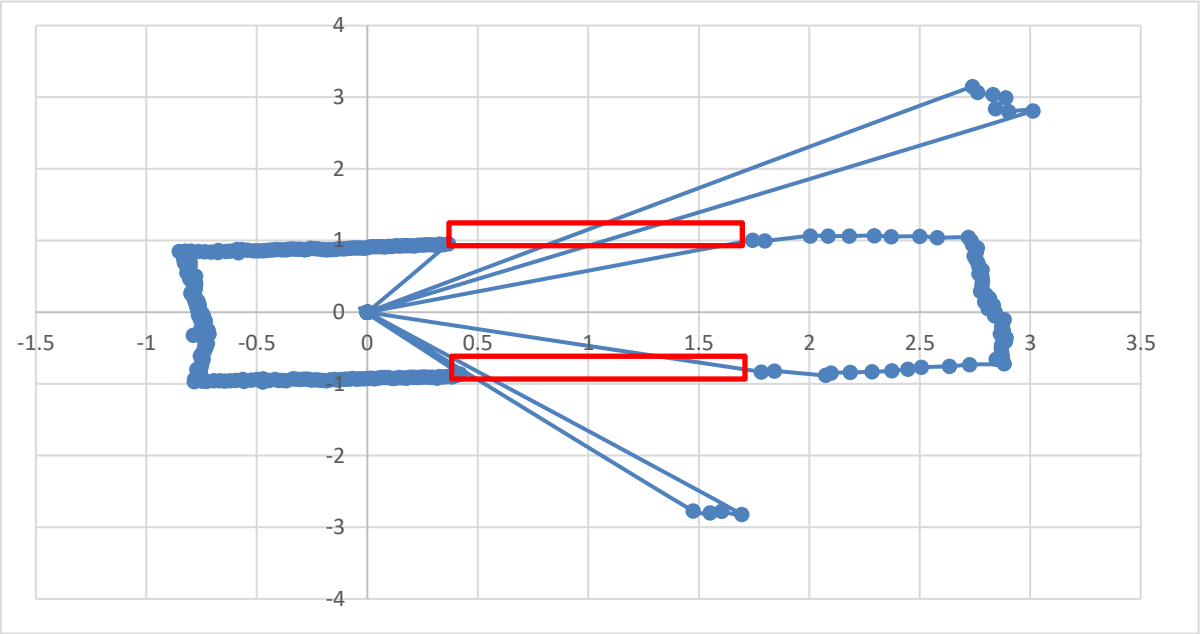

Figure 51: 360 laser scan points for range mapping radially around the robot (Test #3)

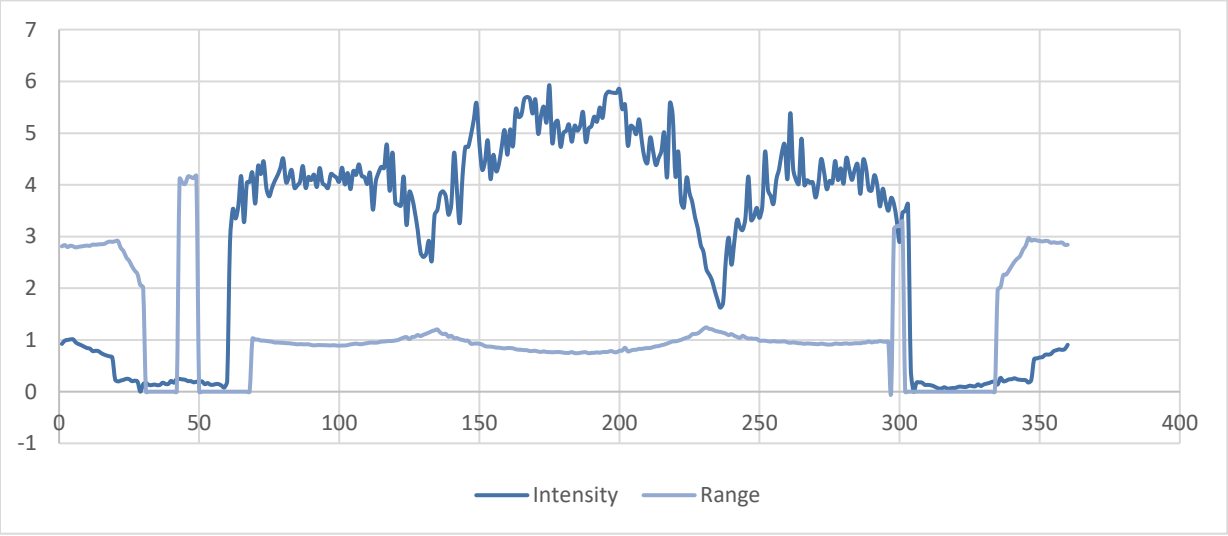

Figure 52: 360 laser scan points for received intensity mapped linearly (Test #3)

For both normal detection and false detection, the data collected shows some noise inconsistencies in intensity data.

The following figures show some data collected for Test 4. This is a good example to show some normal boundary conditions in comparison to false boundary detection and robot self-detection when the robot is in the middle position of the test.

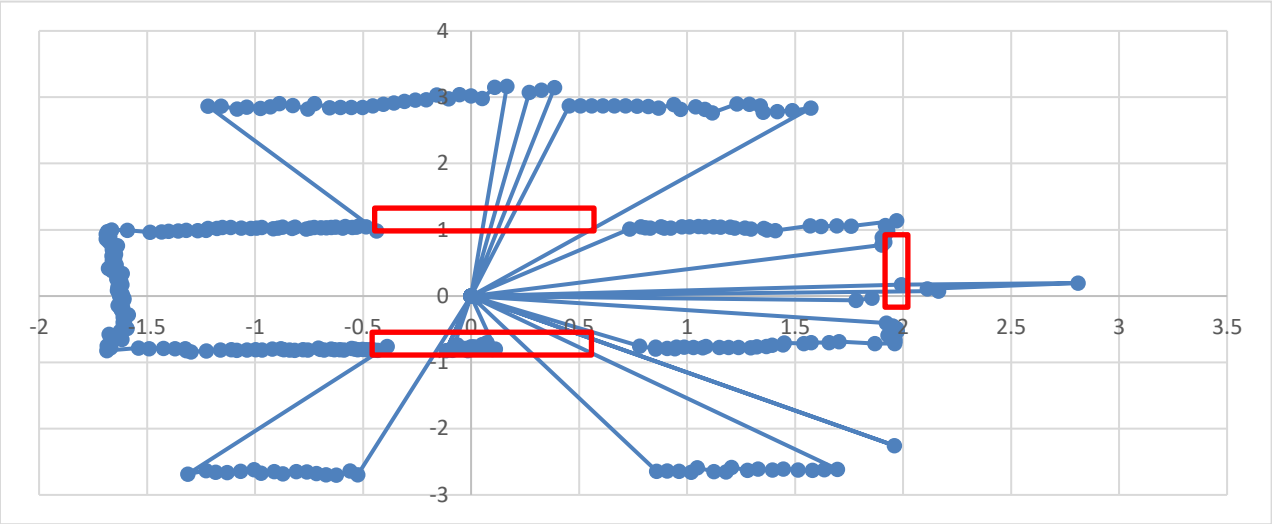

Figure 53: 360 laser scan points for range mapping radially around the robot (Test #4)

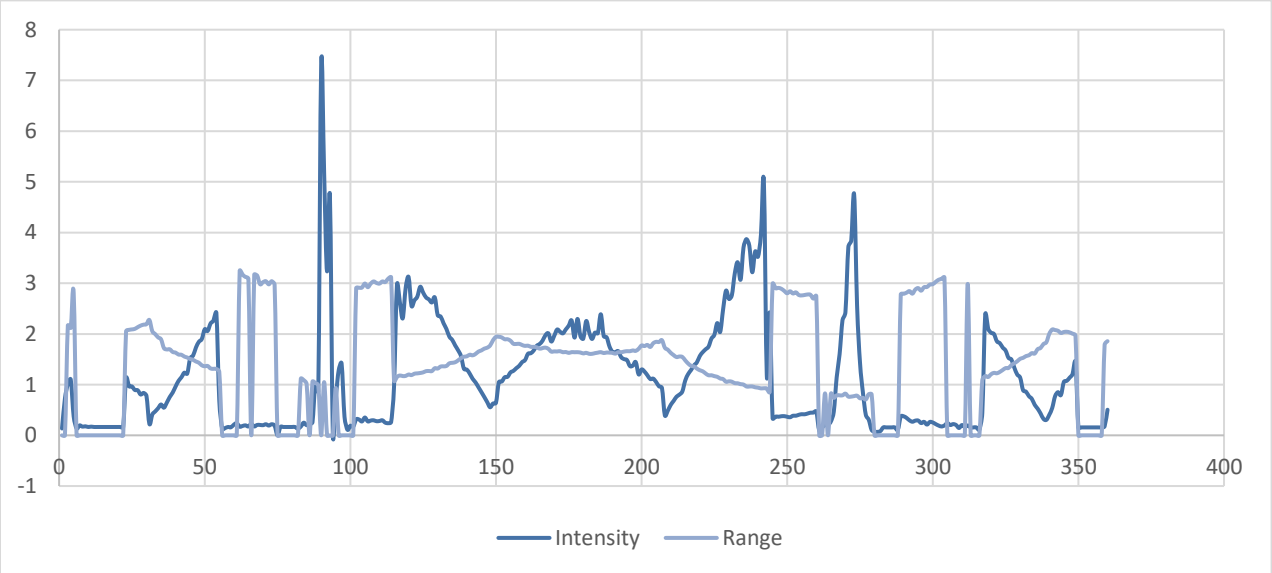

Figure 54: 360 laser scan points for received intensity mapped linearly (Test #4)

As this data was collected at a different robot location and environment, different range and intensity data is seen for normal detection through data patterns is similar. High intensity was seen for self-detection and inconsistent readings with respect to range data were seen at mirror locations.

Normal detection or the detection of diffuse boundaries that are detected and mapped correctly by the LiDAR. Negative space or the lack of detection and lack of mapping in the range between the LiDAR and mirror at various ranges and directions of motion with respect to the mirror.

Reflective detection or detection of mirror reflection of various diffuse surfaces where the mirror reflection of the surface is in range of the LiDAR. Self-detection of the robot on the mirror surface can also be considered.

5.2. Potential Approach - Concepts

This provides the opportunity to use a classification-type algorithm to work alongside the SLAM algorithm to map out mirrored boundaries. A high-level process is highlighted in the figure below.

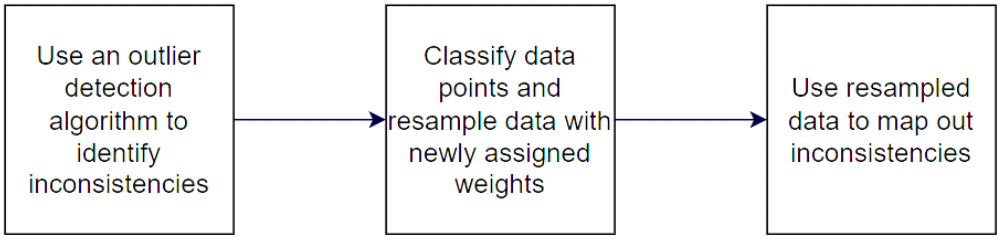

Figure 55: High-level solution approach for mirror detection issues

Machine learning algorithms are considered for outlier detection. In recent times, applications of machine learning span several industries, including healthcare, retail, and education. Artificial intelligence and machine learning have become very popular in robotics and related problems. Machine learning is the study and application of systems and algorithms that use data to continuously learn and update on their own through experience [87].

Machine learning can be divided broadly into three categories depending on the goal of the learning algorithm and the information available to it.

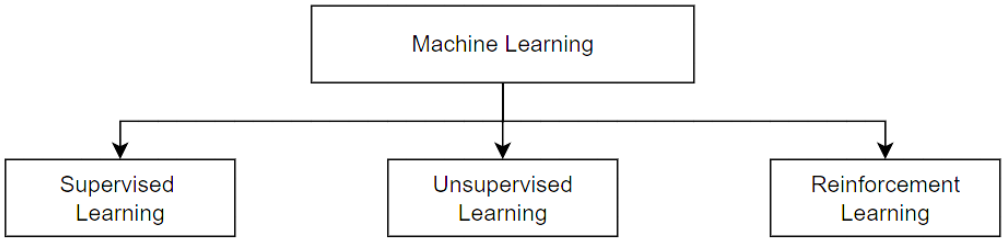

Figure 56: Classification of types of Machine Learning

Supervised learning algorithms are data-driven techniques that use known labeled data to teach the system how to perform a prediction on a future scenario [88]. Supervised learning is useful for problems that require prediction or regression (such as weather/behavior prediction) or classification (such as image search) where large amounts of data exist for training.

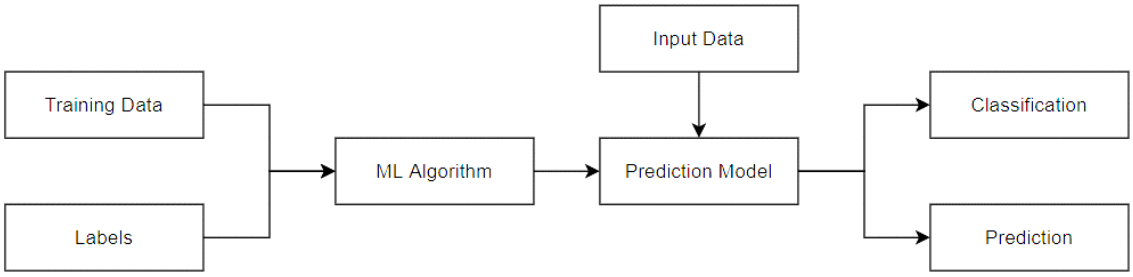

Figure 57: Simplified algorithm for supervised learning

Supervised learning has several algorithms associated with it, including linear classifiers, such as naïve Bayes classifier and support vector machine; random forest decision trees; neural networks, etc. [88].

Unsupervised learning does not require a labeled dataset to teach the system; rather, it learns through its own experience using data collected in real time. It is very useful to find unknown patterns and can take place in real time. Clustering models allow for the input data to be divided by similarity into clusters. Association models are used to identify sequences in the data.

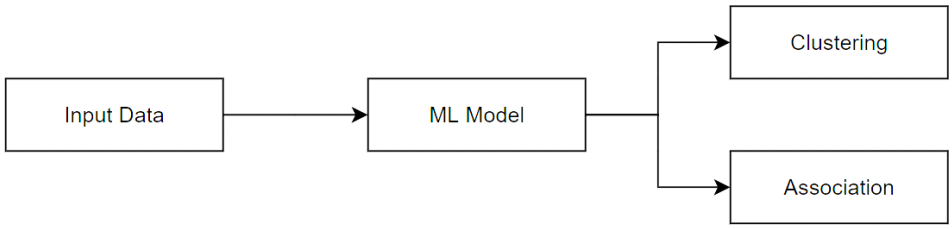

Figure 58: Simplified Algorithm for Unsupervised Learning

Reinforcement learning uses the concept of rewarding preferred behavior and will penalize the system for unexpected or incorrect behavior. This is used to teach the system come up with a solution for a problem through trial and error.

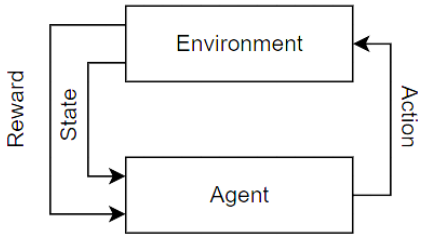

Figure 59: Simplified Algorithm of Reinforcement Learning

There are several other machine classifications, including hybrid types such as semisupervised learning and statistical learning methods.

Given the mirror data collected and the types of machine learning algorithms, the type of machine learning that would be most suitable for the mirror problem application would be an unsupervised learning algorithm performing clustering. Clustering is chosen because there is no learning period for mirror data classification that can be provided to teach the model prior to mapping. It can also be implemented in real time.

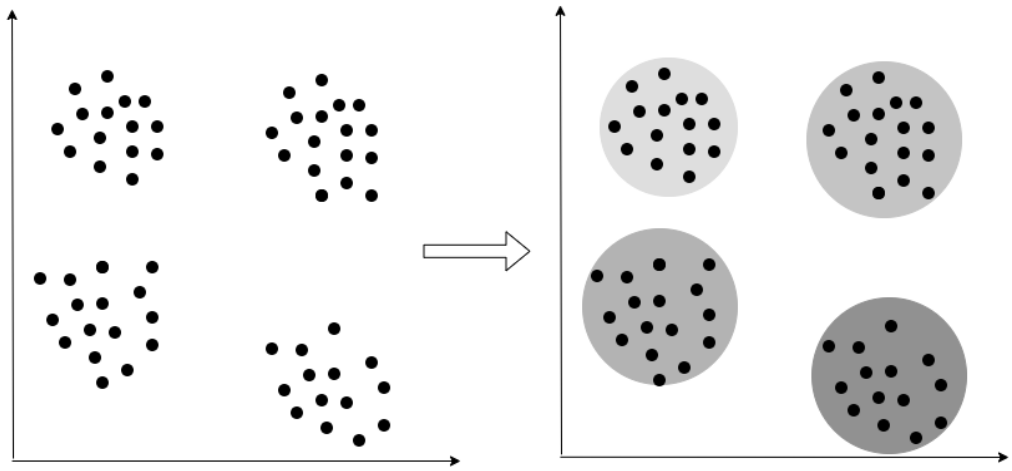

Figure 60: Pictorial representation of Clustering

Clustering can be performed based on several topics, including distribution, partition, and density [89]. A few of the more common clustering algorithms are detailed below.

Table 6: Comparison of Clustering Algorithms (K-Means, GMM and DBSCAN)

Algorithm	Advantages	Disadvantages
K-Means	- Less Complex - Can be extended	- Form circular clusters - Difficult to cluster outliers.
Gaussian Mixture Models (GMM)	- Soft Classification - Flexible for more data scans - Highly scalable	- Highly Complex - Lack of Flexibility
DBSCAN	- Does not require a specification on clusters - Noise identifiable - Can be used with various sized and shaped clusters	- Poor with highly dimensional data

An example of partition-based clustering is K-Means. This is one of the simplest unsupervised learning algorithms. In K-means, k represents the target number of centroids around

which clusters are grouped. The k-number of randomly selected centroids will be iterated until they stabilize [90]. These clusters assume circular shapes.

The Gaussian Mixture model (GMM) is a distribution type clustering algorithm. It works similarly to k-means; however, it assumes that the datapoints follow Gaussian distributions, and it soft assigns the datapoints in the cluster to consider and can therefore account for more uncertainty.

Density-based spatial clustering of applications with noise (DBSCAN), as seen in the name, is a density-based clustering algorithm. This algorithm works by visiting each point in the dataset and determining which cluster each of them belongs to depending on the distance.

There are several other clustering algorithms with different properties (picture included in Appendix D). Comparing the three common clustering algorithms [89], the proposed chosen algorithm for the mirror problem application is DBSCAN.

DBSCAN is a density clustering algorithm. This is mostly due to the type of data seen in the mirror experiments. The clusters created by DBSCAN can be arbitrarily shaped and therefore should likely fit.

Two main parameters are used in the DBSCAN algorithm that need to be configured for optimal clustering.

Epsilon (ε) is a specified distance that allows the algorithm to understand how close together data points can be to be considered part of the same cluster. Therefore, if the distance between two data points is d_{12},

$$\varepsilon \geq d_{12} \rightarrow points\ 1\ and\ 2\ are\ neighbors \qquad (17)$$

ε needs to be chosen optimally, as if the eps value is too small, several points that do not fit into that distance will be considered outliers. However, if the epsilon value is too large, optimal clustering will not take place, as too many points may fall into the same neighborhood. The distance of the dataset can be used to find the ε value.

MinPoints is another parameter used for DBSCAN. This parameter specifies the minimum number of data points required to create a cluster. It is based on the number of dimensions (D) of the dataset.

$$minPoints \geq D + 1 \qquad (18)$$

Therefore, the minimum value of minPoints is 3. Larger datasets would likely have larger values of minPoints. Datasets with considerable noise should also use larger minPoint values.

5.3. Potential Solution Approach

Figure 57 shows the sequence of events to enable clustering through DBSCAN. Here, density connected points refer to two clusters having one or more points that have neighbors within the ε distance of both clusters.

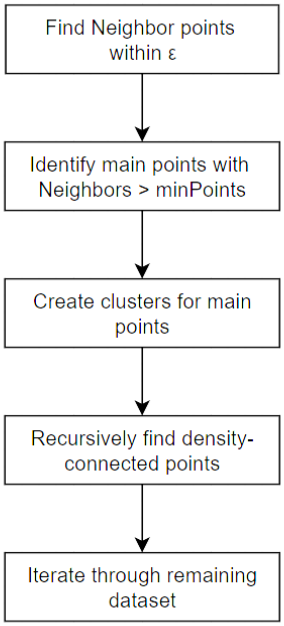

Figure 61: Simplified algorithm of DBSCAN Clustering

Machine learning algorithms can be used on the ROS platform and the Turtlebot. Anaconda can be used with a compatible version of Python. For example, Anaconda 5.2.0 can be used for Python 2.7.

ROS packages need to be installed as well as dependency packages between ROS and Anaconda. Through Anaconda, TensorFlow and Keras can also be used. After this installation, NymPy libraries can be installed, and Pandas can be used to bring in DBSCAN algorithms to the environment.

The proposal to solve the mirror problem is to combine the Rao Blackwell Particle Filter that is used in the GMapping node along with DBSCAN to cluster mirrored points and proceed to remove them in a post filtering process. In the prefiltering process, lidar scan readings are used as input to the DBSCAN clustering algorithm. As seen in the characterization of the results, all behaviors affected by mirrors have distinct separations from normal scan data on diffuse objects.

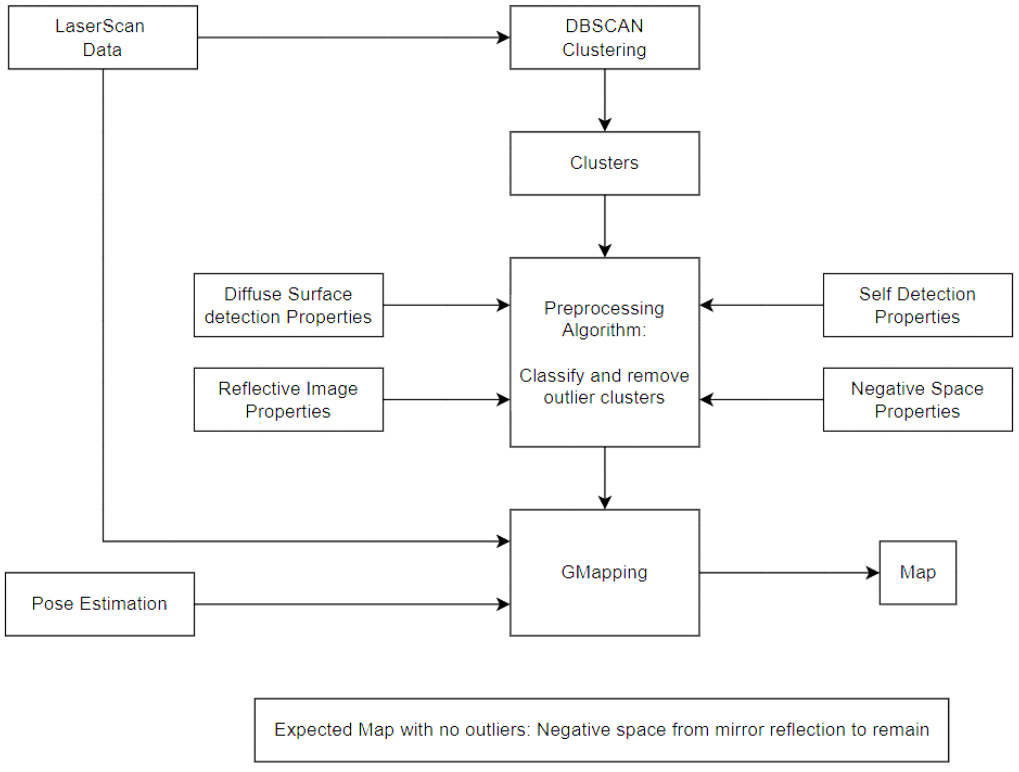

Figure 62: Proposed Preprocessing Algorithm

Each outlier property can be categorized based on range and intensity readings. Diffuse surface properties will remain within a normal range of range and intensity. Negative space properties should provide no range readings and very low or no intensity readings. Self-detection properties can be characterized with very high-intensity readings. Mirror reflection properties will mostly include an area of no range or intensity. These properties can be fed into the algorithm to remove clusters that are affected by mirror reflection.

The next step would be to implement a postprocessing algorithm. Considering all mirrors to be plane mirrors, in this algorithm, border conditions of all empty map scans will be considered. Using ROS parameters to update the SLAM node, the new map should connect the boundary points.

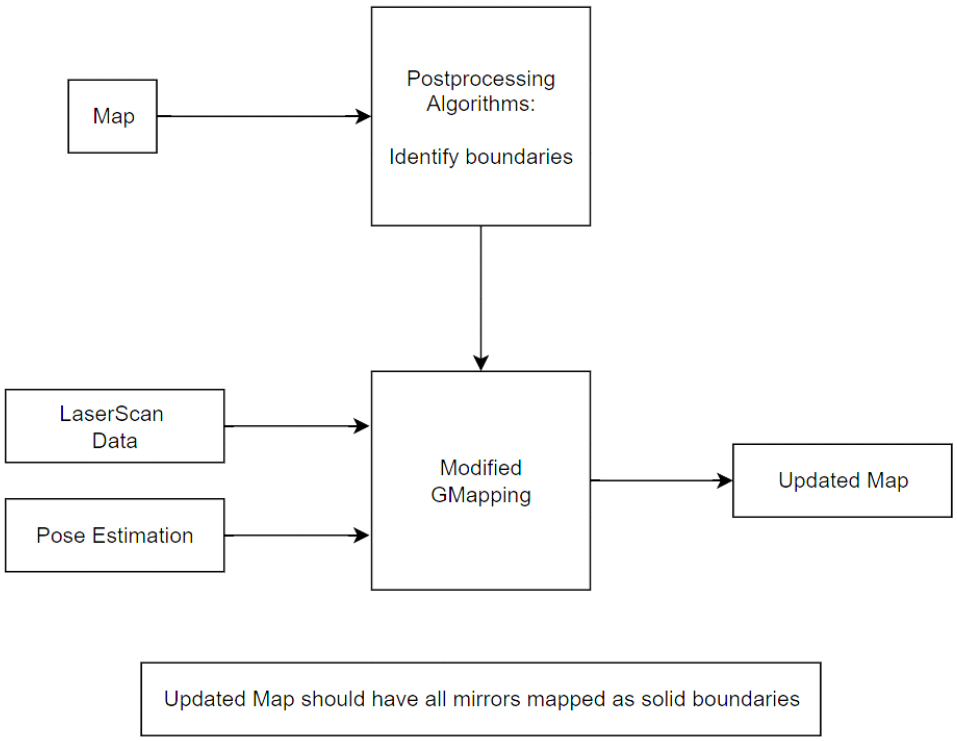

Figure 63: Proposed Postprocessing Algorithm

Implementing the pre- and postprocessing algorithms along with the continuous map update should enable all mirrors to be mapped as diffuse objects in real-time.

5.4. Future Work

This work can be applied in the mapping of indoor spaces with high amounts of optically unique surfaces, including modern office buildings, hospitals, museums, heritage sites, etc. The current proposal is to use a combination mirror properties and outlier identification through a DBSCAN clustering algorithm. Implementation of the proposal and experimentation with different types of classification and learning algorithms would be in the scope of future work.

Solutions to this problem can also likely be implemented for 3D LiDAR applications, including perhaps autonomous vehicles in urban environments with highly reflective surfaces as well as through monocular or camera-based SLAM. Camera-based systems also use principles of light. Therefore, specular and transparent obstacles would pose an absolute issue in obstacle avoidance, SLAM and autonomous systems.

In the future, this research could be extended to apply a separate layer of mapping for reflective systems. The research done here only outlines multiple mirror conditions for the case of plane mirror light properties that are reasonably predictable. It would be interesting to examine the behavior of LiDAR data collection and slam mapping for different types of optical environments, including environments with transparent and translucent boundaries such as glass, which are commonly used as dividing walls in office buildings, spherical or curved mirrors, etc.

Concave and convex mirrors have very different image buildings than plane mirrors. Plane mirrors create an imaginary image at an equal distance from the mirror as the object. However, concave mirrors always create inverted images that can vary in size according to the distance of the object from the mirror and the radius of curvature of the mirror. Convex mirrors always provide erect images but will have a different pattern of reflection than plane mirrors. Gridded mirrors, cracked mirrors and reflection properties from angled mirrrors can also be tested.

Although using a 2D LiDAR may limit the reflection problem, running similar tests with a 3D LiDAR or camera may be more advantageous for a larger scope of applications. These possibilities can also be future research considerations.

5.5. Conclusion

A potential fully autonomous robot was implemented using Turtlebot along with the ROS ecosystem to run experiments, and the SLAM node tested was Gmapping. Several other SLAM algorithms exist and can be used in future work. Conducting tests with mirror objects at various locations along the navigation path yielded several important observations.

- Negative space: When a mirror was in the range of the LiDAR, negative space or undetected area was seen in the range between the LiDAR and mirror due to complete reflection of all the laser scans. This occurs in Position A of all tests.
- False detection: while traveling alongside a mirror, detection was seen behind the surface of the mirror when diffuse surfaces were in the range of the LiDAR. The robot detected and mapped the mirror reflection of the opposite wall to the mirror in Tests 2-4.
- Self Detection: The robot detected itself in the mirror at all points where it was normal to the mirror. This suggests that the mirror acts as a diffuse object when it is normal to the laser scan. This was seen in all tests.
- Noise: It was noticed that in the case of two mirrors, several times the robot was only detected in one mirror. This infers that those inconsistencies in light reflection on the mirror may affect laser scans as well. This was seen in tests 3 and 4.

From these observations, a proposed way of solving the mirror reflection problem is by incorporating the data into a clustering algorithm, DBSCAN. Applying DBSCAN prefiltering to remove inconsistencies and a postprocessing algorithm for mapping could be a solution method.

REFERENCES/BIBLIOGRAPHY

[1] . A. Chatterjee, A. Rakshit and N. N. Singh, Vision Based Autonomous Robot Navigation: Algorithms and Implementations, Springer Berlin Heidelberg, 2013.

[2] W. Ju, "Application of autonomous navigation in robotics," *Journal of Physics Conference Series,* vol. 1906, 2021.

[3] D. D. Paolo, A. Milella, G. Cicirelli and A. Distante, "An Autonomous Mobile Robotic System for Surveillance of Indoor Environments," *International Journal of Advanced Robotic Systems,* vol. 7, no. 1, pp. 019-026, 2010.

[4] J. L. Jones, N. W. Mack, D. M. Nugent and P. E. Sandin, "Autonomous floor-cleaning robot". Burlington, MA, United States Patent US 6,883,201 B2 , 26 April 2005.

[5] J.-H. Kim, "Autonomous Navigation, Perception and Probabilistic Fire Location for an Intelligent Firefighting Robot," ProQuest Dissertations Publishing, Blacksburg, VA, 2014.

[6] E. F. Z. Santana, G. Covas, F. Duarte, P. Santi, C. Ratti and F. Kon, "Transitioning to a driverless city: Evaluating a hybrid system for autonomous and non-autonomous vehicles," *Simulation Modelling Practice and Theory*, vol. 107, 2020.

[7] B. M. Alatise and G. P. Hancke, "A Review on Challenges of Autonomous Mobile Robot and Sensor Fusion Methods," *IEEE Access*, vol. 8, pp. 39830-39846, 2020.

[8] S. Huang and G. Dissanayake, "Robot Localization: An Introduction," *Wiley Encyclopedia of Electrical and Electronics Engineering,* 2016.

[9] E. Kelasidi, S. Moe, K. Y. Pettersen, A. M. Kohl, P. Liljebäck and J. T. Gravdahl, "Path Following, Obstacle Detection and Obstacle Avoidance for Thrusted Underwater Snake Robots," *Frontiers in Robotics and AI,* no. 6:57, 2019.

[10] S. G. Tzafestas, Introduction to Mobile Robot Control, Athens: Elsevier, 2014.

[11] F. Gul, W. Rahiman and S. S. N. Alhady, "A comprehensive study for robot navigation techniques," *Cogent Engineering: Electrical & Electronic Engineering*, vol. 6, no. 1, 2019.

[12] A. Khattab, "Static and Dynamic Path Planning Using Incremental Heuristic Search," Institute of Transportation, German Aerospace Center (DLR), Cologne, 2017.

[13] E. W. DIJKSTRA , "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik*, vol. 1, pp. 269 - 271, 1959.

[14] N. Tyagi, "Dijkstra's Algorithm: The Shortest Path Algorithm," Analytics Steps Infomedia LLP, 15 Februay 2021. [Online]. Available: https://www.analyticssteps.com/blogs/dijkstras-algorithm-shortest-path-algorithm. [Accessed 22 November 2021].

[15] K. Karur, N. Sharma, . C. Dharmatti and . J. E. Siegel, "A Survey of Path Planning Algorithms for Mobile Robots," *Vehicles,* vol. 3, pp. 448 - 468, 2021.

[16] M. Lukic, "Graphs in Python: Dijkstra's Algorithm," Stack Abuse, 20 November 2021. [Online]. Available: https://stackabuse.com/dijkstras-algorithm-in-python/. [Accessed 22 November 2021].

[17] O. A. Gbadamosi and D. R. Aremu, "Design of a Modified Dijkstra's Algorithm for finding alternate routes for shortest-path problems with huge costs.," in *2020 International Conference in Mathematics, Computer Engineering and Computer Science (ICMCECS)*, Nigeria, 2020.

[18] G. Qing, Z. Zheng and X. Yue, "Path-planning of automated guided vehicle based on improved Dijkstra algorithm," in *29th Chinese Control And Decision Conference (CCDC)*, 2017.

[19] P. E. Hart, N. J. Nilsson and B. Raphael, "A formal Basis for the Heuristic Determinatin of Minimum Cost Paths," *IEEE Transactions of Systems, Science and Cybernetics*, vol. 4, no. 2, pp. 100 - 107, 1968.

[20] H. Choset, "Robotic Motion Planning: A* and D* Search," The MIT Press, Boston, 2005.

[21] J. S. Zelek, "Dynamic Path Planning," in *Conference: Systems, Man and Cybernetics 1995 Intelligent Systems for the 21st Century., IEEE International Conference*, 1995.

[22] A. Stentz, "The D* Algorithm for Real-Time Planning of Optimal Traverses," The Robotics Institute, Carnegie Mellon University , Pittsburgh, Pennsylvania, 1994.

[23] F. A. Raheem and U. I. Hameed, "Path Planning Algorithm using D* Heuristic Method Based on PSO in Dynamic Environment," *American Scientific Research Journal for Engineering, Technology, and Sciences (ASRJETS)*, vol. 49, no. 1, pp. 257-271, 2018.

[24] A. O. DJEKOUNE, R. TOUMI and K. ACHOUR, "A Sensor Based Navigation Algorithm for a Mobile Robot using the DVFF Approach," *International Journal of Advanced Robotic Systems*, vol. 6, no. 2, pp. 97-108, 2009.

[25] R. Barber, J. Crespo, C. Gómez, A. C. Hernámdez and M. Galli, "Mobile Robot Navigation in Indoor Environments: Geometric, Topological, and Semantic Navigation," in *Applications of Mobile Robots*, IntechOpen, 2018.

[26] F. Gul, W. Rahiman and S. S. N. Alhady, "A comprehensive study for robot navigation," *Cogent Engineering: Electrical & Electronic Engineering*, pp. 1-25, 2019.

[27] B. K. Patle, G. B. L, A. Pandey, D. R. K. Parhi and A. Jagadeesh, "A review: On path planning strategies for navigation of mobile robot," *Defence Technology*, vol. 15, pp. 582 - 606, 2019.

[28] A. Thondiyath, "Mobile Robot Navigation," NPTEL - NOC IITM, Madras, Tamil Nadu, 2021.

[29] S. M. LaValle, "Rapidly-Exploring Random Trees: A new tool for path planning," *The Annual Research Report*, 1998.

[30] S. J. Kuffner and L. M. Steven, "RRT-Connect: An Efficient Approach to Single-Query Path Planning," in *IEEE International Conference on Robotics and Automation*, Palo Alto, 2000.

[31] P. Zhang, C. Xiong, W. Li, X. Du and C. Zhao, "Path planning for mobile robot based on modified rapidly exploring random tree method and neural network," *International Journal of Advanced Robotic Systems,* vol. 15, no. 3, 2018.

[32] U. A. Okengwu, E. O. Nwachukwu and E. N. Osegi, "Modified Dijkstra Algorithm with Invention Hierarchies Applied to a Conic Graph," Cornell University, 2015.

[33] P. Costa, P. Moreira and P. Costa, "Real-time path planning using a modified A* algorithm," in *ROBOTICA 2009 - 9th Conference on Mobile Robots and Competitions,* Castelo Branco, Portugal, 2009.

[34] F. Duchon, A. Babinec, M. Kajan, P. Beno, M. Florek, T. Fico and L. Jurišica, "Path planning with modified A star algorithm for a mobile robot," *Periodica Engineering - Modelling of Mechanical and Mechatronic Systems MMaMS 2014,* vol. 96, pp. 59-69, 2014.

[35] F. A. Raheema and M. M. Badr, "Development of Modified Path Planning Algorithm Using Artificial Potential Field (APF) Based on PSO for Factors Optimization," *American Scientific Research Journal for Engineering, Technology, and Sciences,* vol. 37, no. 1, pp. 316-328, 2017.

[36] W. Sakpere, M. Adeyeye-Oshin and B. N. Mlitwa, "A State-of-the-Art Survey of Indoor Positioning and Navigation Systems and Technologies," *South African Computer Journal*, vol. 29, no. 3, pp. 145 - 197, 2017.

[37] NovAtel Inc, An Introduction to GNSS - GPS, GLONASS, BeiDou, Galileo and other Global Navigation Satellite Systems, Calgary, Alberta, Canada: NovAtel Inc., 2015.

[38] M. Z. Rahman and S. Jin, "Beyond Trilateration: GPS Positioning Geometry and Analytical Accuracy," in *Global Navigation Satellite Systems - Signal, heory an Applications*, Rijeka, Croatia, InTech, 2012, p. 241.

[39] T. N. Van, D. A. Ngyuyen, C. T. Duc, D.-T. Tran and T. P. Van, "15-State Extended Kalman Filter Design for INS/GPS Navigation System," *Journal of Automation and Control Engineering*, vol. 3, no. 2, pp. 109-114, 2015.

[40] F. A. Faruqi and K. J. Turner, "Extended Kalman filter synthesis for integrated global positioning/inertial navigation systems," *Applied Mathematics and Computation*, vol. 2, no. 3, pp. 213 - 227, 2000.

[41] R. J. Noriega-Manez, "Inertial Navigation," Stanford University, Stanford, CA, 2007.

[42] B. Siciliano and O. Khatib, "Inertial Sensing, GPS anf Odometry," in *Springer Handbook of Robotics*, Springer International Publishing, 2016, pp. 737 - 751.

[43] D. Titterton and J. L. Weston, "Basic principles of strapdown inertial navigation systems," in *Strapdown Inertial Navigation Technology*, London, UK, The Institution of Electrical Engineers, 2004, pp. 17 - 65.

[44] M. Straeten and M. J. Ahamed, "Intuitive ultrasonic INS augmentation for pedestrian path tracking and navigation," *Sensors and Actuators A: Physical*, vol. 299, 2019.

[45] M. Brossard, A. Barrau and S. Bonnabel, "AI-IMU Dead-Reckoning," *IEEE Transactions on Intelligent Vehicles*, vol. 5, no. 4, pp. 585-595, 2020.

[46] A. R. Jimenez, F. Seco and J. Guevara, "A comparison of Pedestrian DeadReckoning algorithms using a lowcost MEMS IMU," in *IEEE International Symposium on Intelligent Signal Processing*, Madrid, Spain, 2009.

[47] I. Amundson and X. D. Koutsoukos, "A Survey on Localization for Mobile Wireless Sensor Networks," in *Mobile Entity Localization and Tracking in GPS-less Environnments*, Springer-Verlag Berlin Heidelberg, 2009, p. 235–254.

[48] W. C. S. S. Simões, G. S. Machado, A. M. A. Sales, M. . M. d. Lucena, N. Jazdi and V. . F. d. Lucena. Jr, "A Review of Technologies and Techniques for Indoor Navigation Systems for the Visually Impaired," *Sensors (Basel)*, vol. 20, no. 14, 2020.

[49] R. M. Hegde and S. Kumar, "A Review of Localization and Tracking Algorithms in Wireless Sensor Networks," *ARXIV,* 2017.

[50] S. A. S. Mohamed, M.-H. Haghbayan, T. Westerlund, J. Heikkonen, H. Tenhunen and J. Plosila, "A Survey on Odometry for Autonomous Navigation Systems," *IEEE Access,* vol. 7, pp. 97466-97486, 2019.

[51] R. Seigwart and I. R. Nourbakhsh, "Mobile Robot Localization," in *Introduction to Autonomous Mobile Robots*, Cambridge, Massachusetts, The MIT Press, 2004, pp. 159-229.

[52] L. Fernández , L. Payá, F. Amorós and O. Reinoso, "A Comparison of Appearance-Based Descriptors in a Visual SLAM Approach," *Encyclopedia of Information Science and Technology,* vol. 3, pp. 3187 - 3196, 2015.

[53] D. Fox, J. Hightower, L. Liao and D. Schulz, "Bayesian Filtering for Location Estimation," *Pervasive Computing,* vol. 03, pp. 1536-1268, 2003.

[54] M. Khalaf-Allah, "A Real-Time Implementation of a Probabilistic Localization Method for Mobile Robots," University of Hannover, Hanover, Germany, 2004.

[55] C. Yang, W. Shi and W. Chen, "Comparison of Unscented and Extended Kalman Filters with Application in Vehicle Navigation," *The Journal of Navigation,* vol. 70, no. 2, pp. 411 - 431, 2016.

[56] D. Bagnell, "Statistical Techniques in Robotics - Occupancy Maps," [Online]. Available: https://www.cs.cmu.edu/~16831-f14/notes/F14/16831_lecture06_agiri_dmcconac_kumarsha_nbhakta.pdf. [Accessed 07 September 2022].

[57] A. Carballo, J. Lambert, A. Cano, D. R. Wong, P. Narksri, Y. Kitsukawa, E. Takeuchi, S. Kato and K. Takeda, "LIBRE: The Multiple 3D LiDAR Dataset," in *2020 IEEE Intelligent Vehicles Symposium*, Tokyo, 2020.

[58] A. Diosi and L. Kleeman, "Advanced Sonar and Laser Range Finder Fusion for Simultanious Localization and Mapping," in *IEEE/RSJ Interanational Conference on Intelligent Robots and Systems*, Sendai, Japan, 2004.

[59] S.-W. Yang and C.-C. Wang, "Dealing with Laser Scan Failure: Mirrors and Windows," in *IEEE International Conference on Robotics and Automation*, Pasadena, CA, USA, 2008.

[60] S.-W. Yang and C.-C. Wang, "On Solving Mirror Reflection in LIDAR Sensing," *IEEE/ASME Transactionals on Mechatonics*, vol. 16, no. 2, pp. 255 - 265, 2011.

[61] B. Peasley and S. Birchfield, "Real-time obstacle detection and avoidance in the presence of specular surfaces using an active 3D sensor," *IEEE Workshop on Robot Vision (WORV)*, pp. 197 - 202, 2013.

[62] P. Foster, Z. Sun, J. J. Parl and B. Kuipers, "VisAGGE: Visible angle grid for glass environments," in *IEEE International Conference on Robotics and Automation*, Ann Arbor, Michigan, 2013.

[63] R. Koch, M. Stefan and A. Nüchter, "Detection and Purging of Specular, Reflective and Transparent Object Influences in 3D Range Measurement," *International archives of the photogrammetry, remote sensing and spatial information sciences*, Vols. XLII - 2, no. W2, pp. 377 - 384, 2017.

[64] R. Koch, M. Stefan and A. Nüchter, "Identification of transparent and specular reflective material in laser scans to discriminate affected measurements for faultless robotic SLAM," *Robotics and Autonomous Systems*, vol. 87, no. 1, pp. 296-312, 2017.

[65] X. Zhao, Z. Yang and S. Schwertfeger, "Mapping with Reflection - Detection and Utilization of Reflection in 3D Lidar Scans," in *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Abu Dhabi, UAE, 2020.

[66] H. Tibebu, J. Roche, V. De Silva and A. Kondoz, "LiDAR-Based Glass Detection for Improved Occupancy Grid Mapping," *Sensors*, vol. 21, no. 7, 2021.

[67] Y. Pyo, H. Cho, R. Jung and T. Lim, "Chapter 2 - Robot Operating System ROS," in *ROS Robot Programming*, Seoul, Republic of Korea, ROBOTIS Co., Ltd., 2017, pp. 10 - 20.

[68] Open Robotics, "Distributions," Open Robotics, 15 May 2021. [Online]. Available: http://wiki.ros.org/Distributions. [Accessed 8 July 2022].

[69] Open Robotics, "The ROS Ecosystem," Open Robotics, 2021. [Online]. Available: https://www.ros.org/blog/ecosystem/. [Accessed 07 July 2022].

[70] L. Joseph and J. Cacace, "Introduction to ROS," in *Mastering ROS for Robotics Programming*, Packt, 2018.

[71] K. Bipin, "ROS Architecture and Concepts I," in *Robot Operating System Cookbook*, Packt, June 2018.

[72] Y. Pyo, H. Cho, R. Jung and T. Lim, "Important Concepts of ROS," in *ROS Robot Programming*, Seoul, Republic of Korea, ROBOTIS Co.,Ltd., 2017, pp. 41 - 89.

[73] Y. Pyo, H. Cho, R. Jung and T. Lim, "Mobile Robots," in *ROS Robot Programming*, Seoul, Republic of Korea, ROBOTIS Co.,Ltd., 2017, pp. 279 - 308.

[74] Open Robotics, "Gazebo," Open Robotics, 2022. [Online]. Available: https://gazebosim.org/. [Accessed July 2022].

[75] ROS Wiki, "rviz," ROS.org, 2018. [Online]. Available: https://wiki.ros.org/rviz. [Accessed July 2022].

[76] Robotis, "Turtlebot 3," Robotis, 2022. [Online]. Available: https://www.robotis.us/turtlebot-3/. [Accessed July 2022].

[77] Open Source Robotics Foundation, Inc., "TurtleBot," Open Source Robotics Foundation, Inc., [Online]. Available: https://www.turtlebot.com/. [Accessed July 2022].

[78] Robotis, "Turtlebot 3 - Features," Robotis, 2022. [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/features/#specifications. [Accessed July 2022].

[79] Raspberry Pi, "Raspberry Pi 3 Model B," Raspberry Pi, 2022. [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-3-model-b/. [Accessed July 2022].

[80] Robotis, "360 Laser Distance Sensor LDS-01 (LIDAR)," Robotis, 2022. [Online]. Available: https://www.robotis.us/360-laser-distance-sensor-lds-01-lidar/. [Accessed July 2022].

[81] Robotis, "XM430-W210," Robotis, 2022. [Online]. Available: https://emanual.robotis.com/docs/en/dxl/x/xm430-w210/#specifications. [Accessed July 2022].

[82] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," The Internet Society, 2006.

[83] ROS Wiki, "roscore," ROS.org, 2019. [Online]. Available: http://wiki.ros.org/roscore. [Accessed 2022].

[84] ROSIN Ljubljana ROS Summer School, "ROS Basics," GitHub, 2020. [Online]. Available: https://web.fs.uni-lj.si/lampa/rosin/ROS%20Summer%20School/Day%201/ros_intro/. [Accessed July 2022].

[85] ROS Wiki, "tf," ROS.org, 2020. [Online]. Available: http://wiki.ros.org/tf. [Accessed July 2022].

[86] X. Zhan, J. Lai, H. Li and M. Fu, "2D Lidar-Based SLAM and Path Planning for Indoor Rescue Using Mobile Robots," *Journal of Advanced Transportation,* vol. 8867937, p. 14, 2020.

[87] T. Mitchell, Machine Learning, McGraw Hill, 1997.

[88] T. . O. Ayodele , "Types of Machine Learning Algorithms," in *New Advances in Machine Learning*, InTech, 2018, pp. 19-48.

[89] D. Xu and Y. Tian, "A Comprehensive Survey of Clustering Algorithms," *Annual Data Science,* vol. 2, no. 2, p. 165–193, 2015.

[90] M. J. Garbade, "Understanding K-means Clustering in Machine Learning," Towards Data Science, September 2018. [Online]. Available: https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1. [Accessed July 2022].

[91] scikit-learn, "Comparing different clustering algorithms on toy datasets," scikit-learn, 2022. [Online]. Available: https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html. [Accessed July 2022].

[92] S. Schwertfeger, X. Zhao and Z. Yang, "Mapping with Reflection - Detection and Utilization of Reflection in 3D Lidar Scans," in *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR),* Abu Dhabi, UAE, 2020.

[93] S. Schwertfeger, X. Zhao and Z. Yang, "Mapping with Reflection - Detection and Utilization of Reflection in 3D Lidar Scans," in *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR),* Abu Dhabi, UAE, 2020.

[94] Z. Yang, X. Zhao and S. Schwertfeger, "Mapping with Reflection - Detection and Utilization of Reflection in 3D Lidar Scans," in *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR),* Abu Dhabi, UAE, 2020.

APPENDICES

Appendix A

Bayes Theorem

Where P(A) is the probability of event C without any effect on any other event. P(B) is the probability of event B without any effect on any other event, and P(A|B) is the probability of event B given that event A is true, providing a probability between 0-1.

$$P(A|B) = \frac{P(A \mid B)P(A)}{P(B)}$$

The following figure shows different probabilistic representations:

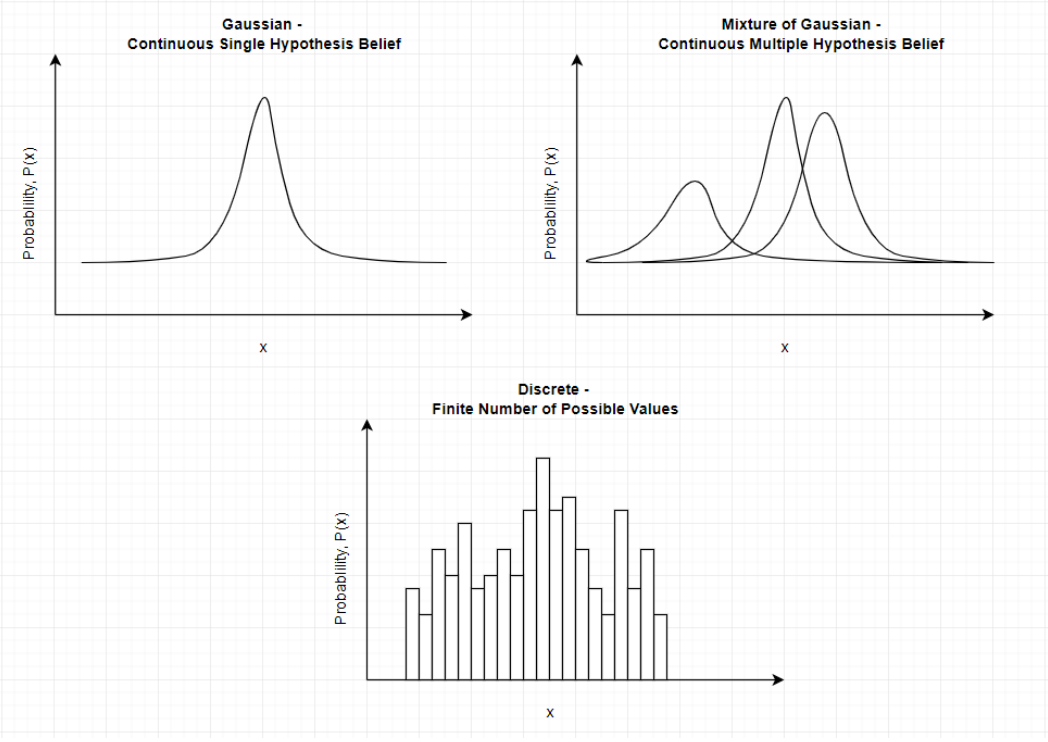

Figure 64: Probabilistic representations for hypothesis beliefs - Gaussian and discrete distribution

Appendix B

Steps for setting up the remote PC and Turtlebot.

1. Download preferred Ubuntu package on Remote PC
    - In this thesis, Ubuntu 16.04 was used because it was the most stable.

2. Install preferred ROS version on remote PC
    - In this thesis, ROS 1 was used to be compatible with Ubuntu 16.04

3. Install all dependent ROS packages, which includes several packages:
    - Teleoperation-related files
    - Laser Scan perception Packages
    - Image Perception and Transport Packages
    - Launch files for RGB-D devices
    - Arduino Packages compatible with ROS
    - Python Packages compatible with ROS
    - ROS Serial Packages
        i. rosserial server
        ii. roserial client
        iii. rosserial messages
    - AMCL (adaptive Monte Carlo localization) packages
    - Map-related packages (Server/Saver)
    - URDF (unified visual robot model) packages
    - Xacro (XML macro language) Packages
    - Rqt-related packages (rqt is for GUI plugins)
    - Navigation packages
    - Interactive Marker Packages (used in tools such as RViz)

4. Install Turtlebot Packages and configure system to match the model being used.
    - In this thesis, the Turtlebot 3 Waffle-Pi was used
5. Configure the network for the remote PC
    - The remote PC is the ROS Master
    - IP URI should be in the form of: https://IP_remotePC:11311
    - The IP address can be found by:

```
$ ifconfig
```

    - Update ROS IP settings

```
$ nano ~/.bashrc
```

    - Source bashrc with

```
$ source ~/.bashrc
```

6. On the Raspberry Pi, download the correct ROS image and burn the image
    - Here, Raspberry Pi 3 is used, and Raspberry Pi 4 onward is no longer compatible with Ubuntu 16.04
7. Configure the network for the Raspberry Pi
    - The Raspberry Pi of the Turtlebot is the host
    - IP URI should be in the form of: https://IP_remotePC:11311
    - The IP address can be found by:

```
$ ifconfig
```

    - Update ROS IP settings

```
$ nano ~/.bashrc
```

    - Source bashrc with

```
$ source ~/.bashrc
```

8. Install OpenCR packages onto the Raspberry Pi
9. Upload Firmware to OpenCR

Appendix C

Turtlebot/Ros nodes and topics in use

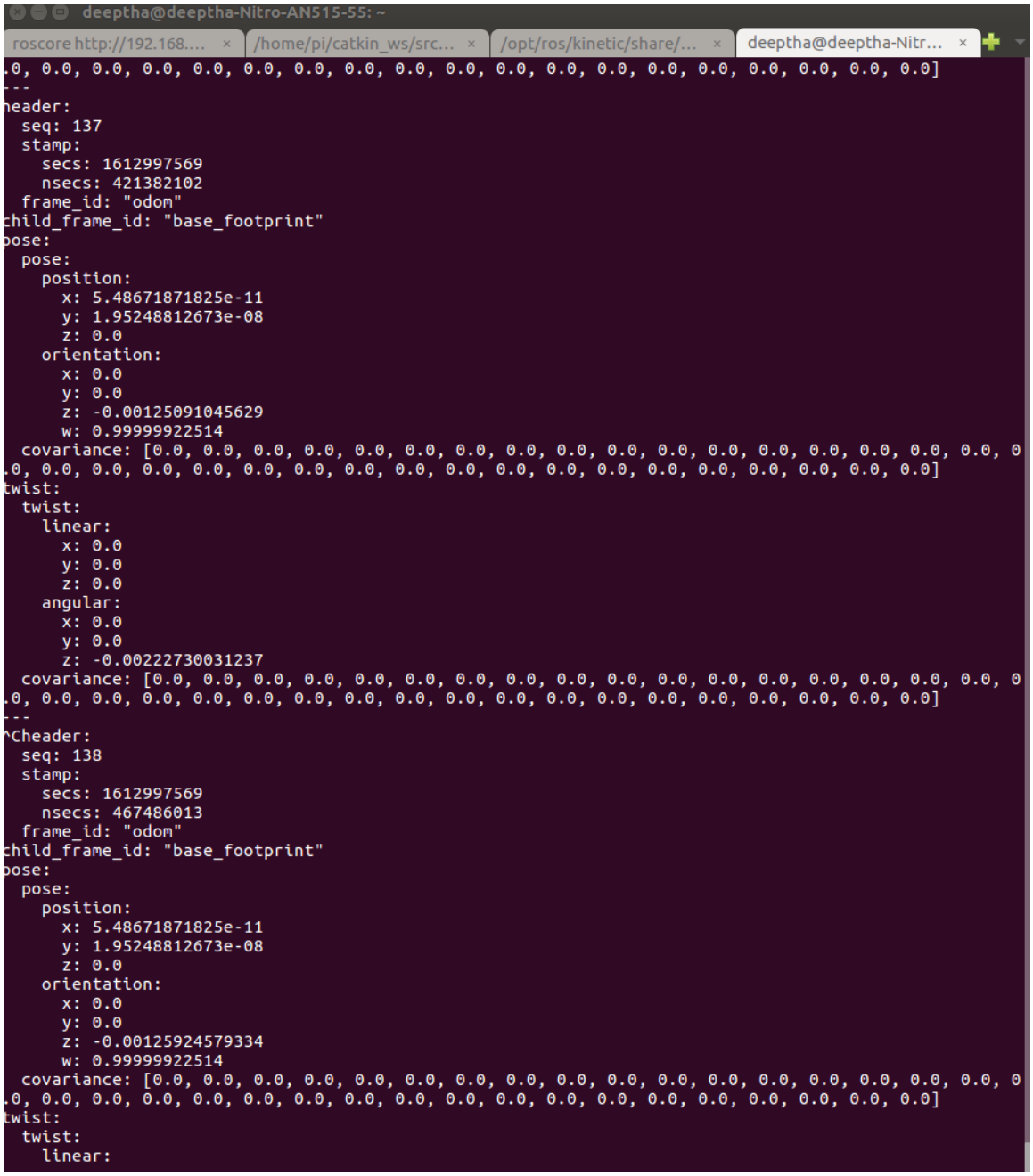

Figure 65: SLAM node data collection as seen in the Ubuntu command window

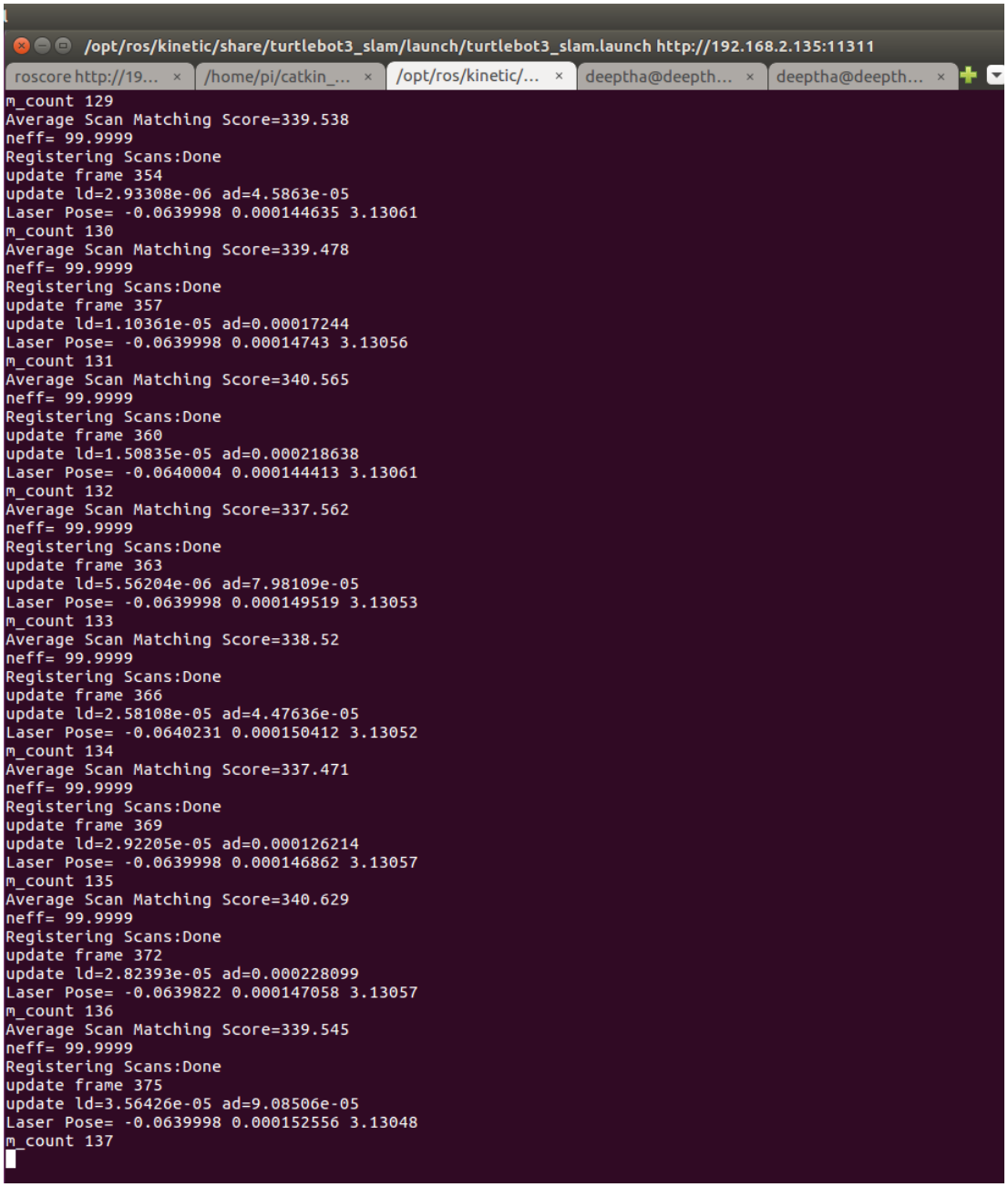

Figure 66: Odometry-position estimation as seen in the Ubunty command window during SLAM

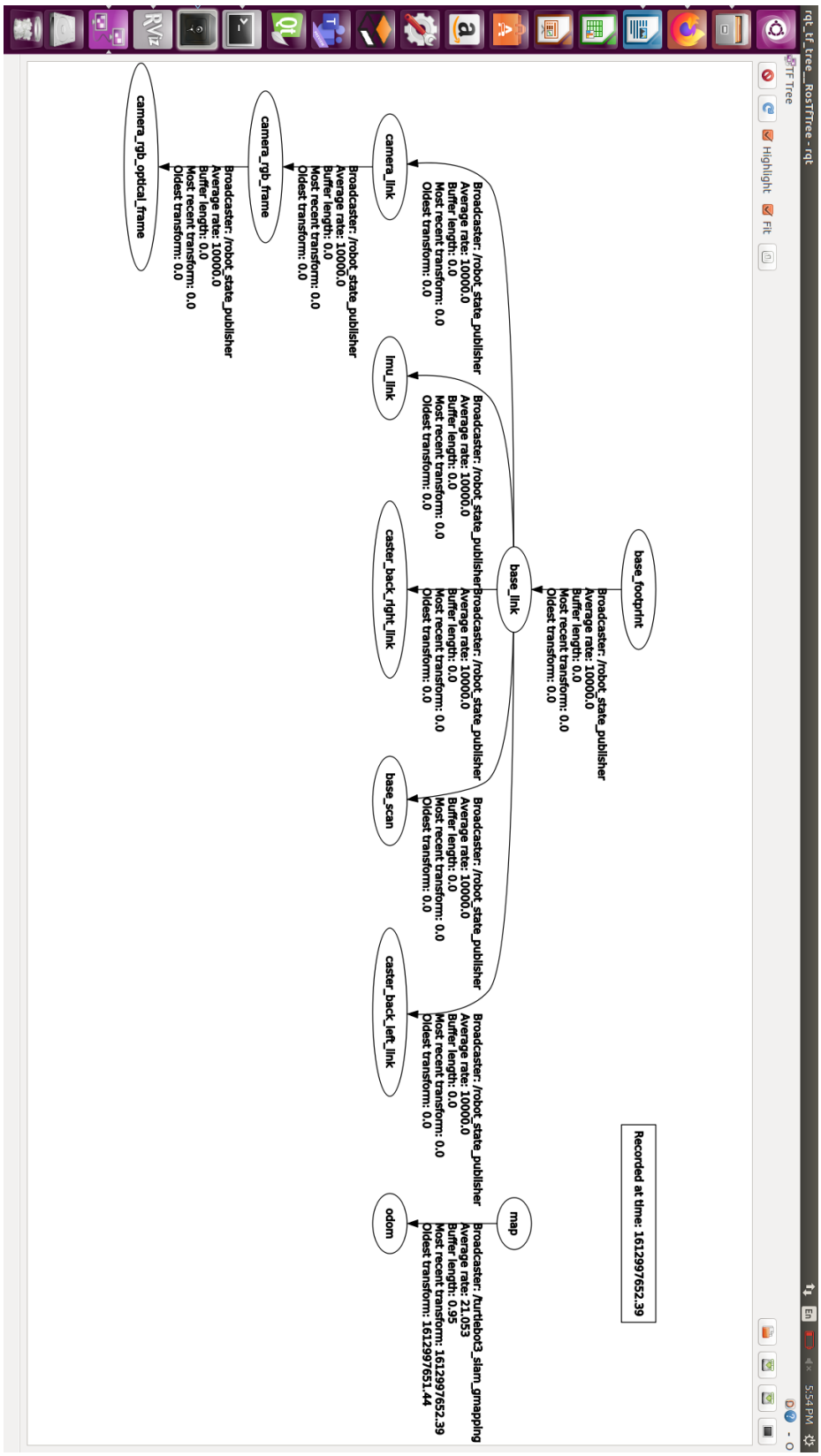

Figure 67: ROS RQT Map retrieved

Appendix D

This image shows how various unsupervised clustering algorithms work.

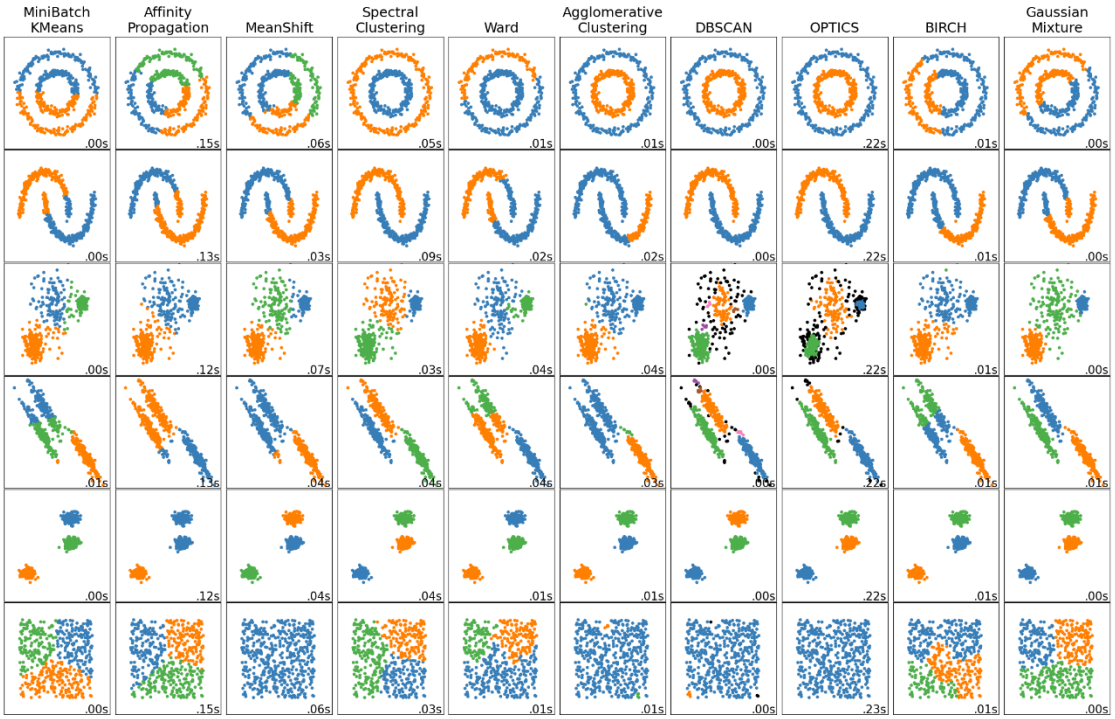

Figure 68: Clustering for various algorithms [91]

VITA AUCTORIS

NAME:	Deeptha Damodaran
PLACE OF BIRTH:	Ottawa, ON, Canada
YEAR OF BIRTH:	1996
EDUCATION:	B.A.Sc. Mechanical Engineering with Automotive option University of Windsor Windsor, Ontario 2015 - 2018
	M.A.Sc. Mechanical Engineering University of Windsor Windsor, Ontario 2019 - 2022