2-1-2016

A hybrid approach to simultaneous localization and mapping in indoors environment

Amr Morssy

The American University in Cairo

School of Sciences and Engineering

A Hybrid Approach To Simultaneous Localization And Mapping In Indoors Environment

By

Amr B. Morssy

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science in Robotics Control and Smart Systems

Under supervision of:

Dr. Maki Habib

Professor, Department of Mechanical Engineering

July, 2016

Declaration

I declare that this thesis is my work and that all used references have been cited and properly indicated.

Amr Morssy

ACKNOWLEDGMENTS

First I praise God Almighty for granting me the capability to complete this work and for facilitating it for me.

Then I would like to thank my supervisor Dr Maki Habib for his generous support and guidance as well as his comments and time to make my work better. I greatly appreciate the time he spent with me, and his genuine patience with me.

I would like to thank my parents for their support and words as well as my beloved grandmother whose presence in my life encouraged me a lot for work before she passed away.

I would like to thank the lab technician Mr. Mohammed, and the previous Lab engineer Mr. Mohammed Shalaby for their support and help with the setup of the robot for experiments. Many thanks to my friends Moatez, Amr Henawey, and Ahmed Samir for their help and the resources they provided me with to better understand probability theory and programming.

Abstract

Simultaneous localization and mapping (SLAM) is a demanding and important requirement for truly autonomous robots. The ability to localize in an unknown environment enables the robot to operate autonomously. Many problems need to be tackled before the goal of SLAM is achieved. Moreover, added to the many challenges in the field is the problem that robots have limited computational resources and limited sensors. Furthermore, many sensors may suffer from degraded performance in different situations in the environment.

The environment itself can be a challenge for SLAM as it may include moving (dynamic) objects of various shapes and irregular velocity profiles, changes in semi static structures (chairs and tables indoors), changes in illumination in case of camera, and ambiguity that could result from wrongly identifying landmarks. For this reason, many researchers limit their approaches to simple scenarios or to situations where only a subset of the problem is solved. The environment could be modified to overcome the ambiguity of landmarks or dynamic objects could be excluded to simplify the problem. Thus more research and work is needed to lift those assumptions to enable truly autonomous robots. There are many techniques and solutions used to face the mentioned challenges such are sensor fusion, modeling the environment, data association and use of bioinspired models.

This thesis will present SLAM in the current literature to benefit from then it will present the investigation results for a hybrid approach used where different algorithms using laser, sonar, and camera sensors were tested and compared. The contribution of this thesis is the development of a hybrid approach for SLAM that uses different sensors and where different factors are taken into consideration such as dynamic objects, and the development of a scalable grid map model with new sensors models for real time update of the map. This is challenging since the interaction between modular algorithms changes when they are integrated in a full system with other algorithms due to data type modification and timing. The thesis will show the success found, difficulties faced and limitations of the algorithms developed which were simulated and experimentally tested in an indoors environment. From the simulations and experiments it was concluded that the hybrid approach is able to operate in an indoors environment and build a scalable map.

List of figures

List of Tables

List of Acronyms

ARCOS Advanced Robot control and operations software

DATMO Detection and tracking of moving objects

EKF Extended Kalman Filter

FOE Focus of expansion

GA Genetic Algorithm

ICP Iterative closest point

RANSAC Random Sample Consensus algorithm

SLAM Simultaneous localization and mapping

TBF Triangulation based fusion

UKF Unscented Kalman filter

List of symbols

b Robot wheel base distance

c Grid cell

d Distance between robot position at two different sensor readings

m Map

MOT Moving object tracking

p(a) Probability of event a

P2P Point to point scan matching

Q Measurement noise covariance matrix

q_k Laser scan at time k

R Maximum sensor range

R Model noise covariance matrix

r Sensor reading range

S samples

SSE Sum of squared errors

T Edge

u Control signal

v Dynamic model noise

w Measurement noise

W Weight of subsumption layer output

w_{σ_0} UKF sigma point weight

X_r Robot state

x_t System state at time t

z Measurement

α Odometry drift angle

β sonar beam half angle

μ Mean

Σ Covariance matrix

θ Robot orientation

λ UKF scaling factor

Contents

Chapter 1

1.1 Introduction

In this chapter the objectives and motivation behind this thesis will be presented. Further, the foundation of simultaneous localization and mapping (SLAM) in mobile robots will be introduced together with the different approaches and methods. Finally, the thesis outline will be presented.

1.2 Motivation and objectives

The objectives of this thesis are:

- The development and testing of a SLAM system for indoors environment
- Adopting a hybrid approach using grid maps with option for scalability
- Investigate and benefit from sensor fusion to benefit from the advantages of sensors
- Detection of dynamic objects and exclusion from the map via tracking

In details, the objective of this thesis is to investigate the possibility of developing a hybrid SLAM system that will address some of the research challenges in SLAM such as scalability problem, sensor fusion, bounding odometry error, and dynamic objects extraction. The SLAM system to be developed will be used in indoors environment extracting features in structured environments. In this thesis sensor fusion will be used to enable the robot to integrate information from sonar, laser and camera sensors. The image processing will be carried out using openCV software with the goal of detecting dynamic objects in images, and carrying out obstacle avoidance.

Further, the proposed approach will be tested for each algorithm individually before applying it in combined form to reflect the goals and see the effects and difficulties that arise from integrating the algorithms. The system will be developed and will implement scalable mapping techniques to enhance its ability to operate in larger environment. Moreover, the proposed method will include the extraction of objects in a dynamic environment where dynamic objects which otherwise create a distraction that appears as an increase in uncertainty will be integrated in the map or rejected according to the nature of object. The dynamic objects will be identified and tracked for exclusion from the static map. The goal is to produce a final global map in an indoor environment and compare to the actual environment layout.

1.3 SLAM

1.3.1 History of SLAM

Autonomous navigation and mapping is an essential prerequisite for service robots to work efficiently [7, 73]. Precise estimation of robot pose and map of the environment is needed for successful navigation. The process by which this task is carried out is called SLAM [1, 2, 8]. The solution to the simultaneous localization and mapping (SLAM) problem with certain assumptions has been one of the greatest successes of the robotics community over the past years. History of SLAM can be traced back to Gauss who formulated the least squares method [1, 72, 74, 75]. Probabilistic SLAM was first adopted as the robotics and artificial intelligence community were adopting probabilistic methods to enhance machine intelligence, and the probabilistic methods became the standard in SLAM problem solution with the extended Kalman filter receiving most attention [2, 75, 82].

Later it was found that the states of landmarks are correlated and as the robot moves around it will have to build a big state vector making it computationally expensive to update [3, 77]. This was the scalability problem which limited probabilistic mapping to small environments. Further, it was shown that the correlation of landmarks prevents from discarding low value elements of covariance matrix as the whole matrix is needed for convergence due to landmark correlation [77]. However, SLAM is convergent and a probabilistic solution is possible [1, 2, 4]. Research included indoor environments, outdoor environments and undersea environment, loop closure, scalability problems, various sensors as well as the use of real environment data to investigate the effect of data association [72, 74, 76].

1.3.2 Probabilistic SLAM

The aim of SLAM is to determine the robot pose in an unknown environment while at the same time building up a map of that environment [8]. SLAM depends greatly on the use of exterioceptive sensors such as laser ranging sensors, or sonar sensors, or camera which can be a mono camera or a stereotype camera [8]. In many cases some form of sensor fusion is employed to increase the robustness of the SLAM process as it enhances the result of SLAM through combining the results from different sensors as sensor fusion overcomes the limitations of each sensor [9].

There is a set of tasks that need to be carried out by robots to work autonomously in an environment as shown in figure 1. From this figure the two elements needed for SLAM are localization and mapping [10, 82].

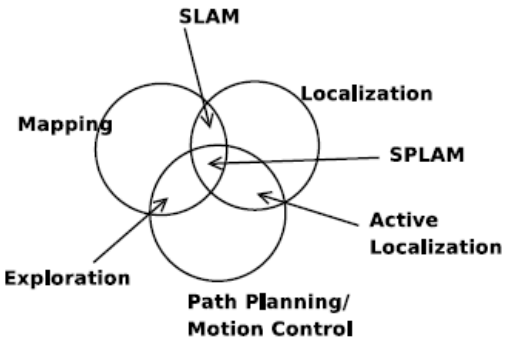


Figure 1: SLAM is a subset of the tasks that need to be carried out by autonomous robots [10].

Theoretically, the robot state changes by a process model that relates the state at time t+1 to the state at time t by:

$$\boldsymbol{x_r(t+1) = f\big(x_r(t), u(t), v(t)\big)} \quad \textbf{(1)}$$

Where $x_r$is the robot state,$u(t)$ and $v(t)$ are the control signal and noise at time t respectively. It is accepted that some assumption is made about the initial robot pose that it is well known [11, 77]. Therefore, the pose of the robot can be tracked and observed via the process model but with increasing uncertainty so sensor update is needed to limit uncertainty.

A graphical model can be made to describe the dependencies between measurements, robot path, and control input in SLAM. This is shown in figure 2.

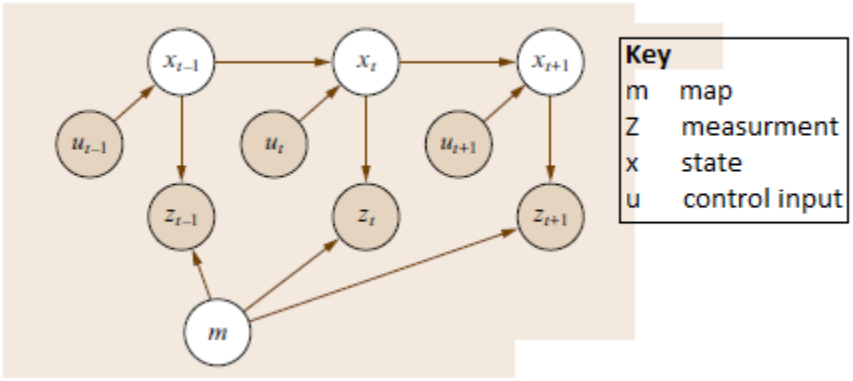


Figure 2: The graphical model of SLAM shows the dependencies in SLAM [1].

The states of the robot are represented by *x*, the odometry readings are represented by *u* also called the control input, and the measurements are symbolized by *z* and originate from the static map *m* [1, 11]. Probabilistic SLAM tries to find the robot state *x*, and map *m* probabilistically from the inputs *z* and *u.* This chain can be represented by the probabilistic posterior over the robot path and map $P(x_{1:t}, m | z_{1:t}, u_{1:t})$ which is also known as online SLAM where the path is not recovered. Algorithms that solve the online SLAM problem are incremental and hence are called filters [1].

1.4 SLAM frameworks

SLAM frameworks refers to the different SLAM paradigms from which different algorithms have been derived. Three different paradigms built on filters are described by [1]. Since the estimation filters used rely on models for the environment and the process, the probabilistic models will be discussed before the different paradigms.

1.4.1 Probabilistic models

For probabilistic SLAM, the process model is assumed to be a probabilistic state transition since the robot motion in the environment is not perfect and includes process noise. Therefore, the process model is given by the state transition probability:

$$\boldsymbol{P(x_t | x_{t-1}, u_t)} \quad \textbf{(2)}$$

The state transition probability is assumed to obey the Markov assumption [2].The observation or senor model is used which describes the probability of making an observation given the current map with robot pose. This is described as:

$$\boldsymbol{P(z_t | x_t, m)} \quad \textbf{(3)}$$

The SLAM process then becomes a two stage process. The first is the time update of the state and the second is the measurement update [2]. The measurement update is based on the observation or sensor model. Some of the different SLAM paradigms using different filters are discussed next.

1.4.2. Extended Kalman Filter (EKF) SLAM

i. Formulation of EKF SLAM

EKF is used in SLAM as opposed to KF since it is able to handle non-linear process. In EKF SLAM the state of the robot and the environment are represented as a single state vector with a

covariance matrix that describes the uncertainty in the state vector as well as the correlations between the landmarks. As new features are discovered, they are added to the state vector and the covariance matrix increases in size in a quadratic manner and the extended Kalman filter is used to update the state vector and the covariance matrix [1, 73, 78]. In this approach, the state of the environment is represented by features that are two dimensional in appropriate domains. This means that for N features, the size of the state vector is 3+2N since three variables are needed to describe robot pose (position in 2D coordinates and orientation) and two per landmark in a 2D environment [1, 77, 78]. Essentially, the extended Kalman filter represents the state of the robot and the environment as a multivariate Gaussian distribution of mean μ and covariance matrix Σ.

The probabilistic state transition model is related to the dynamic model therefore:

$$\boldsymbol{P(x_t|x_{t-1}, u_t)} \;\leftrightarrow\; \boldsymbol{x_t = f(x_{t-1}, u_t) + w_t} \quad \textbf{(4)}$$

The function f models the robot kinematics and could be non-linear in nature while the variable w_t is the additive zero mean Gaussian noise with covariance Q_t. Similarly, the observation probability is expressed as a measurement function which is dependent on the state and the environment. Therefore,

$$\boldsymbol{P(z_t|x_t, m)} \;\leftrightarrow\; \boldsymbol{z_t = h(x_t, m) + v_t} \quad \textbf{(5)}$$

Where v_t is the zero mean Gaussian noise with covariance R_t and h is the measurement function relating measurements to the geometry of the environment [2]. Those two probabilistic models are used by the EKF in its processes to update the robot state.

ii. Problems with the EKF SLAM

An important issue in EKF SLAM is the problem of data association. Data association is about confirming the identity of the current landmark whether it is new or has been observed before. EKF is sensitive to wrong data association and this can lead to errors which can cause failure. Accordingly, any newly detected landmarks are compared to previous ones based on a distance criterion which explains how the current feature's position compares with the position of a previously found feature [1]. Further, some approaches keep track of the number of times a landmark has been observed before adding it to the map if the number of times seen exceeds a certain threshold to minimize the effect of wrong data association [77, 1].

Another problem is the large state covariance matrix that has to be maintained and inverted due to off diagonal correlations between landmark positions.

1.4.3. Particle filters SLAM methods

SLAM methods that rely on particle filters have been introduced such as the FastSLAM algorithm which was the first to directly represent non linearity and non-Gaussian distributions. Particle filter is an implementation of Bayes probabilistic filter where the state of the system is represented as multi hypotheses with weights assigned to each [2, 82]. The hypotheses are represented as a set, S, of N weighted states (pose and map) [5]. Thus:

$$\boldsymbol{S = \{s^i, w^i\}\ i = 1, 2, \ldots, N} \quad \textbf{(6)}$$

The weights of sets sum up to one. The particle filter works by drawing samples for the distribution it should represent. The more samples chosen, the more accurate the representation is. Figure 3 shows the sampling of two distributions with function shown and the samples drawn are shown as vertical lines under it. As can be seen, it is possible to sample multi-modal and non-Gaussian distributions. The left function in figure 3 is a Gaussian function while the function on the right is not. The modeling of non-Gaussian functions is an advantage compared to Kalman filters [5].

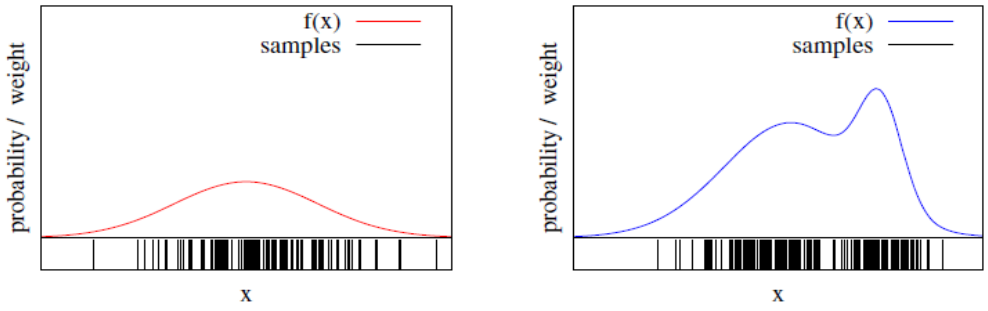


Figure 3: An example of two functions and their representation (black lines) by particle sampling [5].

When using the particle filter resampling is needed to introduce new particles but this should be done carefully and when needed to avoid particle depletion [5]. The problem with particle filters in original form is that the space of SLAM is large and the number of particles needed to represent such a space will grow exponentially with space size as the number of features of increases so to overcome this problem, fastSLAM uses a set of particles, S, to represent the robot state at a particular time instant, t, where each particle maintains N two dimensional

Gaussians to represent N landmarks. FastSLAM that uses particle filters has many advantages compared to other EKF based approaches. The nature of the particle filter enables multi-hypotheses representation which greatly aids to overcome errors in data association. Further, particle filter can be very efficient to compute with its SN two dimensional Gaussians compared to the N^2 Gaussian of the EKF [1].

1.4.4. RatSLAM

i. Introduction to RatSLAM

RatSLAM is a SLAM paradigm that is based on biologically inspired methods based on experimental evidence that rodents use specialized place cells and grid cells for navigation that are affected by animal position [13]. Since then, work has been done to apply the findings in a new bioinspired SLAM technique that has been called ratSLAM [13, 14, 15].

ii. ratSLAM Algorithm

The ratSLAM system is based on two attractor neural networks. The robot pose is represented by activity packets in the network. Once visual stimulus is associated with pose, it can be used to modulate the activity packets. Using this approach allows for topological representation and multiple hypothesis in the presence of ambiguous visual input as the activity packets will compete with one another until visual perception strengthens one of them [14]. The system architecture is shown in figure 4. The information from odometry is integrated to form the path and injects packets into the pose cells in order to modify the pose of the robot based on odometry. Further, visual cues if already associated with a pose will inject packets into the pose cells.

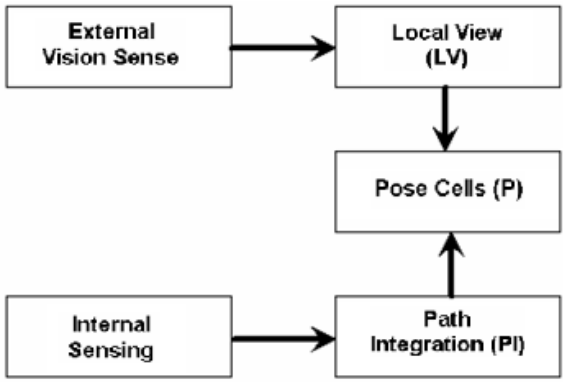


Figure 4: The system architecture of the ratSLAM system [14].

Pose cells represent pose without separation since the separation of orientation and location in two competitive attractor networks will fail to represent multiple hypothesis properly [14, 15]. To link visual input to pose, local view cells are used. These are activated whenever an obstacle with certain features is encountered.

The visual and path integration effects are the only modifiers of network activity in pose cells.

iii. Experience maps

RatSLAM also features experience maps that represent the environment in topological manner using "experiences" as nodes and the transition between them as links [12]. Experiences represent a snapshot of the local view and pose in a particular location in space and each experience is affected by a zone of association which make it fire when the cells in the local view and pose cells are closest to.

Experience maps can be useful for human robot interaction and for the robot to take orders from human agents. Milford et al [12] describe a system where linear neural networks were used to correlate places in the experience map with room names to enable communication between the human operator and robot.

iv. Comparison between probabilistic SLAM and ratSLAM

In ratSLAM the robot pose is represented by the activity packets in the attractor neural networks which can be multiple in cases of multiple hypotheses. In probabilistic SLAM however, the pose is represented by the state vector and associated covariance. In the prediction step of probabilistic SLAM, uncertainty of the root pose is increased due to robot motion while in ratSLAM the activity packet is only shifted according to the robot motion without adding any uncertainty. Landmarks in probabilistic SLAM are used to reduce state uncertainty multiplicatively while in ratSLAM they serve as global references that inject activities to modify the state of the network in an additive fashion. Thus in ratSLAM, inactive cells could become strongly activated by landmarks [13]

1.4.5. Vision SLAM

Vision SLAM is SLAM that relies only on vision for localization, and mapping making it challenging compared to other approaches. There are many challenges imposed by vision only SLAM. First, the input rate is very high and the very rich amount of data to deal with is large

which makes real time SLAM using all data hard to achieve. Further, there are no long term features to track for consistent SLAM [8, 16]. Furthermore, there is lack of depth measurement in vision SLAM and therefore only few successful vision only SLAM systems are known [16]. It must be noted that stereo camera offer the capability to estimate depth unlike mono cameras.

According to [16] a reasonable visual SLAM system is possible using a sparse set of landmarks provided that they are chosen carefully. Alternatively, high performance techniques such as scale invariant feature transform (SIFT) features could be used for SLAM since these provide high accuracy localization due to their success in object recognition but these are away from being real time due to high computational power needed.

The performance of vision SLAM can be enhanced by sensor fusion and the paper by [9] proposes a new method which fuses sonar data with CCD camera where information fusion occurs at the level of features. This technique improves the reliability and precision of the environment observations used for the SLAM problem [9].

Sometimes optical flow is used by vision SLAM and its computation relies on the comparison of two frames and finding corresponding pixels in the two frames. Assumptions are made such as a constant image brightness, and smoothness of pixel motion [41]. In dense optical flow methods such as Horn and Schunk as well as Nordberg and Farneback methods, an optical window of size of m by n is considered to estimate the optical flow at its center point. Dense optical flow can easily show the presence of dynamic objects in a frame [41].

Sparse optical flow on the other hand, such as the Lucas and Kanade algorithm extracts points of interest from the images before computing optical flow. Those points of interests can be extracted using Harris corner detector for example and are then tracked from the current frame to the next for computing optical flow [41]. Optical flow in the case of a stationary camera can be very useful for detecting dynamic objects in the scene with ease by comparing pixels. However, the situation becomes significantly challenging when the camera is moving since the background acquires motion opposite to that of the camera [45].

1.5 Mapping approaches

Maps are used to represent the environment in which the robot navigates and operates [5]. Navigation is an important ability that is needed for the operation and survival of an autonomous robot in a real world environment [17, 82]. The random movement of a robot in an environment may not be sufficient for its operation or it may not be the most efficient. It can be improved by relying on sensors and simple reflex actions to avoid obstacles. However, the performance and range of tasks that can be performed by robot will greatly be increased if the robot can navigate intelligently while keeping an internal representation of the environment called a map [17]. It is a requirement to map the environment while localizing within it simultaneously so the use of maps is important for environment representation and is an outcome of SLAM [18].

Classically, there have been two map types which are metric and topological maps. In metric maps the location of obstacles and the structure of the environment are stored with respect to common reference frame and in topological mapping the locations that the robot can reach and discriminate based on features are stored together with their position information and topology [17, 82]. Figure 5 shows the metric versus topological representations for the same environment. It is common to implement metric maps using grids that discretize the environment [5].

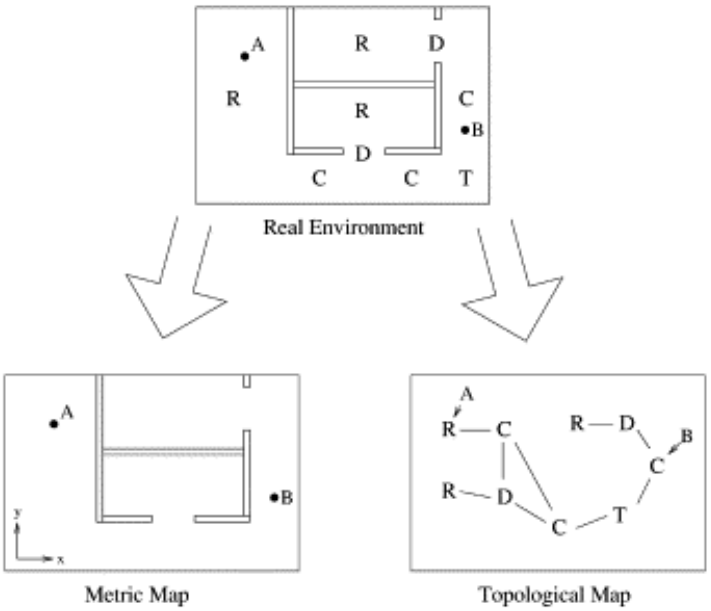


Figure 5: The difference between topological and metric representations of the same environment [5].

1.5.1. Grid map

Grid maps discretize the environment into cells which can be according to the geometry of the environment and could result in varying cell sizes being very compact as the large empty areas are one space or can be discretized into fixed sized cells [20]. Each cell holds the probability that this region of space is occupied by an obstacle regardless of its shape. An important assumption is that the value in one grid cell is independent of the value in other cells. Thus the probability of the grid map P(m) is simply the product of the occupancy probabilities of individual cells P(c).

$$\boldsymbol{P(m) = \prod p(c)} \qquad \textbf{(7)}$$

Assuming a stream of sensor readings, $z_{1:t}$, and history of robot motion (pose), $x_{1:t}$, and applying the Markov assumption) an update equation can be found based on Bayes theorem:

$$\boldsymbol{P(c|x_t, z_t) = \frac{P(z_t|c,x_t)P(c|x_t,z_{t-1})}{P(z_t|x_t,z_{t-1})}} \qquad \textbf{(8)}$$

This equation can be further modified to reach a more efficient odds representation as shown in page 15-16 of reference [5]. The term $P(z_t|c, x_t)$ describes a sensor model which depends on the type of sensor used. A commonly used sensor model of the Polaroid sonar sensor is given by reference [19] shown in figure 6. Consider the sonar to produce a beam with a half angle of β which is superimposed on the grid map as shown in figure 6. The beam can be divided into three regions along its length these are: Region I where the cells lying there are likely to be occupied and region II where the cells are likely to be empty. Region III has an unknown status hence it is excluded from the update.

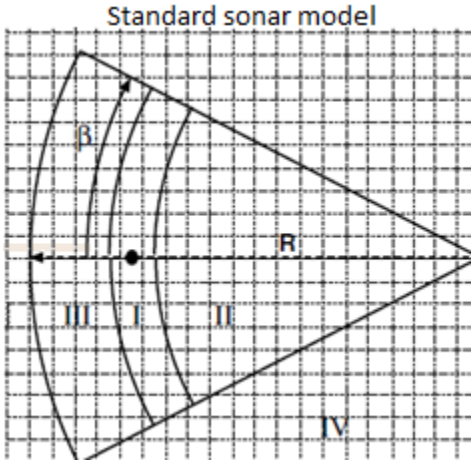


Figure 6: The sonar model used to build grid maps [19].

Grid cells can also be built using the Dempster-Shafer theory where certainty is represented by belief functions. The belief function enables the sensor to express the case of limited knowledge about the environment it has a value of 1 but it is distributed between three elements. This means the sate of a cell can be occupied, or empty, or unknown. The update methods and operation of Dempster-Shafer theory based grid cells is explained in details in chapter 11 of reference [19].

The simplest way of creating a grid map is by the incremental scheme approach where the robot position is first estimated in the local map and the map is built around the estimate. Therefore, the robot estimates the map locally as it builds it [17].

In case of the grid map, the nearest unexplored cell can be used as a guidance for the robot to move to. In this case, the robot will move so as to explore the environment quickly. Furthermore, laser scans can be used instead of sonar scan to enhance the map accuracy.

Other forms of grid map updates use techniques like the histogram in motion mapping where the cells are given a score and the higher the score the more likely is the cell to be occupied. Another decomposition method is to use a variable resolution map where the world is started as an empty single cell and as obstacles are discovered, the space around the obstacle is refined to a cell that surrounds the obstacle [17].

1.5.2 Topological map

It has been proven that using an all metric global approach to map building is not feasible due to high computational cost. Therefore topological maps were developed that enable navigation between places that can be recognized using behavioral methods such as wall following or image matching [21]. Compared to metric mapping the topological approaches have advantages such as easy path planning, efficiency and easy human machine interface as well as easier in maintaining global consistency. However, they fail in capturing important geometrical information which limits their resolution [20].

In topological mapping, place definition in the form of features is stored in topological nodes and their relative positions in links. A very important act in topological mapping is correct node identification from topological features to ensure correctness of localization. If a new node is found, the link data is updated and a new node is added [82]. This can lead to the problem of

locally consistent but globally inconsistent map as the link data can be only locally correct with the error building up. Figure 7 shows an example of a locally consistent map that is not globally consistent [17].

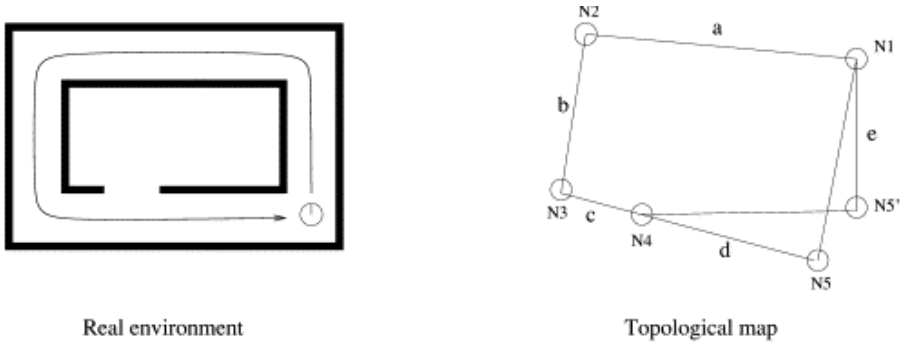


Figure 7: Local consistency does not guarantee global consistency [17].

The problem with relying on place definition only is that it is only valid for dealing with never seen before nodes and an already known node may generate sensor information that corresponds to an undiscovered node. Therefore, odometry information has to be taken into account to accommodate for this or odometry may be used on its own [17]. Topological maps can also be constructed using Voronoi diagrams [22]. In the Voronoi diagram each basic element used to divide the space is called a site and is built around points. In many cases, the points in space under consideration are used to divide the plane into convex polygons with the property that any point within the region is closer to the point forming that region than any other points [22]. This is explained in figure 8 where all points within the polygon around p_1 are closer to p_1 than any other point.

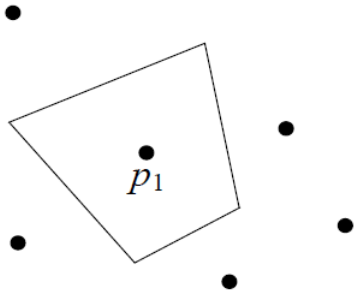


Figure 8: Points within the polygon are closer to p_1 than any other point [22].

Voronoi diagrams are affected by sensor noises and can lead to creation of unnecessary nodes that is why thinning algorithm is used to overcome these limitations [20]. Image thinning is an

algorithm used to reduce an image to a skeleton of lines that describe the topology of the image and can allow generation of topological maps from grid maps. An example of thinning is shown in figure 9.

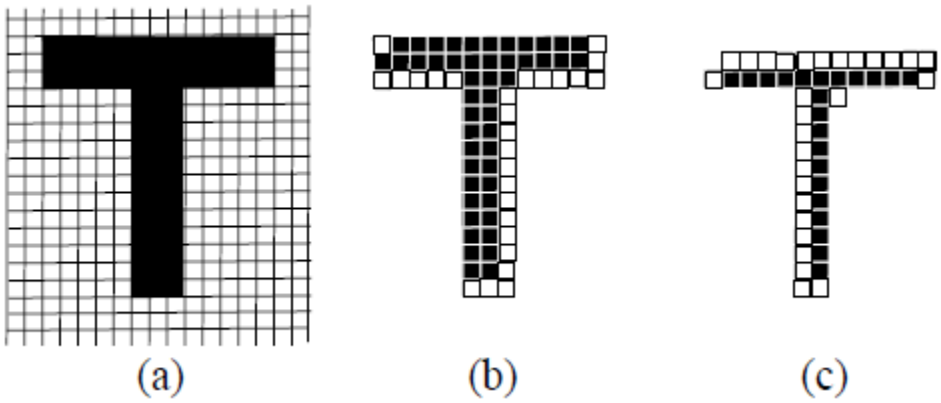


Figure 9: The thinning of a T shape in two steps [22].

1.5.3. Hybrid map

One of the problems with metric maps is that the process of path planning as well as maintaining the map becomes increasingly difficult memory wise as the map grows in size. On the other hand, topological maps with their abstract nature are efficient and allow easy path planning but are not suitable for local navigation and obstacle avoidance. This has led researchers to propose hybrid solutions for better results and is gaining popularity [21, 82].

In the context of hybrid mapping, metric maps can be used to locally track and build a representation of the environment and since the operation is local, there is no need to solve the loop closure problem while the arrangement of local metric maps is represented topologically [23]. Alternatively grid maps could be used for mapping to provide an input to topological maps such that topological features are extracted from grid maps. This makes an easy navigation and path planning by the robot [23].

1.6 Dynamic objects

1.6.1Dynamic objects in SLAM

The importance of dealing with moving objects will enable SLAM operation in dynamic environments which will open up the way for many applications of autonomous robots [24]. Most researchers in their approach to SLAM assume a static rigid environment that is unchanging. This essentially means that dynamic objects (other than the robot) do not exist or

their effect is minimized. However, the accuracy of localization and mapping is badly affected when dynamic objects exist in the environment and are not taken care of [24].

There are two tasks associated with dynamic objects these are detection, and tracking of moving objects (DATMO). According to reference [24], a DATMO algorithm has to satisfy the following:

a. Detection of objects
b. Initiation of new moving objects list
c. Data association for the moving old moving objects with the newly acquired ones
d. Merging different entities of the same object
e. Removal of objects that disappeared from the new view

The problem of DATMO is difficult mainly due to the variety of velocities with which moving objects move, the possibility that objects appear and disappear, the shape of an object may change form one scan to the other. Further, in outdoor environments the problem is complicated as there are pedestrians, bikes, and cars [24].

1.6.2. DATMO formulation

To simplify the SLAM with DATMO problem, assumptions are made that the sensor measurement can be decomposed into two components one representing the static objects, Z^s and another for the moving objects, Z^m. This implies that measurements from static objects are independent of dynamic objects [24].

SLAM process in DATMO is like the ordinary SLAM in its inputs and outputs while the moving object tracking (MOT) accepts exterioceptive measurements, Z, and outputs a list of dynamic objects, O, and optionally their states, S e.g. accelerating or constant speed, at certain location. SLAM with dynamic object tracking combines the two processes as shown in figure 10 [24].

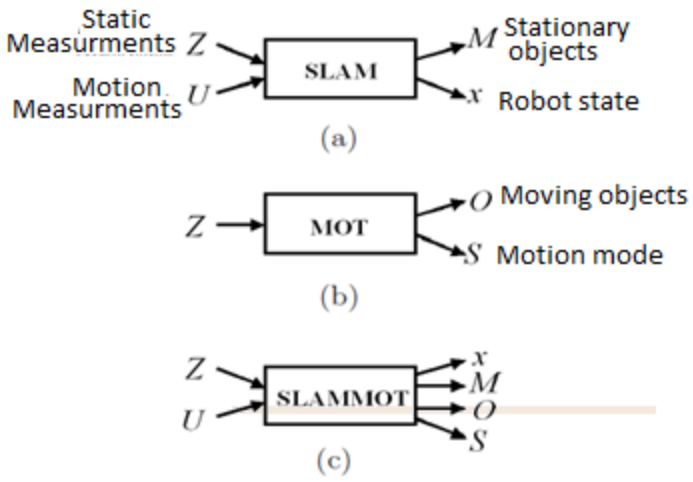


Figure 10: The inputs and outputs of (a) SLAM (b) MOT (c) the complete system [24].

Graphically MOT can be represented as in figure 11 where arrows indicate dependencies.

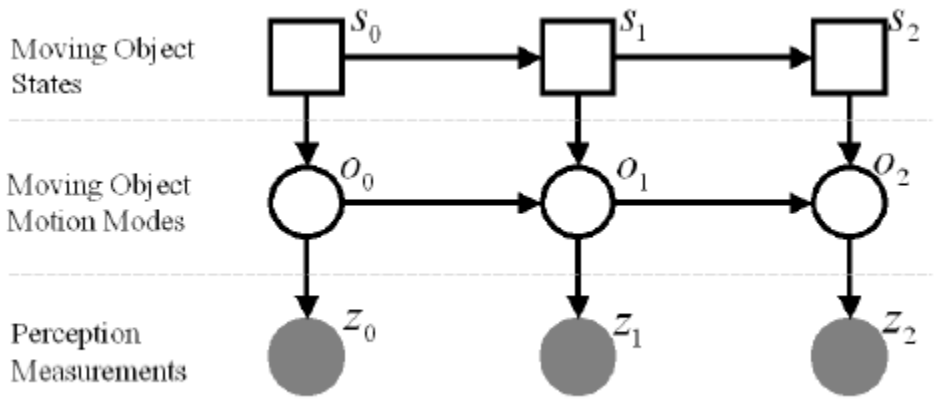


Figure 11: Dynamic Bayes network for moving object tracking [24].

The process of MOT can be represented probabilistically (notation) as:

$$\boldsymbol{P(o_t, s_t | z_t)} \quad \textbf{(9)}$$

Where o indicates the moving object states, s is the motion state, and k is the time step. Using Bayes rule:

$$\boldsymbol{P(o_k, s_k | Z_k) = P(o_k | s_k, Z_k).P(s_k | Z_k)} \quad \textbf{(10)}$$

The first term is the state inference while the second term is the learned mode of motion. To simplify matters, objects can be described as having predetermined motion states such as constant velocity or constant acceleration making it a discrete variable (as shown by the squares

in figure 11) but this limits the robustness of the approach as it cannot deal with varying motion states [24]. Despite this, tracking of objects is still difficult as the dynamic object could change its motion mode many times making it expensive to maintain a list of motion modes and variables.

Two approaches are traditionally used for detecting moving objects. These are feature based and appearance based approaches which are used with laser range sensors and cameras respectively. Both of these methods require the prior knowledge of the object. Another approach developed by [24] is motion based approach which detects moving objects regardless of their shape. However, this method may fail to detect slow moving pedestrians.

1.6.3 Human detection

For indoor environments, the moving objects are likely to be humans and this simplifies somehow the problem of dynamic objects as there is only 1 class to deal with. Humans have been detected mainly using two sensors, cameras and laser sensors. There are two approaches to human detection. The first is using probabilistic approach using a particle filter or Kalman filter (or its derivatives for non-linear estimation) to track the object, while the second approach is based on geometric feature detection [20, 24].

When using cameras, there is good probability of detection of humans but the camera is sensitive to the viewed scene and to the brightness. Laser scanners present only a slice of the environment and are not robust to noise but it is accurate to a distance compared to cameras and is not sensitive to lighting conditions in indoor environments [20].

In the case of using laser sensors, legs are modeled by geometrical shapes between circles and ellipses as in figure 12. Legs are differentiated from other geometrical shapes such as lines through the measurement of inscribed angle. Further, for tracking of humans a system is needed that is able to predict human motion in a clutter of other humans which makes multiple hypotheses approaches unsuited for this case. Moreover, the geometric approach may not work properly in environments with shapes close to legs such as fire extinguishers [20].

Another approach of leg detection is through modeling a large number of leg shapes as done by reference [20]. They used the experimental setup in figure 12(a) where one leg was allowed to move in front of a board at various distance and as in figure 12(b), where the human was told to

move in straight line in front of the robot to have good view of the leg. The results from 3258 cases were obtained.

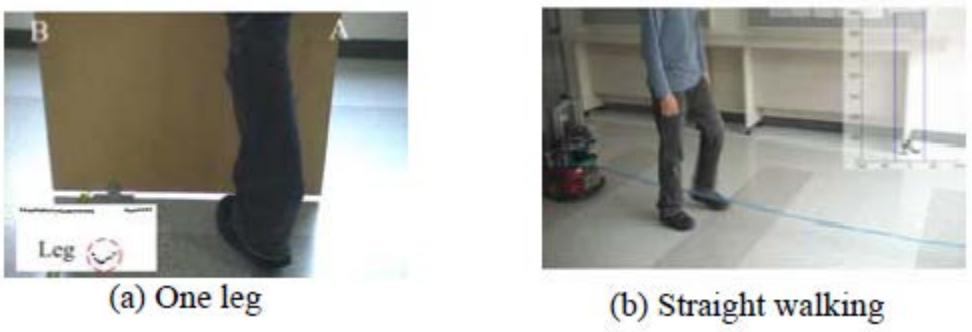


Figure 12: The experiments conducted by [20] to find the geometric features of the leg.

The results showed that the point to point distances lied between 1 cm and 25 cm with 80% of cases less than 5 cm, the length of the legs (girth) lied between 10cm and 25 cm for 80% of the cases. Importantly, leg widths from 9 cm to 18 cm had 98% of the cases. Finally, the maximum depth of the convex legs was 3 cm. this can be used to identify and extract legs in laser scans [20]. This approach is not exhaustive but will cover a wide range of situations.

1.7 Thesis structure

This thesis is organized as follows:

Chapter one serves as an introduction for the topic

Chapter two will present a literature review of some of the methods and results in the field

Chapter three will present the layout of the system developed

Chapter four will introduce and discuss development of each of the algorithms and will present the results obtained along with limitations for each algorithm

Chapter five will present the conclusions drawn from this work and future investigations for the different parts of the approach.

Chapter 2

Literature Review

Robot localisation techniques are broadly classified as either being local or global [26]. Global localisation techniques involve the use of external beacons such as GPS to localise the robot which is regarded as unsuitable for indoor environments [27]. Many of the papers reviewed here make use of the extended Kalman filter (EKF) and feature based SLAM. The computational workload for the EKF is a quadratic function of the number of landmarks integrated [28].

The authors in [29] presented a method for using laser sensors with topological mapping to correct for dead reckoning errors that arise from odometry. The authors' aim was to correct the errors in dead reckoning using data from exterioceptive sensors using a technique that combines the accuracy of grid based mapping and the computational efficiency of topological mapping. This is to ensure scalability and applicability to sparse environments. The technique used was to assign a coordinate frame to each topological node, as shown in figure 13, as well as a node region in which the robot is considered to be in this particular node. In this way, the odometric error becomes bounded as long as the robot is within the region of one node. The robot transfers its local reference frame from one coordinated to the other as it navigates.

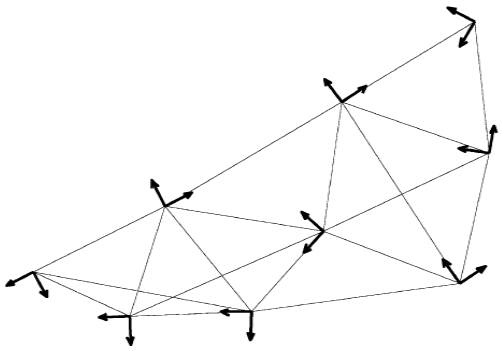


Figure 13: Example of topological map with coordinate frames [29].

To prove the validity of their algorithm, the authors in [29] conducted two outdoor experiments using a utility vehicle that was fitted with wheel encoders to collect odometry data for dead reckoning with SICK laser mounted on the vehicle's front bumper and used to collect data from the outdoor environment. The experimental setup is shown in figure 14.

Figure 14: The experimental setup of [29].

In both cases, feature extraction algorithms were used that classified the raw data as referring to point features or edges and if a set of raw data did not provide an extracted feature that matched any of the modelled features, it was rejected. The results obtained using this method were displayed by the authors who were able to achieve errors on the order of 3 meters [29].

The authors in [30] investigated the use of a localisation system based on the identification of artificial landmarks in an indoor environment with application to automatic guided vehicles (AGV). The experimental environment is shown in figure 15 where P_1, P_2, P_3 are artificial landmark designators made as a barcode that can be read by the laser sensor for the robot to identify its location with respect to the feature.

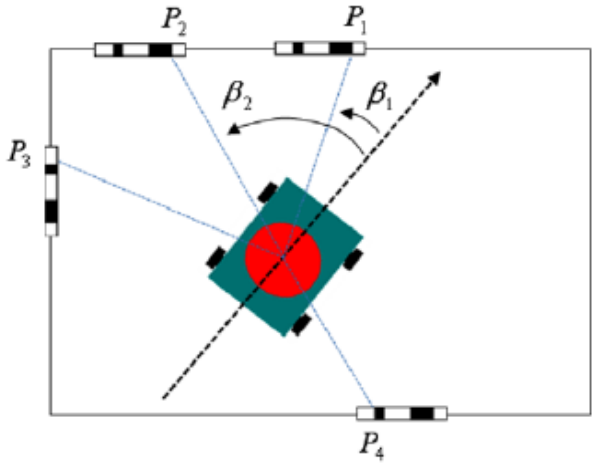


Figure 15: Elements of the setup used by [30].

Reference [30] use a Gaussian weighted fusion approach to localize with respect to the barcode features.

The authors simulated their algorithm to examine its ability to reject misidentified landmarks and their results proved this as wrong landmarks association were rejected. However, the environment had to be modified by attaching luminous metallic barcodes on certain landmarks to act as features for the laser sensor utilised it may not always be feasible or cost effective to modify the external environment for landmarks especially for large sparse environments [30].

The authors in [31] proposes a method when the map is known with certainty but the problem is the location estimate of the robot. Odometry is widely used to aid in dead reckoning, but it cannot be relied upon as it suffers from errors specially those which occur during wheel slippage so a method for fast localisation that does not depend on robot orientation is used [31].

The robot is fitted with 16 sonar sensors in a ring with the readings from the 16 sonar sensors collected and passed through a low pass filter to form a smoother range profile which is called the normalized depth function [31]. The procedure applied for identifying landmarks from sonar readings is to store the resulting plot in memory then apply an algorithm that rotates its contents. Since the sonars are arranged in a circular array, this circular rotation guarantees that a match will be reached regardless of the robot orientation. The remaining process is comparing the rotated readings with the readings stored in the memory for different landmarks. Once the robot identifies landmarks, it can localize through its environment. Experiments were conducted by [31] to prove their concept from which they concluded that the process was fast and computationally efficient but may sometimes fail to differentiate between landmarks.

Kong et al [26] adopted a mobile robot designed specifically to reduce uncertainty in odometry readings. The mobile robot used in experiments by [26] has O rings that contact the floor with a sharp edge while being attached via linear bearings. Reference [26] mentions that contact with the floor will reduce the uncertainty due to wheel base. Features such as lines and corners were used for SLAM. Figure 16 shows a simulation of a typical scan of an indoor environment with points that are circled indicating potential corners.

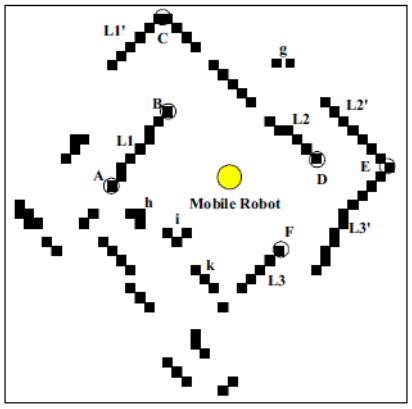


Figure 16: Corners identification in a typical laser scan [26].

The technique used by [26] for detecting lines is based on the continuity of a straight line assuming a vertical scan and every three points are labelled as a line if their orientation with respect to the robot forms two triangles with a common side that can be merged into one. The measurements from dead reckoning were combined with identified landmarks using EKF. Simulation carried out by [26] shows that the robot was able to successfully navigate through an environment of obstacles and localize based on their corner localization method. Result for the path followed by the robot in an experiment is shown in figure 17.

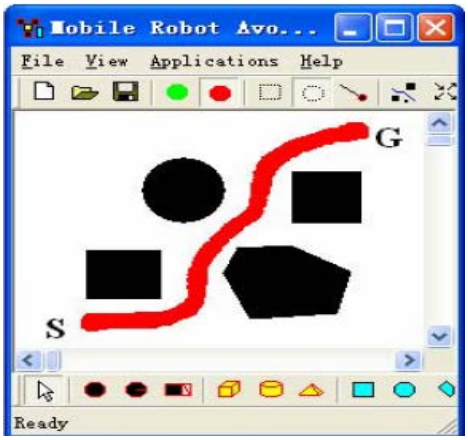


Figure 17: Simulation results obtained by [26].

The authors in [26] concluded that the combination of relative and global localization techniques leads to better results. However, for this method a map must be supplied to the robot.

Fraundorfer et al [32] presented a visual topological SLAM technique that relies on building a topological map using a sequence of visual waypoints online. For the generation of the images a single perspective camera was used. The robot navigates the environment using a series of images stored in the database to work out its relative orientation. Whenever an image is captured, it is compared to the database of images and is discarded if a match is not found. According to the author, the loop closure is inherent in this technique. In case of a robot losing track, the robot moves around randomly until it finds a matching image where it can localize.

Reference [32] calculated that it should take about 4 GB of memory for a robot with a 15 HZ frame rate camera to SLAM for 18.6 hours. The authors after comparing images would create a link between the images using a link graph also called an adjacency list [32].

Experiments were conducted using a pioneer DX robot with sonar and camera. The sonar was used only to prevent collision while the camera was a 640X480 pixels camera at a frame rate of 15Hz. Three sets of experiments were conducted to test various performance issues and in one set of interest the robot was first driven manually then asked to follow the path it was driven in. The results are shown in figure 18 where the black line indicates the driven path and the blue one shows the path followed by the robot. The robot needed 115 images and successfully identified the path.

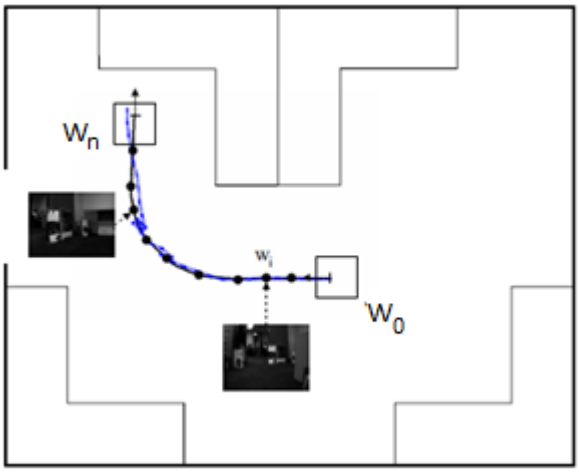


Figure 18: results of the second experiment [32].

Further experiments were conducted by [32] to test their method but for short distances hundreds of images need to processed which makes this SLAM method hard to scale up.

Reference [33] discusses the importance of being able to localize in the case of GPS failure for an outdoor agent. The authors discuss the benefits of topometric localization that uses a fine gridded topological approach compared to a purely topological or metric approach [33]. For creating the map, the authors move the vehicle on the desired route several times using Google street view project. Topological nodes are created at equal intervals and annotated with robot pose. A Bayes filter used to probabilistically localize the vehicle. For the experiments, the authors used a vehicle equipped with two cameras at 45 degrees to the front of the vehicle attached on the right and left together with laser sensors that provided range information. The experiment was carried on a route 8 Km long with varying landscape and features at different seasons.

For the Bayes estimation, the authors used empirical models to learn the similarity measure between observed data and the database to work out the measurement probabilities. For example they found that the measurement probability follows a Chi squared distribution while that for ranged data followed an exponential distribution. Localization experiments were carried out using the laser sensor only, the camera only, and the combination of both to see the effect of the different sensors on performance. Secondly, using the same data set the robot was kidnapped by moving the vehicle to different position and asking it to localize. The localization was done using range data only, vision data only, and both to evaluate the performance. Results are summarized in figure 19 which shows required time and distance for localization in a known environment using image only (a), laser only (b) and combination of both (c).

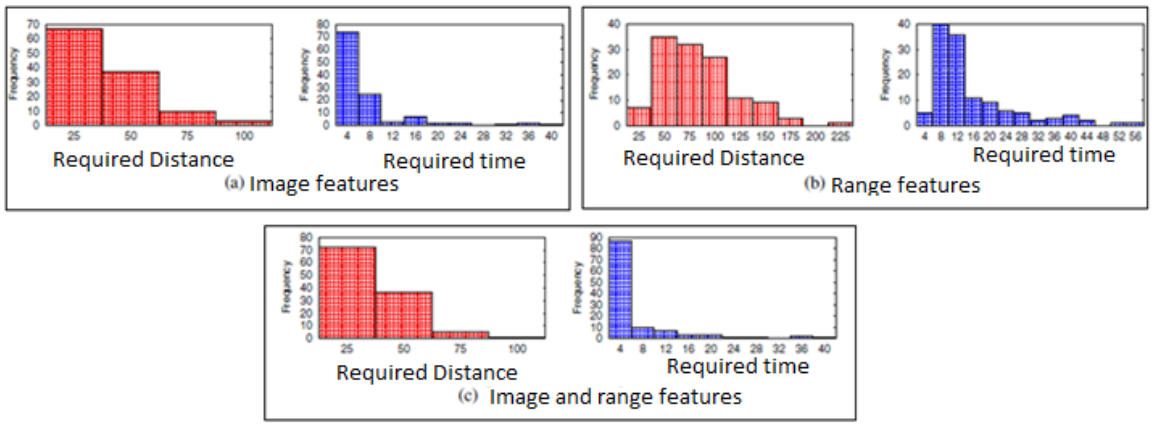


Figure 19: The tests using vision are superior to those using ranged data only [33].

The approach used by [33] required that the route has to be travelled beforehand i.e. the map must be available to the robot, and there is a need for the use of GPS.

Reference [34] describes the use of the EKF-SLAM but with an improvement that is obtained by using thinning based topological information. The method used by the authors is a method used in image processing called thinning method and is used to extract the skeleton of an object in an image by gradually thinning the outside and inside of it until a skeleton is formed. Figure 20 shows an example of a T-shape being thinned to produce its skeleton.

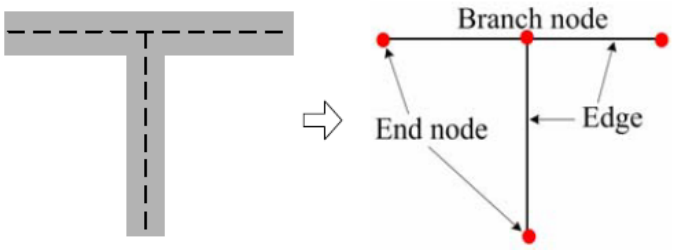


Figure 20: Example of T-shape being converted to skeleton [34].

In their method, [34] used laser range device to scan the environment using a grid based mapping and Bayes theorem was used to update the grid elements based on readings. Thinning was then used to provide a skeleton topological map as figure 21 shows the steps.

Figure 21: Thinning of topological map from occupancy grid [34].

Features are attached to each node locally, and with this method the error in data association can be decreased as the robot does not have to compare every feature in the map but only the local features. For example in figure 22 where there are many nodes with many features in the map, the robot will only investigate the features in the four nodes adjacent to it.

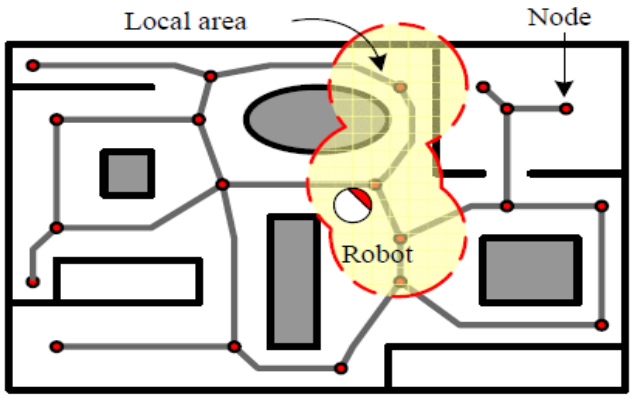


Figure 22: The robot investigates features in its vicinity [34].

For experiments, a pioneer P3-AT robot equipped with SICK laser was used and the robot moved at 0.2 m/s. The first experiment was conducted in an office environment 9.5 m X 7 m modeled with a grid 10 cm X 10 cm. Figure 23 shows the map generated by the robot (b) compared to actual map (a) [34].

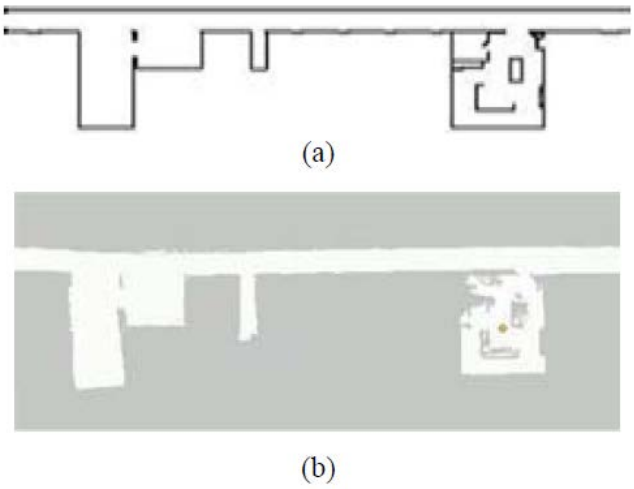


Figure 23: The map of the environment (a) and the map resulting from experiments (b) [34].

The authors compared the time needed for their algorithm to map in an increasing environment with the standard EKF SLAM and concluded that their algorithm runs constant in time for an increasing space while the traditional EKF SLAM grows quadratically as the number of features increases and the errors obtained were about +/- 20 cm for position and +/- 5 degrees for pose [34]. However, [34] did not mention how the grid map was obtained. This is important since grid maps by nature require a lot of computation unless a model is developed that minimizes computation time.

Reference [35] mentions the importance of dealing with dynamic objects and presents a system that splits SLAM from moving objects tracking (MOT). The architecture of the system is shown in figure 24. Starting at the robot pose, the algorithm analyses landmarks and groups them into dynamic and static landmarks. On the right side are the static ones which are divided into existing and new landmarks. Both of these are updated using the traditional EKF. On the left side is the update for dynamic landmarks [35].

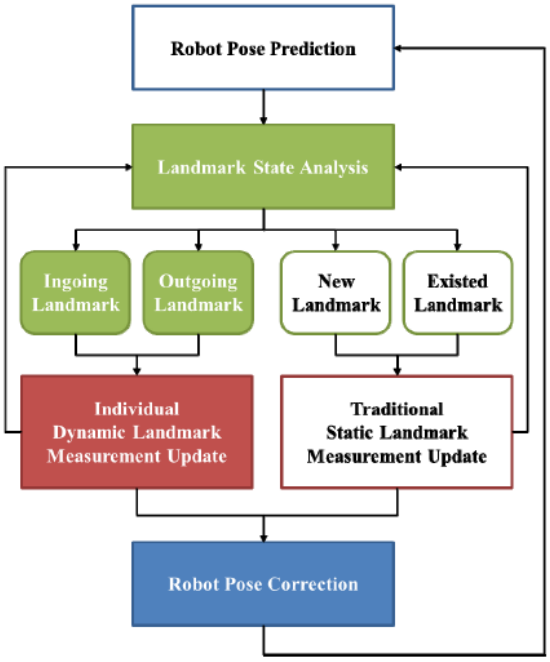


Figure 24: The architecture of dynamic EKF-SLAM [35].

Simulations were conducted in a MATLAB environment that featured an environment of 250m by 200m and included 35 landmarks with four dynamic landmarks [35]. Three simulations were made and the result for one run is shown in figure 25 that shows the robot path error variation. The results indicate that the error in robot path is bounded and not growing.

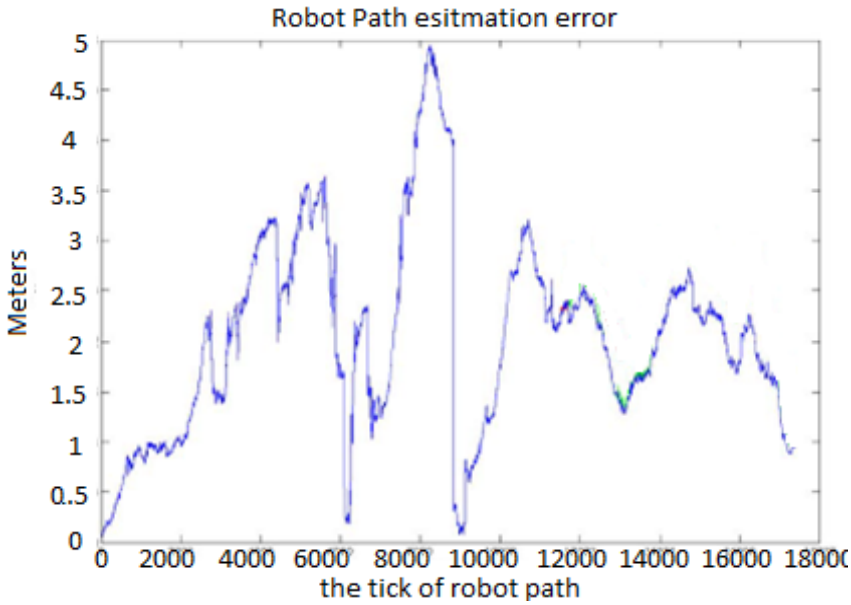


Figure 25: The robot path estimation error for three different simulations adapted from [35].

Figure 26 shows the variation in landmarks location error for one of the three runs.

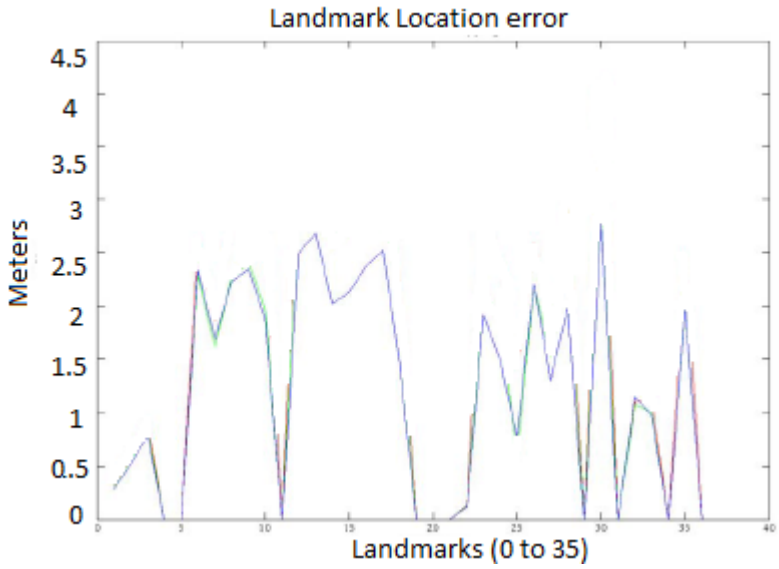


Figure 26: The error in landmarks locations for each landmark in the three different simulations adapted from [35].

In [35] Simulations were conducted without experiments to confirm their algorithm. The authors said that the proposed method had to work with an extra number of EKF without mentioning the effect on time.

The author in [36] emphasize the importance of incorporating human beings in the SLAM process as interactive human guidance can be used to guide the robot during SLAM.

Reference [36] presents an approach where the robot creates a topological map using help from a human guide. The human indicates the presence of a node by sending a signal to the robot using wireless laptop and the guide describes the connections between nodes as being door, corridor, or room. The robot corrects odometry drift when moving through nodes. For indoors environment with small set of features (structured environments) this method is effective [36].

The map used for localization consisted of point and line features generated during SLAM using a fusion of laser and sonar readings. Further, the system generates a segmented occupancy grid. The tour guides the robot using vocal commands by saying robot we are at room for example to declare a node as a room. When mapping is done, the robot switches to localization and can be commanded to go to nodes that it learned [36].

Scan matching was used by the authors in [36] with a laser sensor for correcting the odometric drift. An occupancy grid was used that can accommodate 700 laser scans in only 1.4 seconds of processing time i.e. 2 ms/scan. A method was needed to identify the human guide during the training phase. To do this, they segmented their laser scans according to depth discontinuities that exceed a preset threshold value. If these were within the expected size of legs, then they were labeled as legs [36]. This method is similar to the approach used in this thesis to detect human dynamic objects. Experiments were conducted by mapping rooms along a corridor that is 70 meters long. The robot did not enter these locations and mapped them from outside [36].

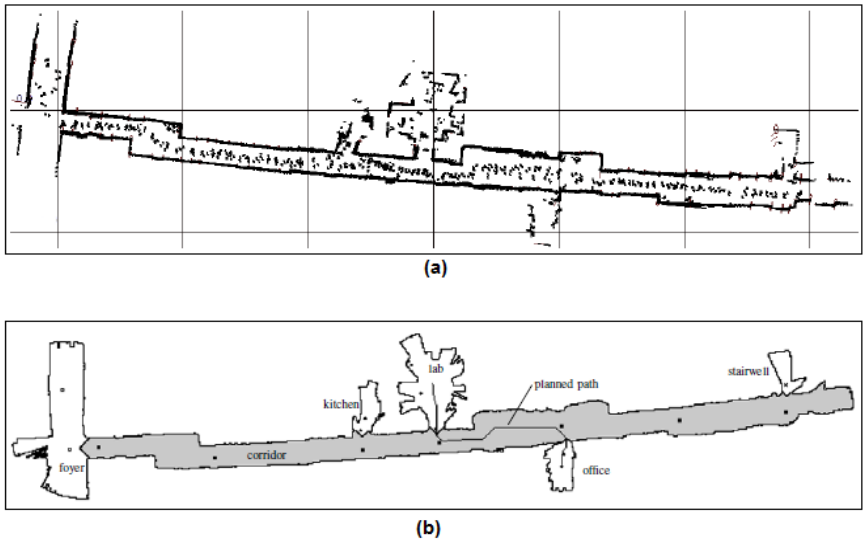


Figure 27: laser scan map before correction (a) and gird map after correction (b) [36].

The features map that resulted from this mapping was curved due to odometry errors (figure 27 part (a)) but when scan matching was used a more accurate map resulted in figure 27 (b).

The method proposed by [54] carries out the scan matching using polar coordinates directly from the laser scanner. The first step in their method is to preprocess the scans by first applying a median filter with a size of 5 to the laser scan to extract moving objects and chair legs. Following a polar scan match transformation is worked out by iteratively minimizing a cost function. The authors in [54] tested their scan matching method by implementing a simple Kalman filter SLAM in where laser poses are used as landmarks. The laser scans were stored at 1 meter intervals to be used as landmarks.

The authors presented the results of one simulation and two experiments. The experiments used a SICK LMS 200 laser. In the simulation, a room was simulated and the reference scan was chosen and then the scan was transformed by 100 cm in x and y while the orientation was shifted 15 degrees. The algorithm was run and the final error was 0.4 cm in x direction, 0.005 cm in y direction, and 0.15 degrees in orientation. However, in some trials there were errors as high as 2 meters and 5 degrees [54].

In a different experiment, the algorithm was tried as part of a SLAM system based on Kalman filter and the results of mapping an indoor office environment are shown in figure 28 with odometry only (part a) and after applying scan matching SLAM (part b).

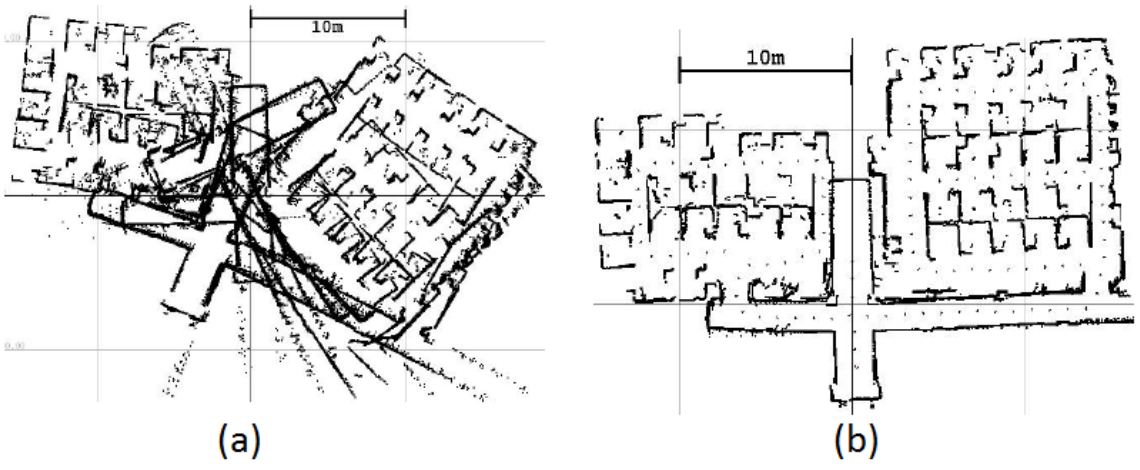


Figure 28: The results of applying scan matching SLAM to indoor environment [54].

Work by [37] is a method for robust sonar feature extraction that can be used to extract two sonar features namely; corners and plane features as shown in figure 29.

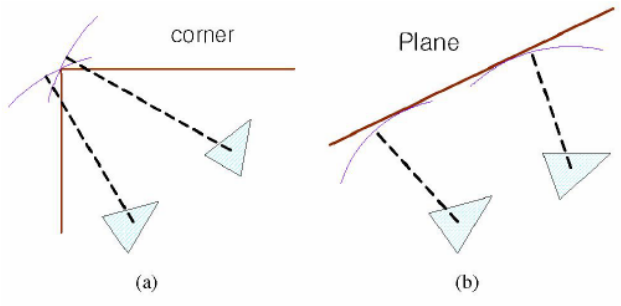


Figure 29: The method proposed by the authors is used to detect corners and planes [37].

The point feature detection that the authors present relies on the triangulation based fusion (TBF) of sonar data with added modifications on the basic TBF technique to make it more robust. The authors have mentioned three main problems with the basic TBF algorithm to do with locations of obtained features which should be more accurate, the urgency of removing false edges which do not occur in environments, and the generation as many features as possible for robot localization [37].

The first improvement made by the authors was the use of a stable intersection versus an unstable intersection. This is shown in figure 30 where (a) shows a stable intersection, and (b) shows an unstable interaction. Whether an intersection is stable depends on the angle of intersection. If it is greater than a preset threshold, then the intersection is stable.

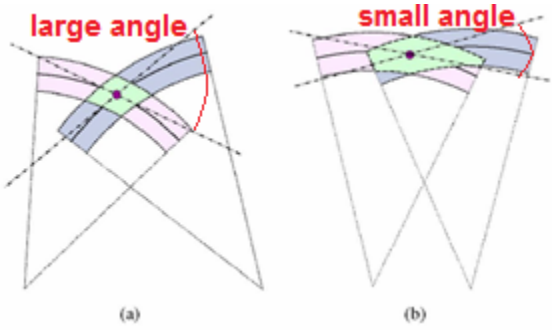


Figure 30: The difference between stable (a) and unstable (b) intersections [37].

The second modification was to the basic TBF window that updated data when the Euclidian distance exceeded a certain threshold which does not take into account the rotation of the robot which could enable the detection of edge features [37]. Further, in [37] it is claimed that

this will cause the heading error to remain uncorrected for. However, this claim is not entirely correct since observing robot pose at each update and relation between corners can recover the change of robot orientation. Experiments were conducted in a structured environment by [37] using poles to act as corner reflectors as shown in figure 31.

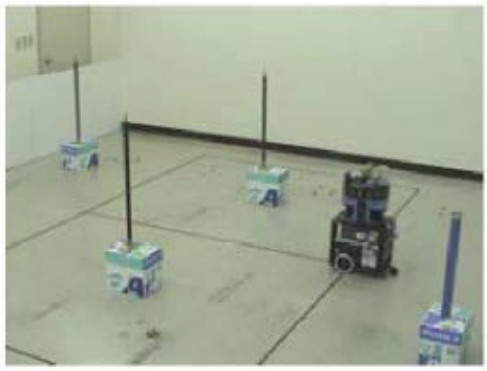


Figure 31: The experimental setup used by [37].

The authors used their modification for the original TBF algorithm to improve the results and reject false corners. They presented results that supported their work. Additionally, line features were detected based on similar readings for adjacent sonars and used by the EKF for SLAM. Overall results by [37] indicate a decrease in error compared to odometry using their method. Fang et al [9] propose a new method which fuses sonar data with data from CCD camera. Fusion was performed at the feature level which improves the precision of the system [9]. The description of the process used by the authors is shown in figure 32.

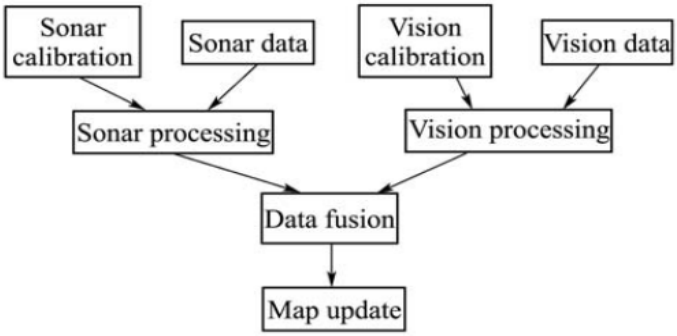


Figure 32: The summary of the whole process used by the authors [9].

The calibrated sonar data is processed then fused with the calibrated and processed vision data to yield the fused data with features to update the map. Since most manmade internal structures (in structured environments) appear as perfect reflectors for the sonar sensors, only

surfaces which lie perpendicular to or close to perpendicular to the sonar central beam axis will be correctly ranged [9]. The Extended Kalman filter (EKF) was used to relate the same features from the two sensors.

To test their system, experiments were conducted with a Pioneer 2DX robot with a sonar array of 16 sensors and a CCD camera. The first experiment was conducted in the corridor of a lab and the second experiment in a square room. The mapping results with sonar data only and sonar camera fusion data are compared in figure 33 parts (a) and (b) respectively.

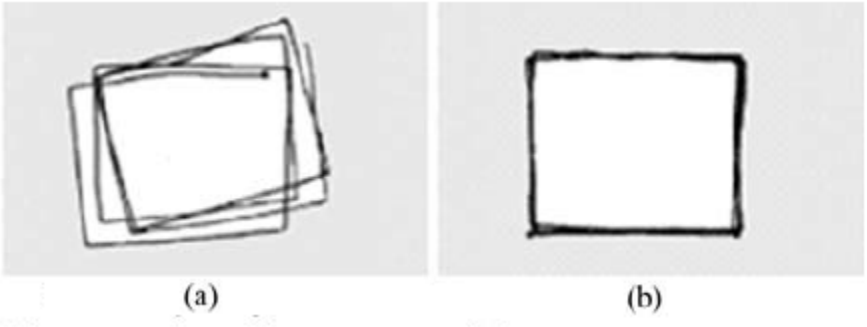


Figure 33: The mapping of square room (a) using sonar only, (b) using multi-sensor fusion [9].

The mapping in figure 33 (a) using sonar only shows typical unbound mapping error due to accumulating odometry errors while in figure 33 (b) the error is smaller with the data from the sonar and camera. The results indicated that there was a noticeable improvement that results from using multi-sensor fusion as compared to using sonar only.

A triangulation based method is proposed by [38] to interpret sonar readings similar to [9] discussed above. The method is called TBF and is a voting scheme that assumes all sonar sensors lie on the same plane and is used to detect point features in an environment. To see how this triangulation method works, consider figure 34.

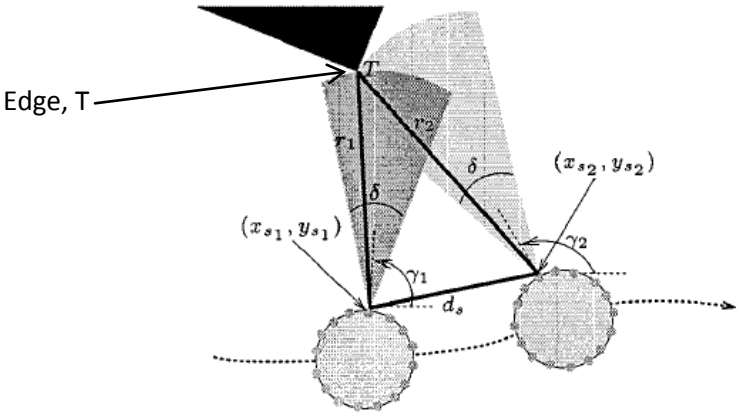


Figure 34: The triangulation geometry [38].

The edge, T, in figure 34 will be detected if the intersection point between the two sonar arcs at different times (corresponding to different robot locations) can be found. This is the case if the edge, T, satisfies the following pair of equations for the two robot positions.

$$(\boldsymbol{x_T - x_{si}})^2 + (\boldsymbol{y_T - y_{si}})^2 = \boldsymbol{r_i^2} \quad \boldsymbol{i = 1, 2} \qquad \textbf{(11)}$$

$$\boldsymbol{\arctan\left(\frac{y_T - y_{si}}{x_T - x_{si}}\right) \in \left[\gamma_i - \frac{\delta}{2}, \gamma_i + \frac{\delta}{2}\right]} \quad \boldsymbol{i = 1, 2} \qquad \textbf{(12)}$$

Where subscript T indicates the coordinate belonging to the edge T, and subscript si indicates sonar coordinates, all symbols are defined in figure 34. Equation 11 describes the sonar cone as a circle whose circumference is at the edge T. Two sonar cones from two circles and the edge can be found by solving the equations for the two circles simultaneously. Equation 12 is used to exclude the intersection which does not lie within the sonar cones. These equations have to be solved for two sonar positions (corresponding to i=1, and i=2). The solution to these equations is given by the following equations.

$$\boldsymbol{\hat{x}_T = x_{s1} + \frac{1}{d_s^2}\left(d_{x_s} d_r^2 \pm |d_{y_s}| \sqrt{r_2^2 d_s^2 - d_r^4}\right)} \qquad \textbf{(13)}$$

$$\boldsymbol{\hat{y}_T = y_{s1} + \frac{1}{d_s^2}\left(d_{y_s} d_r^2 \pm |d_{x_s}| \sqrt{r_2^2 d_s^2 - d_r^4}\right)} \qquad \textbf{(14)}$$

Where:

$$\boldsymbol{d_{x_s} = x_{s1} - x_{s2}} \qquad \textbf{(15)}$$

$$\boldsymbol{d_{ys} = y_{s1} - y_{s2}} \qquad \textbf{(16)}$$

$$\boldsymbol{d_s^2 = d_{x_s}^2 + d_{y_s}^2} \qquad \textbf{(17)}$$

$$d_r^2 = \frac{r_1^2 - r_2^2 - d_s^2}{2} \quad (18)$$

These equations generate the actual coordinates of the edge T, and false solutions. The false solutions can be verified against the first pair of equations [38]. In chapter 3 of this thesis it will be shown in the section about TBF (4.3) that the solution presented by [38] is not correct.

The TBF algorithm is implemented as a sliding window where the rows correspond to individual sonar sensors attached to the robot, while the columns refer to individual scans time wise. The sliding window is dynamic and not static as it is updated when the robot has moved a certain threshold distance varying from 0.05m to 0.25m [38]. As the window is updated, the earliest scans are deleted to keep the size of the window constant. The algorithm works by finding possible triangulation points with the most recent sonar data for every sonar sensor with the allowed distance between predicted triangulation and the compared sonar reading decreasing with successive triangulations.

The authors in [38] presented some results. However, it is not clear how did they obtain their results given the fact their solution to the triangulation problem is wrong.

Roh et al [18] propose a FastSLAM method that uses scan matching and particle weight based grid mapping. This mapping method is a subset of grid maps where the SLAM process is carried out using the particle filter. Their algorithm is summarized in figure 35.

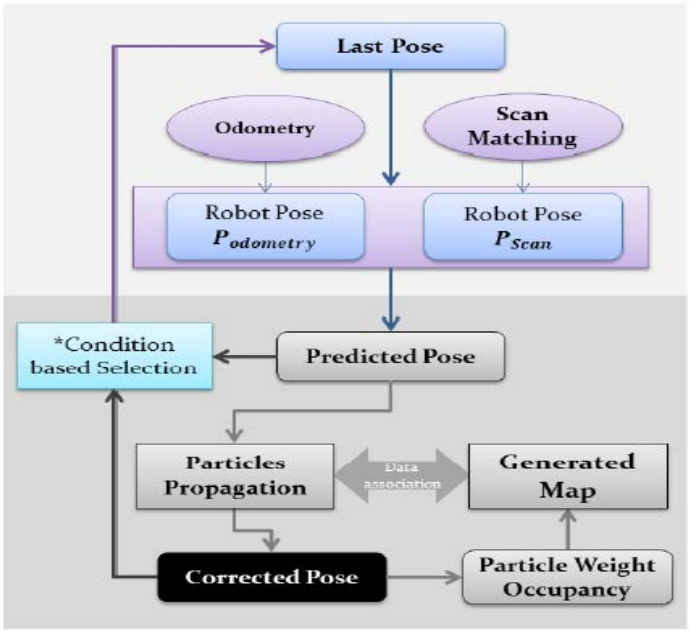


Figure 35: The summary of the algorithm used by [18].

The proposed algorithm is made up of a prediction step that uses odometry readings and scan laser matching to predict the robot pose. This is then used to predict the next pose and generate maps with proper data association and a weight is set to each particle map.

The authors present the results of experiments showing the map from a small room with odometry only (figure 36a) and with the proposed method (36b).

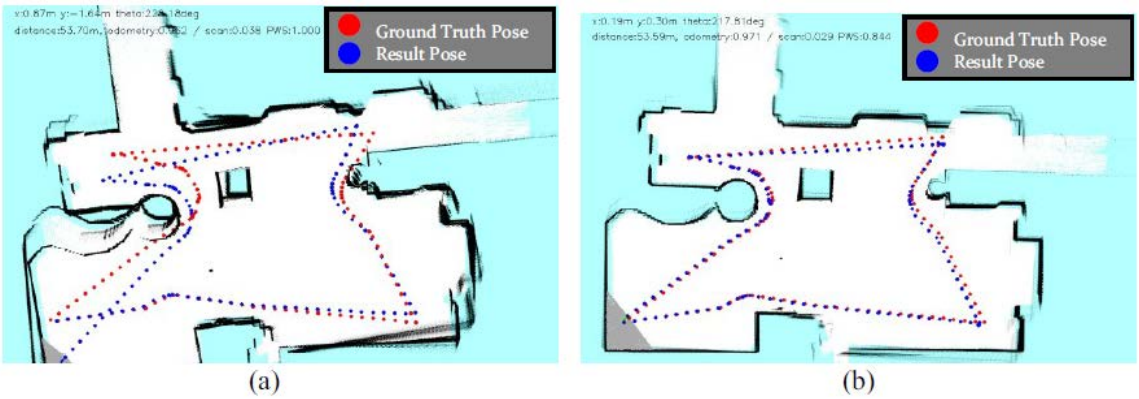


Figure 36: The comparison of mapping results between the raw odometry readings (a) and the proposed method (b) [18].

The authors say that the error is of the order of 7 meters for 340 meters travelled distance [18].

The authors in [39] present a method for topological localization using sonar data where they propose a template grid that is extracted from the grid map around the robot and used to localize as the extracted template is compared to original local grids (in topological map). A similarity measure is worked depending on how similar the current grid map portion to various templates stored in database. The resulting probability is used in the Bayes conditional equation to update robot state instead of the measurement probability. Reference [39] propose a method for the detection of the kidnapping, the authors rely on the entropy of template to detect kidnapping. The entropy decreases as the probability converges towards one node and increases otherwise. The entropy decreases as the robot localizes and increases if it is kidnapped (by external agent), and poor localization can also increase the entropy (positive derivative of entropy) [39].

The results by [39] indicate success in localization for kidnapped robot. However, their method requires a prior map as well as topologically extracted metric nodes beforehand making it suitable more for path planning and localization.

Kim et al [20] present a method to account for dynamic objects (human legs) where the legs are paired based on their dynamic characteristics to detect humans (individual humans). The authors used previous research models for predicting the location of human feet during gait cycle in order to group legs together. Three experiments were conducted with the first one of them being a human being in front of the robot in a complex environment where the robot has to track him. The environment map and scene are shown in figure 37. The dark track in 37(a) shows the person leg position, and the circles with a cross represent robot pose at different times.

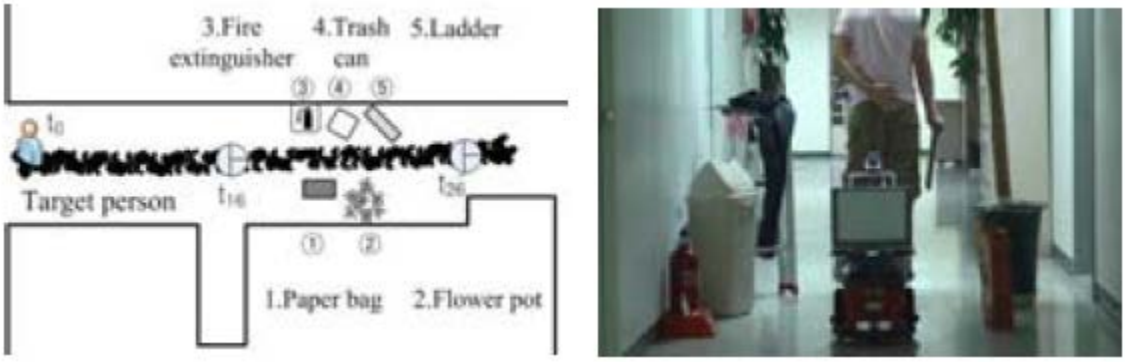


Figure 37: The experiment environment. The map (left) and the corresponding scene (right) [20].

The scan of the laser sensor from one moment during the experiment is shown in figure 38that shows the algorithm successfully extracting two clusters (3,4) and clusters (1,2) that for potential tracking as a human target.

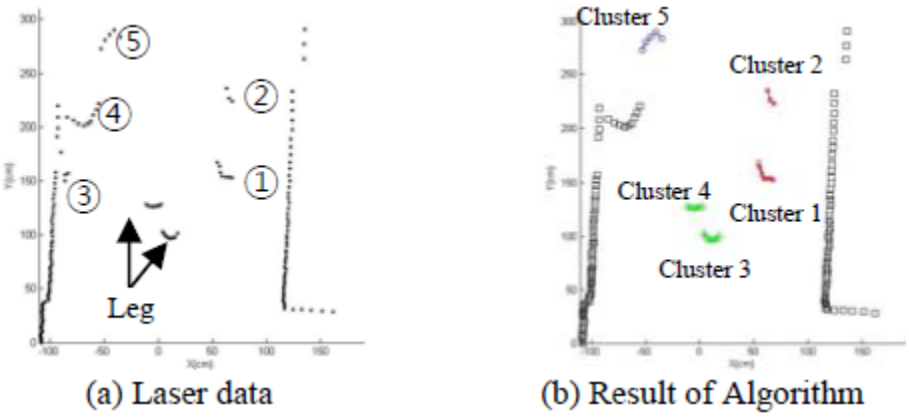


Figure 38: The laser scanand the algorithm result [20].

Another experiment was conducted to differentiate a target person from other people. The experiment relied on the motion model developed to filter unlikely target and was successful as

the authors said. The method developed by Kim et al [20] is similar to the method employed by this thesis except that no model of walking gait is used and no pairing of legs is made. The reasons are that tracking of dynamic objects is not considered but instead extracting dynamic objects to exclude them from the map is what is considered. Therefore, extracted objects need not be tracked and differentiated from each other unless required.

Wang et al [24] discuss the DATMO in urban environments where feature extraction is prone to error and there is no way to estimate the uncertainty in extracted features. Therefore, they propose the use of a scan matching algorithm (iterative closest point) and represent the environment in a grid map fashion. Further, a hypothesis tree is used for data association. Once a new object is detected a track is initialized for that object and its motion model is predicted to be one of three modes which are constant velocity mode, constant acceleration mode, and turning mode.

Experiments were conducted in an outdoors environment and the authors noted the presence of false positives. The false measurements arise in the case of [24], who have used a moving vehicle, due to roll and pitch motions of the vehicle. Further, if the ground is not level the scans may hit the ground yielding wrong moving objects. Despite this the results obtained by the authors showed an improvement over ordinary SLAM.

Martinez et al [40]propose a hybrid approach of (iterative closest point) ICP and genetic algorithm GA to solve scan matching where the GA is used to find a coarse but fast estimate of the transformation matrix and the ICP is used to refine the estimate with a local search. Experiments were conducted using an outdoors vehicle equipped with SICK laser sensors (LMS200) in an alley where there unstructured objects and moving objects such as cars and pedestrians. The vehicle was manually driven for the purpose of collecting data for the experiment [40]. The authors compared the performance of their algorithm with purely GA and purely ICP methods.

The purely GA methods used a population of 120 samples and required 60 iterations for convergence while the hybrid GA required a population of 80 samples and only 40 iterations due to the coarse nature of the estimate. Further, pure ICP requires 15 steps to converge while the hybrid ICP requires 6 iterations to converge to a fine estimate starting from the coarse estimate provided by the GA [40].

The time required for the algorithm to run is between the time required for pure ICP and the pure GA. Therefore, the main advantage of this method is to overcome the local minimum without sacrificing accuracy. However, the average running time is still comparable to GA and in best cases was (60ms) but rises to (200ms) [40]. Given the fact that the laser scanner runs at 10-15Hz, then this implementation may not be real time unless parallel computation is used.

Wang and Thorpe [25] For SLAM used scan matching approach. According to the authors point to point (P2P) scan matching approaches are slow and computationally intensive but they are more robust to errors and unexpected shapes in the environment compared to feature to feature (F2F) matching methods and so the authors chose the former approach.

Wang and Thorpe [25] have carried out a simulation where a typical outdoors environment was used with a grid map of a 5 cm resolution. Further, the authors propose the possibility of combining their point to point scan matching approach with the faster feature to feature matching approach to yield better results. The authors did not explain their criteria for feature extraction and how its accuracy could affect the results. The results presented by [25] indicated the ability of the vehicle to correctly identify pose changes from scan matching using a laser sensor.

Milford et al [14] conducted experiments to test ratSLAM approach. A pioneer2-DXE robot was used with a specialized vision processor for the 40 degree FOV CCD camera and a 1.1 GHZ Pentium III laptop. According to the authors, the cycle time needed for all processes was 200 ms. Experiments were carried out in a controlled indoors environment of size 20 m by 10 m. The environment was modified by adding colored cylinders to act as landmarks for the robot. The cylinders used in the first experiment had unique colors and locations thus no ambiguity was expected to arise from the landmarks. The results of this run are shown in figure 39. In part a, the map shows the expected cylinder locations based on odometry only while part b shows the mapping using ratSLAM.

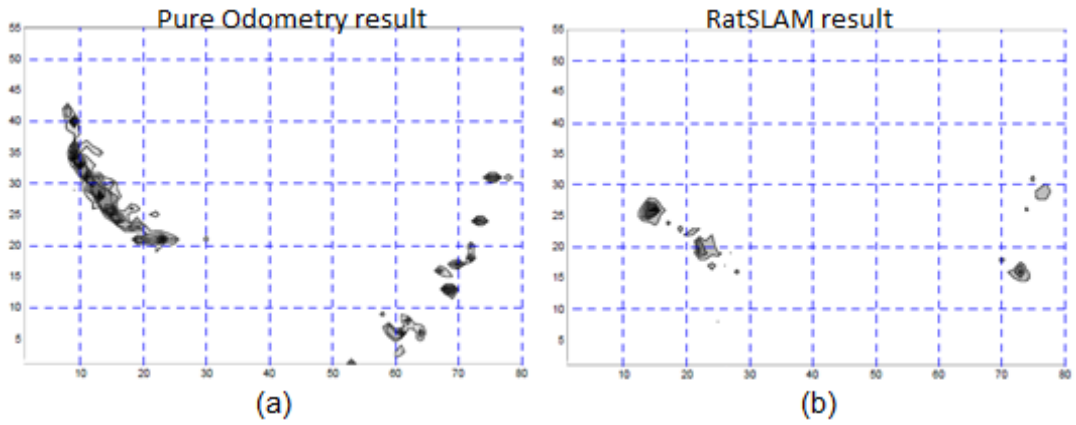


Figure 39: The results from ratSLAM in a unique indoors environment (b) compared to pure odometry (a) [14].

The shades show the location of the cylinders and uncertainty in it. The next set of experiments featured some ambiguity as two red cylinders were used in two different locations. The results indicated the localization error kept increasing until unique landmarks are discovered [14].

RatSLAM does not produce strictly Cartesian representations of the environment. Further, many factors are empirically determined values which are likely to be affected by the environment in which the robot operates. Moreover, the computational load due to relying heavily on image processing causes a low cycle rate (200 ms). Additionally, the experiment conducted by [14] features a controlled environment and does not account for changes in visibility or features due to changes in lighting and shadows. All these factors allow for more future work.

In reference [23] an approach is used where a metric map is used to store a local perceptual map of the neighborhood of the robot. Further, on the control level gateways are identified where the robot action shifts from motion between neighborhoods to localization. This enables the robot to use abstraction of the topological map for large scale navigation while not losing the details at a local state as it relies on a metric map. A gateway is defined by reference [23] as a boundary between different environment regions. The regions are identified by many factors such as major changes of visibility. In buildings these correspond to doorways or walls that create partial separation or limit visibility between two regions. The method used by reference [23] was applied in a large office environment with multiple loops and the authors [23] described the results as successful in bounding localization errors.

Vision SLAM was tested by researchers such as reference [16] who proposes an approach for visual SLAM based on EKF and a feature based map. According to the authors, their approach

has the ability to consistently map and maintain the features in an environment since it propagates and updates the uncertainty in feature position. The number of features used is limited to 100 features for real time implementation since the complexity of EKF update increases as $O(N^2)$ and thus this method is suited only for small environments. However, it is repeatable and consistent as the authors have explained [16]. The authors mention that their system needs special setup to start in that four features (four pixels in a square) of known initial position and depth are supplied to the system to start since it is not possible to recover depth from a single camera using a single frame. The authors used their algorithm with a humanoid robot in a cluttered indoor environment subjected to rapid acceleration of the mobile platform and the authors equipped the robot with a mono camera. The experimental setup is shown in figure 40.

Figure 40: The experimental setup for vision SLAM [16].

The robot followed a driven circular trajectory of radius 0.75 m and the localization error was monitored. The final error was on the order of 2 cm. The results by [16] showed that the error reached its minimum when the loop was closed by being at the starting point. However, like all EKF based SLAM the approach by [16] suffered scalability issues as features grew in size.

Wen et al [8] use a NAO humanoid robot equipped with a camera in its head and a laser sensor with a 240 degree scan range on top of the robot's head. The authors employ EKF based SLAM using both the camera and the laser. Thus, this method cannot be regarded as a vision only SLAM. The authors modified the environment by including special landmarks predefined to be detected by the robot for SLAM. Therefore, this approach modifies the environment similar to the vision only ratSLAM developed discussed by [14].

The experimental environment (indoors) has only two landmarks and the authors used the camera mainly to detect the landmarks for EKF based laser SLAM.

To verify their approach, the authors used the algorithm on a simulator and in a real indoors environment with two obstacles that were of different colors. The objective was for the robot to move from its starting position to destination while mapping itself and avoiding obstacles. The experiment environment is shown in figure 41. Their results indicated that two features were sufficient to limit the localization error.

Figure 41: The experimental environment [8].

The authors in reference [41] propose a method to separate dynamic objects from stationary objects in a moving camera scene. They use one camera and make the following assumptions due to unavailable prior information about image depth:

- The assumption that the majority of optical flow is due to camera motion (ego motion) as the majority of image pixels are due to the stationary background
- The optical flow vectors converge to a single point, the focus of expansion
- Independent dynamic objects will create local foci of expansion corresponding to motion of groups of pixels

The dense optical flow algorithm of Nordberg and Farneback was used by [41] in their work. The calculation of the FOE is used to classify whether objects belong to the background or a dynamic object [41]. Experiments were conducted by [41] to test their methods. OpenCV was used to extract optical flow vectors from video sequence.

Figure 42: Two consecutive frames from the video used by [41].

Applying the triangulation estimate of FOE based on three known background images from the same video as figure 42 yielded an error rate of 14% for the background, and 90% for objects. The authors explain that the reason for such high error rate in dynamic objects is due to the homogenous sky which produces zero optical flow. By labeling the sky pixels as background, the error in dynamic objects using this approach was reduced to 74%. The error rates obtained by [41] were very large and not useful for accurate identification of dynamic objects. Therefore, more refinement is needed before it can be used.

The extraction of dynamic objects from camera images is approached in a different way in [42] where a camera is attached to a robot with the camera pointing downwards at the ground plane. Therefore, given the robot motion (hence camera motion) it is possible to estimate the expected optical flow based on a ground plane model. Pixels motions in the real image are then matched to predicted motions to extract dynamic objects [42].

Experiments were conducted in an outdoor environment with pedestrians crossing in front of the robot. Two consecutive frames are shown in figure 43. The robot works out the similarity of intensities of groups of pixels in the two frames and uses that to track pixels and calculate optical flow.

Figure 43: Part of the experiments conducted [42].

However, the method by [42] is not robust to shadows and may detect a stationary dynamic object as a moving object. An example is shown in figure 44.

Figure 44: Shadows are identified as dynamic objects by mistake while car wheel is identified as dynamic when it is actually stationary [42].

The horizontal white line describes the horizon while the contours around objects describe detected dynamic objects. As can be seen in figure 44, shadows on the left and the right were mistakenly detected as dynamic objects. Additionally, stationary car wheel was detected too due to intensity fluctuation. The computation time of 30 millisecond of the algorithm by [42] is good for real time processing and the ability to perform with skipped frames makes it robust to lost frames.

In [43] a method is proposed to estimate the ego motion of a camera attached to a driving vehicle in an outdoor environment using optical flow. The authors assume that the road is a planar surface that satisfies the rigid world assumption and other objects like cars and non-planar objects are rejected as outliers. With this assumption the ego motion estimate becomes a parameter estimation model [43]. To increase the robustness of the algorithm, the parameters are simplified to only three parameters namely pitch, yaw and forward translation [43].

Therefore, given the optical flow (u,v) the problem becomes how to estimate the motion parameters likely to produce that flow pattern. An initial guess is provided first about the vehicle velocities which can also be obtained from other instruments like the speedometer before refining it using gradient descent to reach the value of actual parameters [43].

Experiments were carried out in real road environment outdoors with a vehicle fitted with a 50 degree field of view camera attached to the rear mirror close to the passenger side. The images were digitized and processed offline at a resolution of 320 by 240 pixels. Figure 45 shows the result of comparing two consecutive frames at the pixel by pixel correspondence (a) and after applying the algorithm to take account of ego motion (b). White pixels indicate higher motion speed.

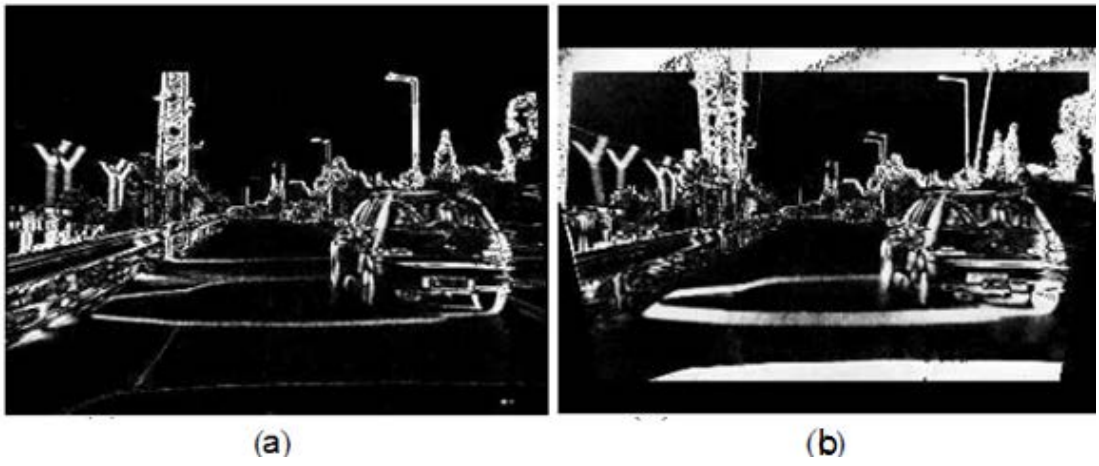


Figure 45: The results of finding the flow between consecutive frames pixel correspondance (a) and after applying algorithm (b)[43] .

To examine the accuracy of their algorithm, [43] used a road sign shown in figure 46 (a) as a starting point and worked out the angular displacement of the car as it drove in circle until it reached the same sign at frame 775 in figure 46 (c). Theoretically it should be 360 degrees and

the algorithm ego motion estimates reported 366.5 degree yielding an error of 1.9% or 0.017 degrees per frame [43].

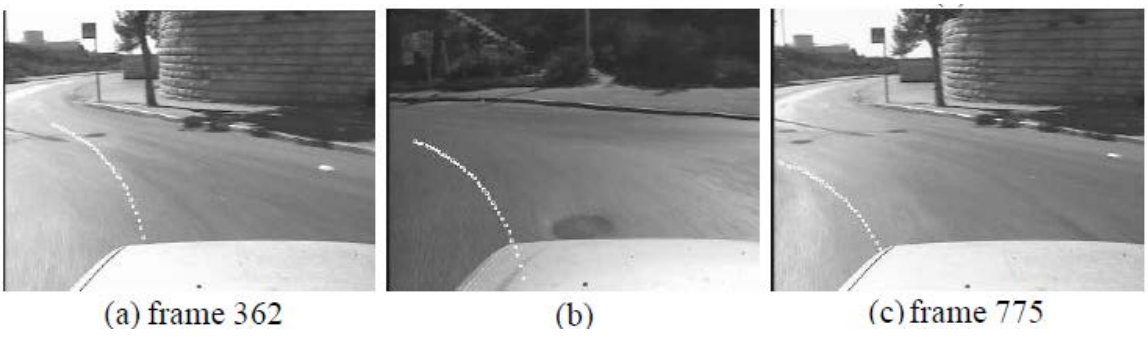


Figure 46: Set of frames taken during drive of the car in circular path to evauate accuray of algorithm egomotion estimate [43].

The authors did not have any estimate for odometry displacement but reckon that its accuracy should be similar to the accuracy of angular displacement estimate. Their method is not a SLAM technique but can be implemented in a SLAM system.

Royden and Holloway [44] develop a neural network based approach to detect the focus of expansion of an optical flow pattern from a moving camera. Their method is based on the fact that the optical flow has rotational component which is depth independent and a translational component which is depth dependent. Their approach is a biomimetic one that relies on the use of speed and direction tuned units to detect the focus of expansion and hence the observer heading.

The idea used relies on subtracting the optical flow of pairs of points in the image at different depths which will cancel the rotational component and will produce the radial pattern of expansion from which to find the focus of expansion [44].

Simulations were carried out with observer motion towards two planes at 400 cm distance and 1000 cm distance from the observer. An object that subtended a field of 6 by 6 degrees was located at the bottom right location of the image at 400 cm from the observer and was moved laterally at a speed of 52.6 cm/s from the observer translating at 200 cm/s [44].

The difference in predicted angle of optical flow vector and actual angle was used to rule out dynamic objects [44]. They used a threshold angle of 15, 20, 25, and 30 degrees.

A result of simulation is shown in figure 47. The dynamic objects detected by the system are labeled by their borders and shown in circles whose size corresponds to the magnitude of difference from the expected flow. However, note that some false corners exist. The authors said that the accuracy for detecting dynamic object borders is highest at a threshold angle of 20 degrees and decreases on both ranges to the side of it. While the false borders increased as the angle threshold was decreased.

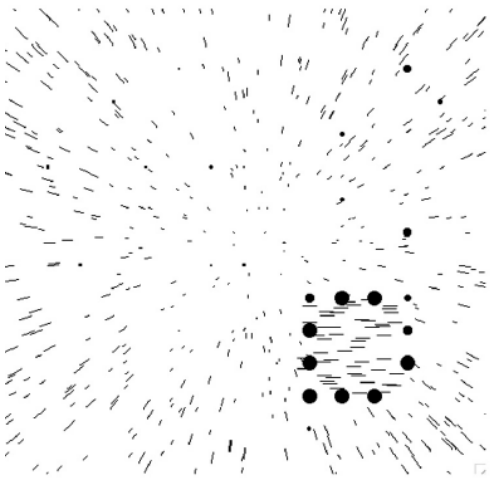


Figure 47: The result of detecting dynamic object based on their optical flow angle compared to predicted angle of optical flow [44].

Royden and Holloway [44] tried combining both speed threshold and angle threshold and the results were much improved with 90% of dynamic object borders detected on average and a false object detection rate of 20% and lower if averaged (since averaging reduces noise). Results showed that the detection is sensitive to object speed getting low at lower speeds but is not affected by heading or object orientation as lower speeds were comparable to noise in magnitude [44].

Image segmentation to background and foreground based on optical flow in the presence of camera motion is discussed in the paper by [45]. Their implementation assumes only translational motion of the camera that is allowed along the three axes. The system implemented starts by defining templates for a large number of translations. The templates show the predicted optical flow for different translations. A probabilistic model is used to fit the observed optical flow to the templates in order to conclude the segments of the image that belong to dynamic objects.

The authors in [45] used a mixture of models since many independent dynamic objects can exist in the scene. Therefore, a method is needed to determine the number of segments or clusters in the image automatically. Further, the zero motion segments have to be accounted for by declaring a pixel as zero motion (hence undefined angle) if its x and y components are less than a threshold. The method of [45] was developed to overcome the problems of image segmentation where pixels belonging to the same object at various depths is mistakenly segmented into different objects. Their work does not consider scenarios of camera rotation around vertical axis as well as heading calculation which is a major drawback.

A matched filter has been used by [46] to fine the FOE location. The idea is based on the fact that if a camera moves in the world coordinate towards a point, P, the FOE will be the projection of P into the image plane. Since by definition the optical flow at the FOE corresponding to point P in the world is zero, then all optical flow rays are divergent from that point. Therefore, the divergence property can be used to detect the location of the FOE even without knowing the magnitudes of the optical flow vectors. Authors in [46] propose the use of a matched filter which is a template of size 2w+1 by 2w+1 that has the origin at the center with each pixel (of the 2w+1 by 2w+1) in that template representing the expected angle of its optical flow vector if the FOE is at the center of the template. This is shown in figure 48.

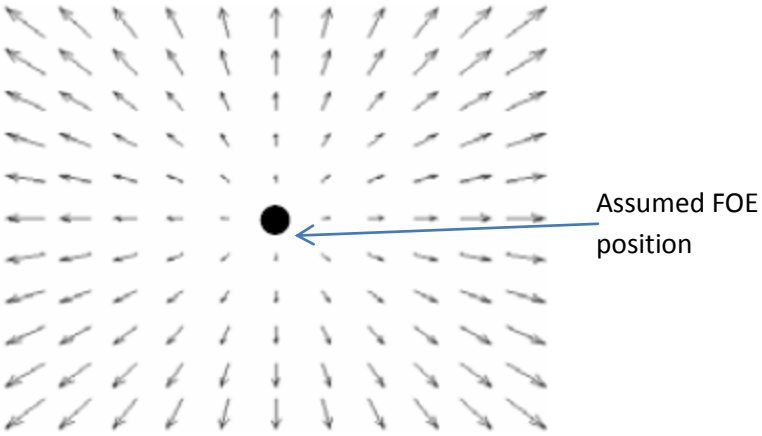


Figure 48: The template used to find the FOE from optical flow [46].

A weight function is used to emphasize the less noise affected far pixels (from the FOE). Different templates were prepared by the authors to take into account the different camera motions when the heading direction is different from perpendicular.

In reference [46]the equation relating motion field to optical flow was simplified to:

$$\begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{pmatrix} = \frac{\boldsymbol{f}}{\boldsymbol{Z}} \begin{pmatrix} \boldsymbol{xV_z - V_x} \\ \boldsymbol{yV_z - V_y} \end{pmatrix} \quad \textbf{(19)}$$

Where (u,v) is the optical flow vector, (x,y) is the pixel location in the image plane, f is the focal length, V is the camera velocity vector, and Z is the depth of the point (x,y). From this equation it is possible to estimate the depth to the scene. For accurate result, the equation is applied to the vicinity of pixels around the point under consideration and an average depth is found [46]. The method was tried on a real world image shown in figure 49.

Figure 49: The scene considered for application of algorithm. First frame (a) and second frame (b) in the video sequence from reference [46].

The algorithm was applied and the result is shown in figure 50. In part (a), the region of the FOE is circled and in part (b) the output from the algorithm is shown with FOE as a black dot.

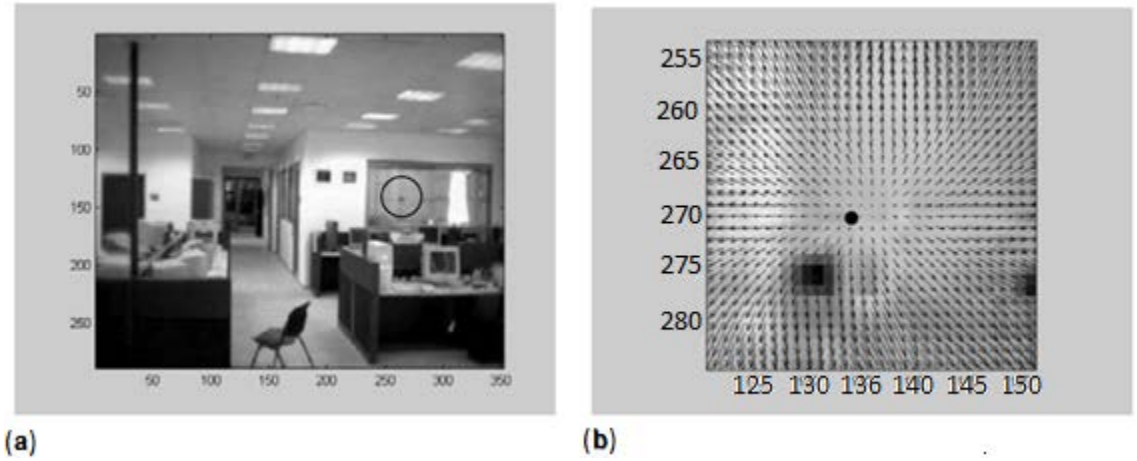


Figure 50: The result from the algorithm (a) real FOE (b) calculated FOE [46].

The actual FOE was at (136, 269) while the algorithm produced (132,273) after 500 iterations while More iterations refine the estimate to the actual.

The authors in [46] tested their algorithm with dynamic objects and noticed that the FOE position calculated was sensitive to dynamic objects. This means their technique may not be suitable for dynamic environments. Further, a large number of iteration was needed to reach any reasonable estimate of the FOE. Even after employing the refinement phase, 1500 were still needed. Thus their method may not be suitable for intensive processing applications like mobile robots which need fast algorithms to work in real time.

The authors in [47] propose an algorithm using mono camera and optical flow to navigate a robot while avoiding collision. The block diagram of the algorithm is shown in figure 51.

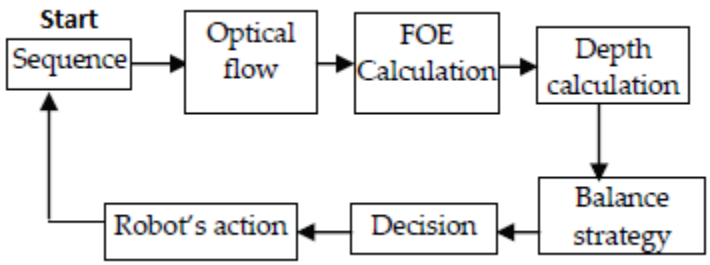


Figure 51:The block diagram for the visual obstacle avoidance algorithm [47].

In the first step the optical flow vectors are extracted from the image sequence and the FOE is calculated to help in depth map calculation which is then used to infer the decisions to be taken by the robot for obstacle avoidance. A standard Horn and Schunck method was used for optical flow calculation based on intensity conservation [47].

The control law developed was a function of the change in optical flow. The relation usedis given by:

$$\Delta \boldsymbol{F}_{\boldsymbol{internal}} = \boldsymbol{f}(\Delta \boldsymbol{flow}) = \frac{\sum|\boldsymbol{w}_{\boldsymbol{L}}| - \sum|\boldsymbol{w}_{\boldsymbol{R}}|}{\sum|\boldsymbol{w}_{\boldsymbol{L}}| + \sum|\boldsymbol{w}_{\boldsymbol{R}}|} \quad \textbf{(20)}$$

Where the change in internal forces acting on the robot wheels is a function of the difference in optical flow on the right (w_R) and left (w_L) sides. For calculating the location of the FOE, [47] used a counting scheme that relied on the divergence property of the optical flow vectors. Thee vectors were resolved into their components and the point from which maximum number of vectors diverged was considered the FOE. The vectors diverge from the FOE location thus the

point along x axis where the vectors change direction and similarly about y axis describe the coordinates of FOE.

Experiments were conducted in an indoors environment that contained no dynamic objects but typical indoors obstacles such as chairs, boards, and walls. The path followed by the robot is shown in figure 52 [47]. Further, the robot in motion with its path highlighted is shown in figure 53. The robot successfully avoided two obstacles in the right and left sides.

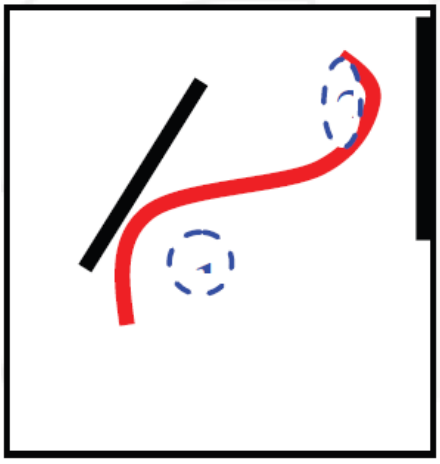


Figure 52: The robot path followed during experiments [47].

Figure 53: The robot in motion during the experiments avoiding obstacles at the board (left) and the radiator (right) [47].

The method proposed by [48] uses a sonar sensor equipped robot for indoors SLAM using line features in an EKF frame. The model used for the line features and the relation between local line properties to global line properties is used for predicting the observation [48]. The direct

control signal history that is needed for the EKF update was not available so difference in odometry readings was used. The authors of [48] propose two steps with the Hough transform applied initially to segment sonar data and the least square method is used to fit the line model.

The authors conducted experiments in an indoor environment having two boxes as obstacles, the environment measured 7.6m by 4.6m. The robot used was a P3Dx robot equipped with sonar sensors that acquired the readings at a rate of 3 Hz. The robot was driven by joy stick in a circular path. The authors presented the results of their experiments as a map of the environment with line features. The results are shown in figure 54 with the robot path plotted in circles and the extracted lines shown.

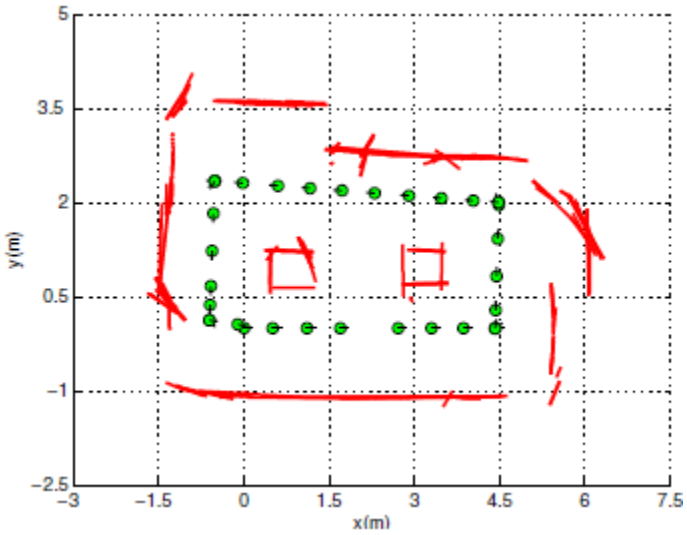


Figure 54: The results obtained by [48].

The method used by [48] combines the benefits of least mean squares method and the Hough transform and is suitable for sonar sensors with low computational load although the Hough transform may not be the best method due to false positive lines that result [79] .

Chapter 3

Design of the Hybrid Approach to SLAM

Different system architectures and functional modules for the purpose of developing the hybrid approach with the architecture shown in figure 55. Figure 55 shows functional modules that make up the system as blocks and the information flow between them with arrows. The process starts at the sensor level using sonar, laser, and camera sensors that produce sonar data, laser data, and camera data respectively which represent the sensory environment of the robot.

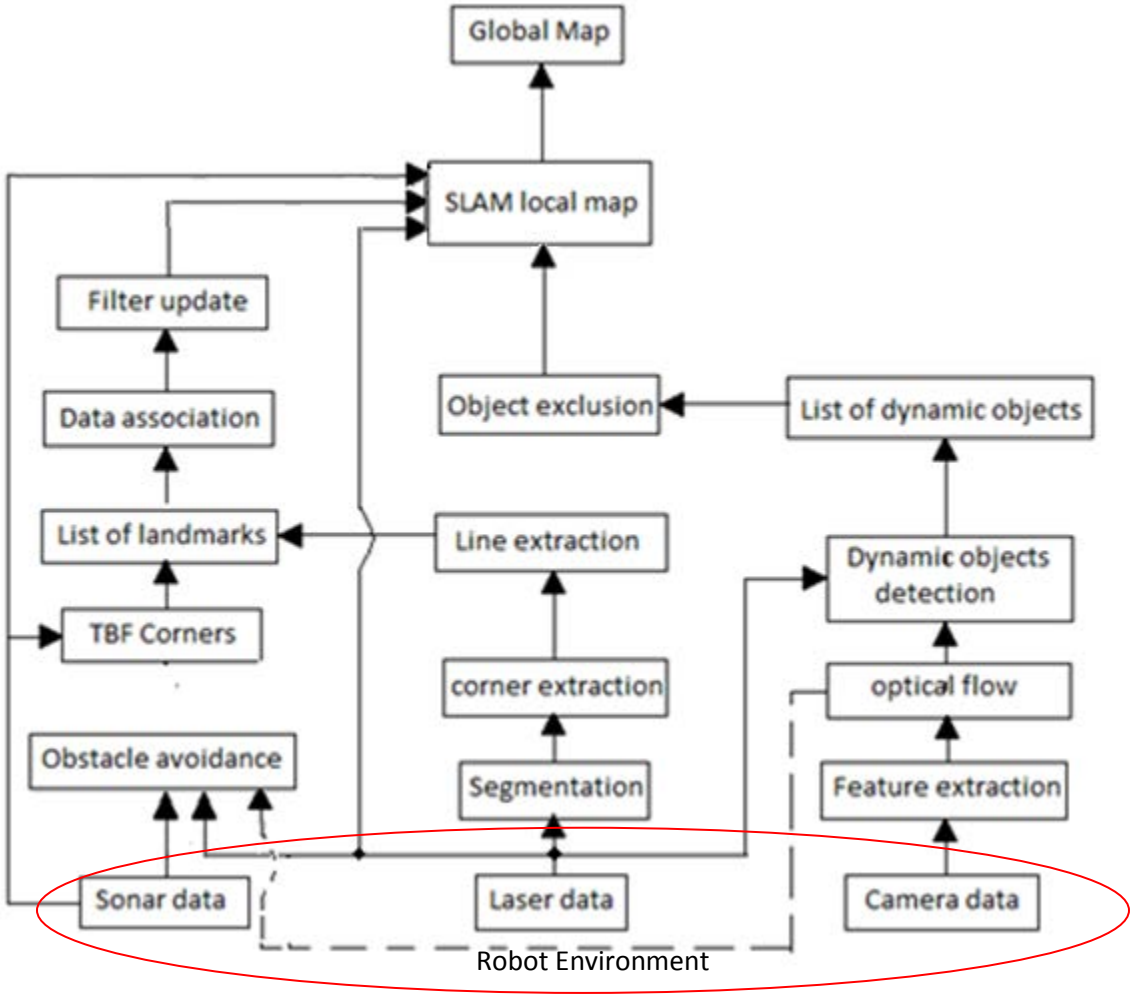


Figure 55: The layout for the system showing the interactions between various algorithms

The sensor blocks sonar data, laser data, and camera data describe the methods and functions developed to acquire and process the data from sensors for use by the robot. The output from sonar data is used to feed the TBF corners block, the obstacle avoidance block, and the local

SLAM map. The output from the laser data is used to feed obstacle avoidance, as well as feeding the local SLAM map, and feeding the blocks for corner extraction and line segmentation. The laser sensor sends data at the same time to dynamic objects detection block which feeds the list of dynamic objects that are then excluded from the SLAM local map.

The camera data is used for feature extraction to extract corners from image which are then tracked by the optical flow block that carries out the optical flow to feed the dynamic objects detection.

The lines extracted from the laser and the corners from the sonar feed the list of landmarks which via the data association are used to update the robot pose via the unscented Kalman filter update. This then affects the SLAM local map by modifying the robot pose. The list of landmarks block accepts corners from TBF block and lines from the line extraction by the laser sensor.

The obstacle avoidance block accepts data from the sonar sensor, laser sensor, and the camera. The fusion between sensors takes place in the obstacle avoidance block. The dynamic objects detection block accepts input from laser and camera to detect the presence of dynamic objects and feed them to the list of dynamic objects for exclusion from the SLAM local map. The SLAM local map accepts inputs from sonar sensors, laser sensor, and UKF update to build the map incrementally as well as data for positions of dynamic objects for exclusion of dynamic objects. The local map is then part of the global map which consists of tiles that are linked by adjacency according to their geometrical relations. The robot only updates the local SLAM map and switches to other tiles when needed.

The details of the blocks in figure 55 are explained in chapter 4 with the experimental tests and simulations carried out as well as showing the structure of the algorithm, the results of simulations and experiments, comparison with other approaches and the discussion of the results and difficulties faced whenever possible.

Chapter 4

Algorithms Developed and Results

4.1 robot calibration

4.1.1 Introduction

An important factor that strongly affects the operation of a mobile robot is the localization error which can lead to faults in mapping thus rendering maps inaccurate [50]. One frequent source of errors is systematic errors [49]. An odometry calibration method discussed in [49] is called UMBmark test was used to calibrate the robot. The result from the UMBmark test will be a cluster of end positions (x,y) for the CW run and the CCW run. To minimize the effect of non-systematic errors, the center of gravity of each cluster is calculated:

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \textbf{(21)}$$

$$\overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \quad \textbf{(22)}$$

Where x bar and y bar indicate the center of gravity. These can be used to quantify the magnitude of two errors in odometry. The first is the drift that occurs when the robot travels a straight path thus leading to a curved path, and the second which is the error in orientation when the robot rotates 90 degrees. The angles of these errors are called β and α respectively. Both are shown below in figure 56.

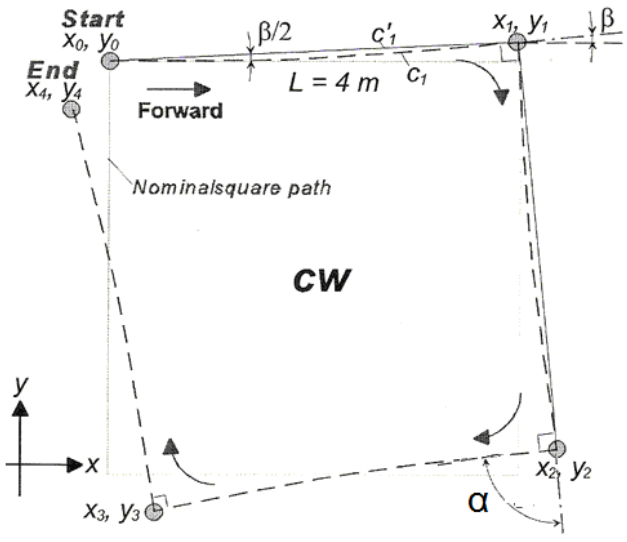


Figure 56: The angles beta and alpha that generate errors [49].

The angle alpha can be found using the difference in x error:

$$\boldsymbol{\alpha = -\frac{14.324(\bar{x}_{cw}+\bar{x}_{ccw})}{L}} \quad \textbf{(23)}$$

And the angle beta is given by:

$$\boldsymbol{\beta = -\frac{14.324(\bar{x}_{cw}-\bar{x}_{ccw})}{L}} \quad \textbf{(24)}$$

Once these values are calculated, correction factors can be worked out. If the robot takes an angle value as input then use the following:

$$\boldsymbol{\theta_{modified} = \theta_{requested}\left(\frac{90}{90-\alpha}\right)} \quad \textbf{(25)}$$

Where$\theta_{modified}$is the modified turn angle and$\theta_{requested}$is the angle the robot is supposed to turn. The error in beta causes the robot to move in circular path with radius, R:

$$\boldsymbol{R = \frac{L}{2\sin\left(\frac{\beta}{2}\right)}} \quad \textbf{(26)}$$

Where L is the length travelled. The ratio of wheel diameters that causes this can be calculated as:

$$\boldsymbol{d = \frac{R+\left(\frac{b}{2}\right)}{R-\left(\frac{b}{2}\right)}} \quad \textbf{(27)}$$

Where b is the distance between wheels and d is the ratio of wheel diameters [49]. Therefore, multiplying the encoder counts for each wheel by these factors will compensate for the error:

$$\boldsymbol{left\ encoder\ factor = \frac{2}{1+d}} \qquad \boldsymbol{right\ encoder\ factor = \frac{2}{\frac{1}{d}+1}} \quad \textbf{(28)}$$

Where d is the ratio of wheel diameters previously calculated in equation 27.

4.1.2 Calibration experiment

The P3DX robot to be calibrated was made to follow the square path in the UMBmark test with each side of the square having length 440 cm (4.4 meters). The test was carried out at different robot speeds of 50 mm/s, 100 mm/s, 200mm/s, 300mm/s, and 400mm/s to investigate the effects of speeds and it was noticed that the error increases slightly at low and high speeds.

The exact square path was marked using tape crosses as in figure 57.

Figure 57: The experimental setup.

The results are summarized in table 1 which shows the errors in cm. the experiment was repeated a minimum of three times for each speed. The average error in x and y is shown in the last two columns.

Table 1: summary of experimental results for CW run.

Speed-CW	X1	Y1	X2	Y2	X3	Y3	X4	Y4	$\bar{x}$	$\bar{y}$
50	-8.5	-5	-12	-12	-10.5	-10.5	-10	-11	-10.25	-9.625
100	-13	-9.5	-11	-10	-10.5	-14	-10	-8.5	-11.125	-10.5
200	-24	-19.5	-17	-19.5	-19.5	-17.5	-16.5	-17.5	-19.25	-18.5
300	-16	-13	-23	-21	-14	-14	-20	-18	-18.25	-16.5
400	16	13	19	23	19	20	18	18	-18	-18.6

The results for the counter clockwise experiments are summarized in table 2.

Table 2: summary of experimental results for CCW run

Speed-CCW	X1	Y1	X2	Y2	X3	Y3	X4	Y4	$\bar{x}$	$\bar{y}$
50	12	10	15	18	20.5	20	26	18	18.375	16.5
100	10	3	8	11	15	16	5	8	9.5	9.5
200	11	14	18	10	9.5	10	9.5	11	12	11.25
300	8.5	9	13.5	10	13.5	9	14.5	19	12.5	11.75
400	9.5	5	13	12	12	10	8.5	9	8.625	9

Based on these data, a value of 110,000 was calculated for R. Thus:

$$\boldsymbol{d = \frac{R+\left(\frac{b}{2}\right)}{R-\left(\frac{b}{2}\right)} = 1.0003} \quad \textbf{(29)}$$

Where the wheelbase, b, was measured to be 33.5 cm. The correction factors for each encoder are:

$$\boldsymbol{right = 0.9998 \quad left = 1.0001} \quad \textbf{(30)}$$

Or normalizing:

$$\boldsymbol{right = 1 \ left = 1.0003} \quad \textbf{(31)}$$

So the left encoder reading must be degraded by:

$$\boldsymbol{Degration = \frac{Left-Right}{Right} \times 100 = \frac{0.0003}{1} \times 100 = 0.03\%} \quad \textbf{(32)}$$

To do this correction, the parameters in the robot microcontroller must be modified using the robot ARCOS software commands. More about the functions used and the ARCOS client server architecture can be found in reference 51.

4.1.3Results

To confirm the calibration, the robot was asked to travel a distance of 4.4 meters in a straight line with and without the correction. A marker was attached to the robot to draw the line while the robot moves. Figure 58 shows the two paths before and after the correction.

Figure 58: The robot path before correction (green) and after correction (red).

As can be seen, the calibrated line is a straighter path and does not veer to the left like before. The distance of the lines from a known fixed reference were measured at the beginning and at the end. The errors before and after were 0.5 and 3 cm respectively.

To confirm the angle alpha, the robot was asked to rotate 3 turns (3X360) on the spot and the error was found and divided by 3. The error was found to be 0.8 degrees per turn i.e. 0.4 degrees per 180 degree turn. The number of revcounts in ARCOS was 16570 from this the desired new value is:

$$New\ counts = old\ revcounts + \frac{old\ revcount \times error}{180} = 16570 + \frac{16570 \times 0.4}{180} = 16607$$

Now this value can be used to help the robot maintain more certain odometry readings to longer distances. With the robot calibrated, the other processes in the SLAM algorithm can be executed with better accuracy.

4.2 Obstacle avoidance algorithm

The developed obstacle avoidance algorithm is a behavior based approach based on the subsumption architecture with two parts for the control of the rotational velocity and for the control of the translational velocity. In this implementation, three layers exist for controlling the translational velocity control and four layers for the rotational velocity control. More about the subsumption architecture is in [80].

4.2.1 The algorithm developed

i. Forward velocity control architecture

The architecture for forward velocity control is shown in figure 59. Each layer outputs a desired forward velocity. The target seeking layer controls the robot forward speed as it heads to target, the velocity control layer is for maintaining a safe velocity in the vicinity of obstacles, and the forward turning layer will slow the robot down if the desired turn radius to clear a front obstacle is beyond the robot capability. The output from the target seeking layer and velocity control layer are compared to pass the minimum of the two as the output. However, the forward turning layer can subsume this output with a different one.

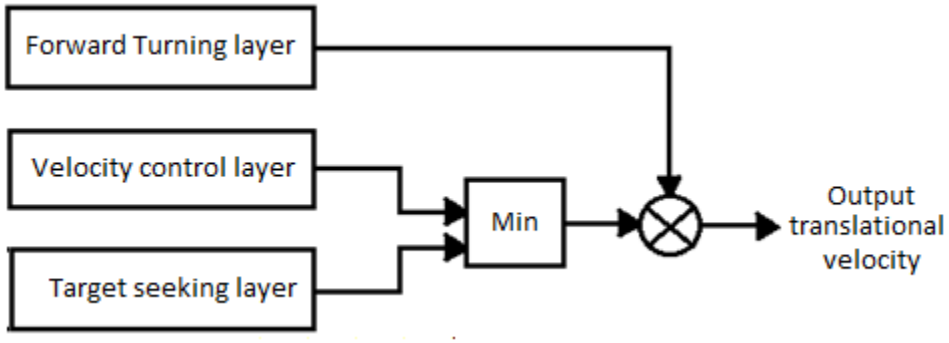


Figure 59: The architecture proposed for forward velocity control.

The target seeking layer outputs a velocity dependent on how far the robot is from the desired target location. The relation is given by:

$$\boldsymbol{desired\ velocity}\ (\boldsymbol{v_t}) = \mathbf{300}\left(\frac{\boldsymbol{d_t}}{\boldsymbol{d_t}+\mathbf{300}}\right) \quad \textbf{(33)}$$

Where d_t is the distance from the current robot location to the target location in millimeters, and v_t is the desired velocity from the target seeking layer in mm/s.

Optionally, an if statement could be coded to stop the robot once the output from target seeking layer drops below a certain value. The velocity control layer slows the robot down as obstacles get closer to the robot for safe motion. It works a space factor, f, for the environment ahead of the robot and multiplies it by 0.034 for desired velocity. Thus,

$$v_c = 0.034f = 0.034 \sum_{s=1}^{8} r_s sin\theta_s \quad (34)$$

Where r_s is the distance from the current sonar sensor, s, to the nearest obstacle in millimeters, θ_s is the orientation of the sonar on the robot, and v_c is the desired velocity from the velocity control layer in mm/s. The maximum possible value of parameter f in equation 34 occurs when all sonar sensors are reading maximum range of 5000 mm and is 14400mm which yields a maximum velocity of about 490mm/s. The translational velocity control architecture uses the output from the target seeking layer and compares it with that from the forward velocity control layer to find the minimum which is selected as the desired output velocity. The forward turning layer, discussed in section (i), is used to turn the robot as it approaches obstacles far enough before reaching obstacles. As the robot approaches the obstacle, the desired turn radius decreases up to the point where it becomes beyond the robot capability. In this case the robot has to turn in place or decrease velocity significantly. A flag is set by the forward turning layer when this happens and any value of robot translational velocity is subsumed with the value of zero.

ii. Rotational velocity control architecture

The architecture for the rotational velocity control is made from four layers as shown in figure 60.

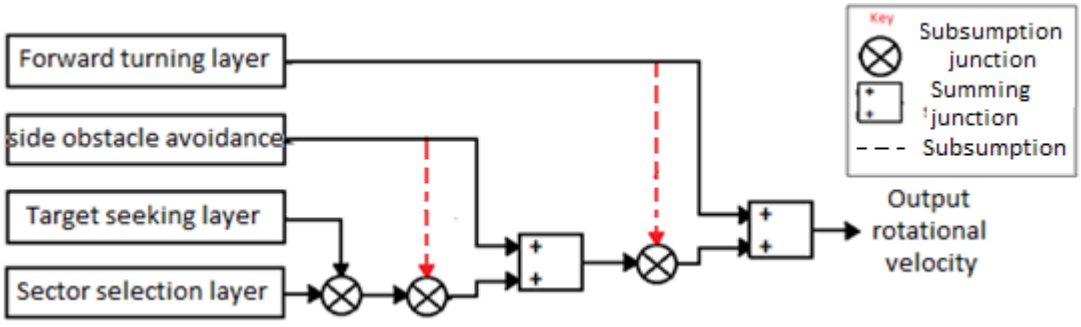


Figure 60: The architecture proposed for rotational velocity control.

In this architecture, the red lines indicate that top layer can subsume the output from the lower layer. For example, the target seeking layer always subsumes the sectors selecting layer. On the other hand, the side obstacle avoidance layer can subsume the output of two layers below (red line) or it can add its own output to theirs (black lines going to summing junction).

The sector selecting layer selects a sector (left or right) from the robot front that has the furthest obstacles in range and returns nothing if no clear sectors were found. The output is the heading leading to that direction. The target seeking layer produces a heading change to guide the robot to the bearing of the desired target. Therefore, it subsumes the outputs of the sectors selecting layer that are far from target and allows only the closest sector to target bearing. The nearest turning direction to target (CW versus CCW) is chosen as the output from the target seeking layer. The flow chart for sector selecting layer is discussed in figure 61.

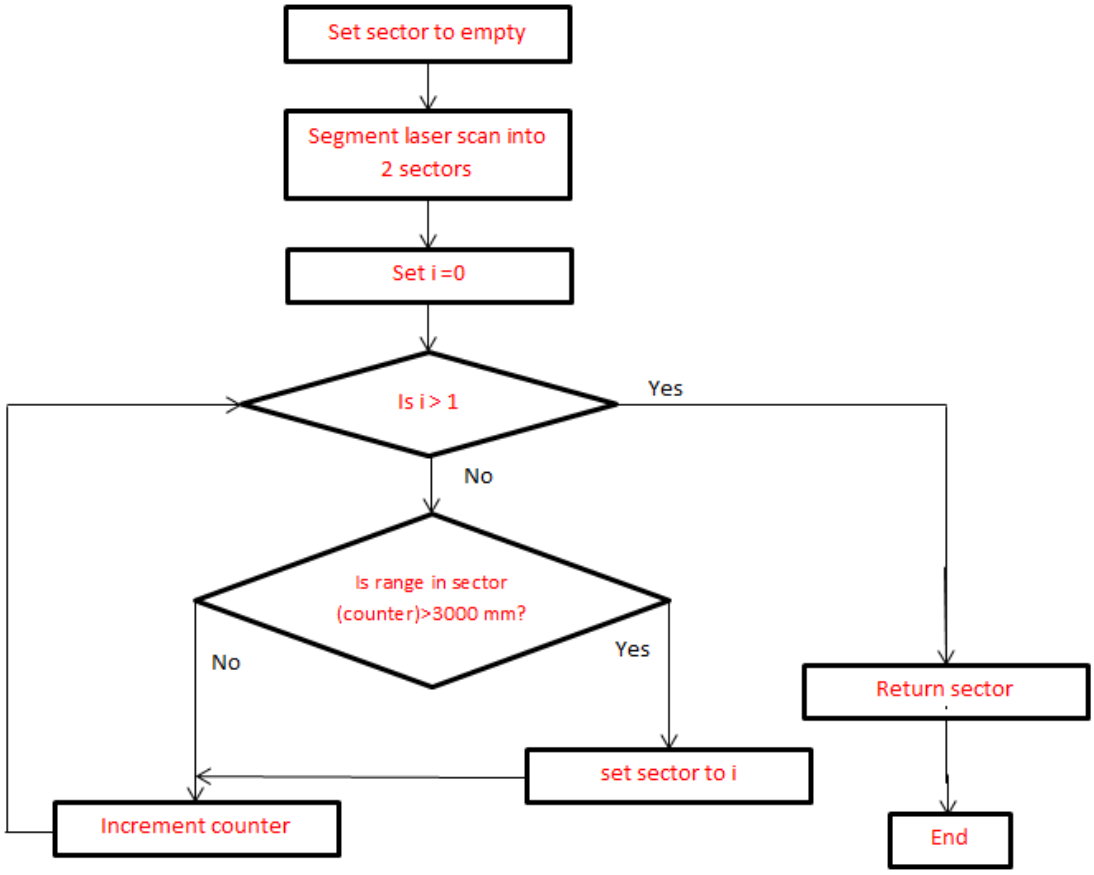


Figure 61: The flow chart for the sector finding layer.

The two sectors returned by the sector selecting layer are all subsumed except for the one in the same direction to the bearing of the target location with respect to the robot. In very crowded

environment, no sectors are returned in this case the default is assumed to be the sector directly in front of the robot i.e. the robot heading. The side obstacle layer operates when there are obstacles closer than a threshold to one side. In this case an immediate rotation command in opposite direction to that side is used and subsumes the previous layers. The flow chart is shown in figure 62.

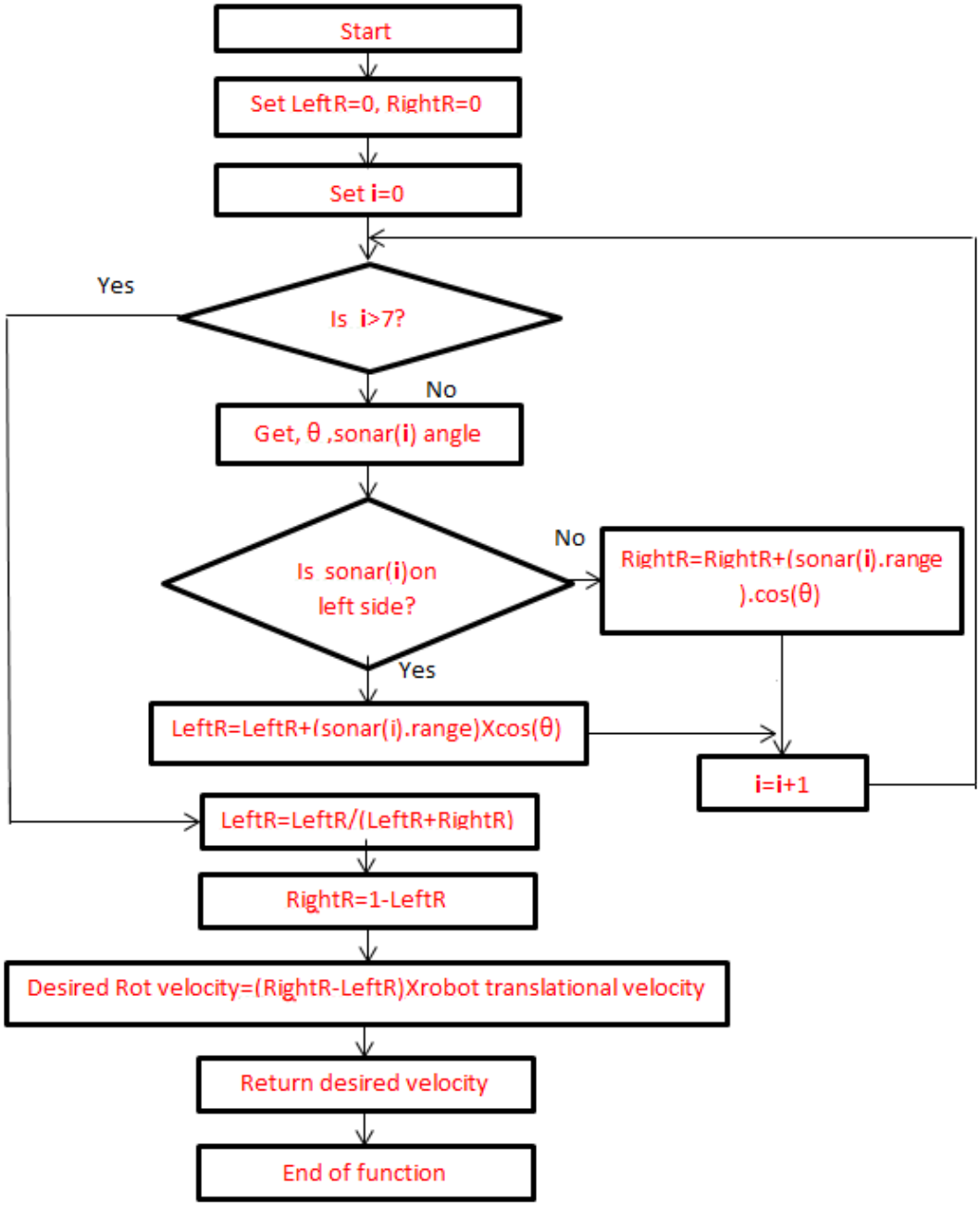


Figure 62: The flow chart for the side obstacle avoidance layer.

The forward turning layer turns the robot away from obstacles that it faces directly ahead at close range (less than 500mm) as well as getting the robot out of corners. If the layer detects that the distance ahead of the robot is less than 500mm from sensor position or that the robot is trapped in a corner, it commands the robot to move backwards and rotate until it has rotated at least 35 degrees away from obstacle or 150mm back away from obstacle. The corner detection is carried out using a developed feed forward non-linear neural network that was trained in MATLAB. The network was trained using data from real situations in which the robot was placed near corners and the sonar data recorded. If the forward turning layer detects a corner or close obstacle ahead it will subsume all other layers fully. Otherwise, it will subsume the layers partially by assigning weights to the layers. The sector selection layer is given a weight, w_s, that is described by:

$$\boldsymbol{w_s = 0.5\left(1 + \tanh\left(\frac{3R}{1000} - 9\right)\right)} \quad \textbf{(35)}$$

Where R is the range to nearest obstacle. The side avoidance layer is given a weight, W_a, which is:

$$\boldsymbol{w_s = \frac{160 - \sqrt{R}}{160}} \quad \textbf{(36)}$$

Where R is the range to nearest obstacle. Note that this weight will never be negative since the maximum sonar range is 5000mm. The equations for weights were chosen based on experimentation to find the most suitable relations for best performance. However, there is scope for improvement and more testing in various environments. The reason for the empirical nature is due to the combined properties of sensors and the subsumption architecture that give rise to a non-deterministic outcome (competing behavior) [80] that needs a lot of testing to understand likely outcome.

The flow chart for the forward turning layer is in figure 63 and the weights set by equations 35 and 36 are displayed graphically versus range to nearest obstacle for visualization in figure 64.

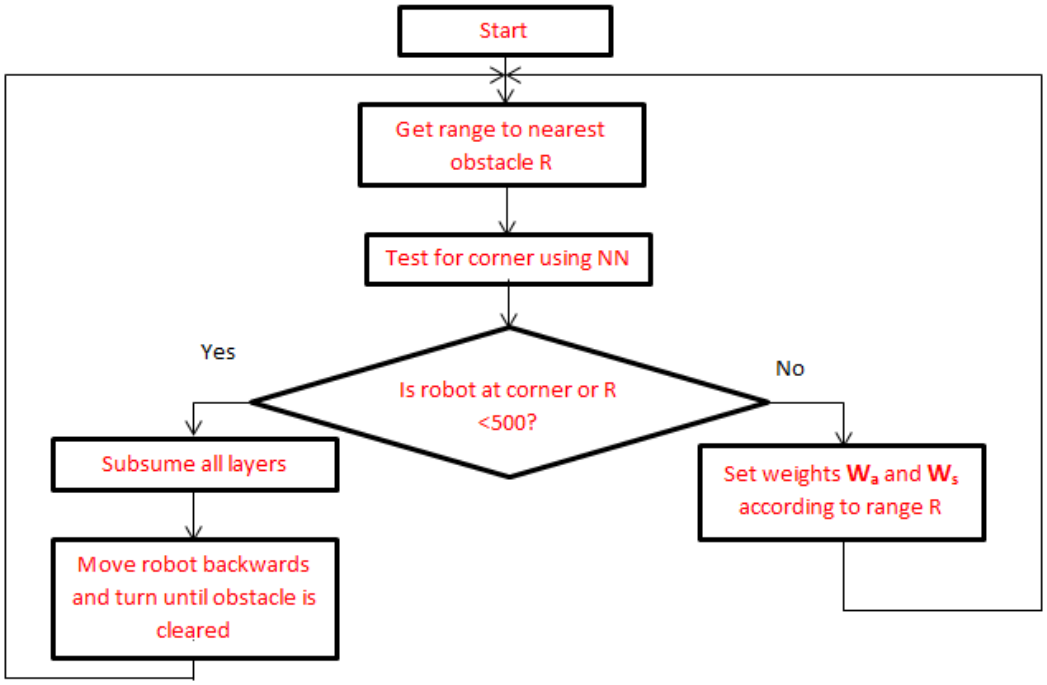


Figure 63: The flow chart for the forward turning layer.

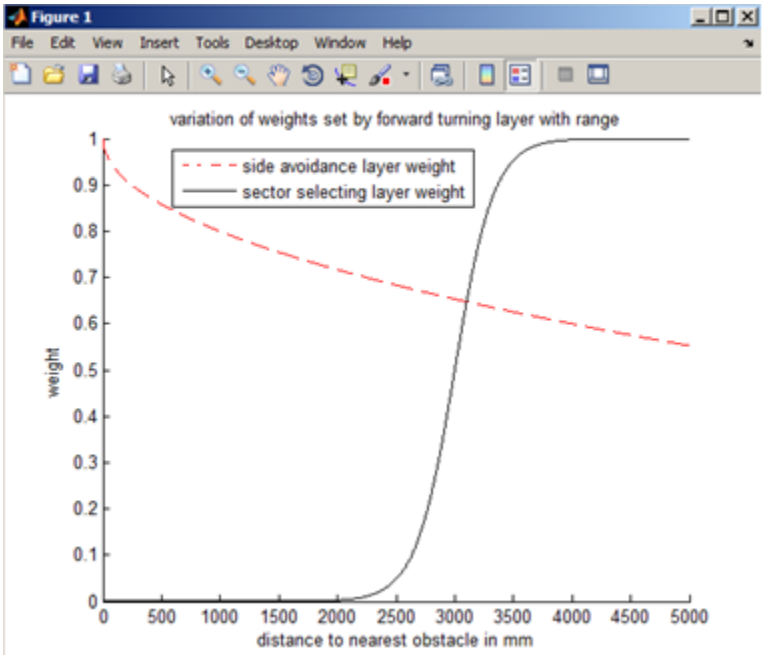


Figure 64: The variation of weights set by forward turning layer by range.

4.2.2 Simulation results

Simulations were carried out using the SDK supplied by MobileRobots with a model P3Dx differential drive robot. The default map of Columbia lab supplied by MobileSim was used and is shown in figure 65.

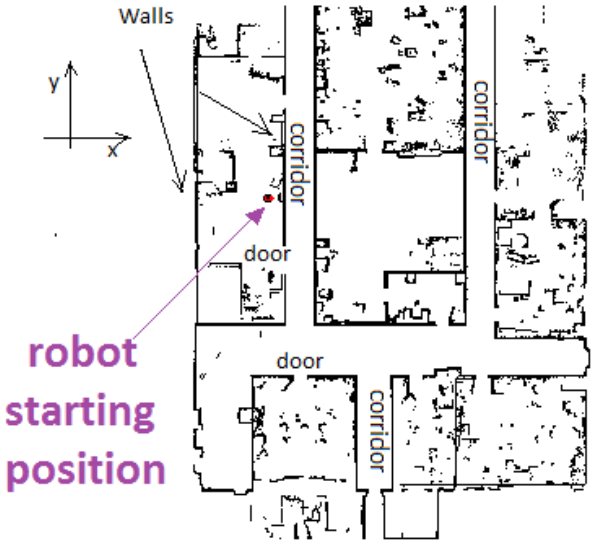


Figure 65: Map of the columbia lap showing robot starting position.

The map was zoomed to show the starting area details and the robot was commanded to go to different locations at different trials. In the first simulation trial, the robot was commanded to move to the location (-500, 500). This location will be to the left and north of the starting position. The robot path showing how it avoided obstacles is shown in figure 66.

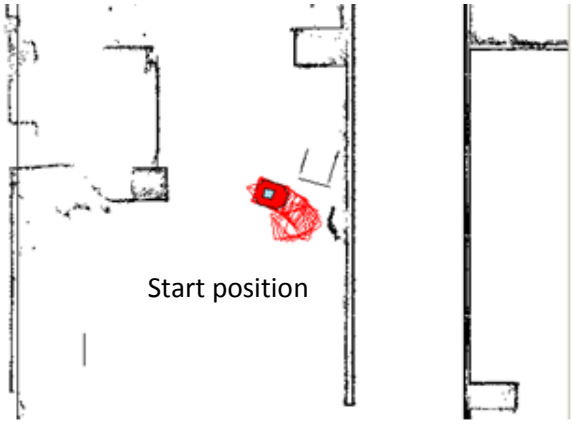


Figure 66:The robot path followed to reach desired target of (-500,500).

The robot starts facing an obstacle (lower part of a drawer) and thus cannot move straight so initially it will turn back. Further, the target seeking layer directs the robot to turn in the direction that faces the target left of current position.

The robot was then commanded in a second trial to move to the location (-500, 3000). However, this time the robot will have to clear obstacles near the target location as well as the starting position as shown in figure 67. The robot starts slowly due to the proximity of obstacles but as the obstacles are cleared and it moves to the free space it speeds up. Further, notice how the robot slows down as it approaches the target evident by the distance between the trails.

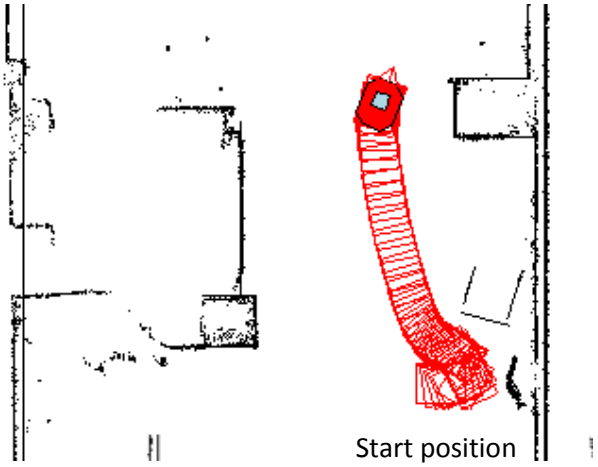


Figure 67: The robot path to location (-500, 3000).

The robot follows a curved path to obstacle (not shortest path) as it moves reactively based on subsumption architecture. This is one drawback of using a reactive system.

The robot was then commanded in a different trial to go to target (100, 3500) to see how it could reach a target obscured by obstacles. The path is shown in figure 68. Notice how the robot was turned away from the obstacle despite the target seeking layer requesting a direction towards the obstacle until the obstacle was cleared.

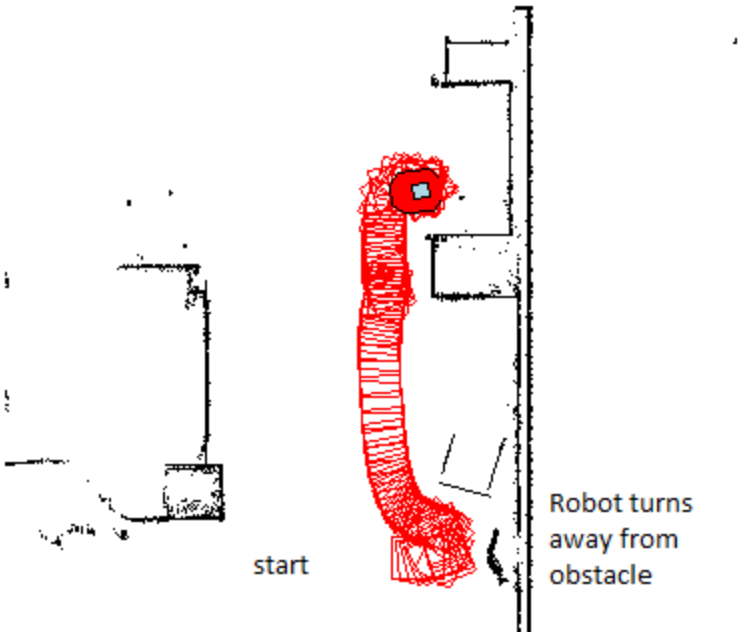


Figure 68: The robot path to location (100, 3500).

4.2.3 Discussion

From the simulation trials, it is concluded that the obstacle avoidance layer operates as desired while not relying on demanding computations. The subsumption architecture was used in a modified way to enable the robot to accept commands to desired location. This is because in the basic subsumption architecture a robot cannot plan for motion to go to a specific target. Therefore, the system was modified to enable this by adding a target seeking layer. Further, the conflict between layers was resolved by using equations to relate the strength of output from each layer rather than the traditional on/off style used in pure subsumption architecture. Moreover, the simple approach which does not require a map beforehand nor does it require the robot to map a grid map enables operation in any place and situation. Finally it is easier to integrate the laser and sonar sensors together for more robustness in environment.

4.3 Sonar corner detection

A triangulation based fusion (TBF) of sonar data to extract point features such as corners from the environment was proposed by the authors in [38]. The TBF is a voting scheme that triangulates sonar readings at different robot position thus the accuracy is limited by odometric drift. References [38] and [37] both discuss the use of TBF method to enhance sonar readings. However, reference [37] adds some modifications to the method. Section 4.3.2 will start by explaining the method by [38] while the approach of [37] to addressing some of the limitations of TBF was mentioned in chapter 2. Further, a mistake in the equation of TBF by reference [38] will be pointed out.

4.3.1. TBF algorithm

In the TBF algorithm, the sonar readings are not used instantly but are stored in a sliding window table. The sonar sensors are read every certain time period or when the robot has moved a predetermined distance. The TBF has been mentioned in chapter 2 when discussing a review of the work by [38]. It is claimed by [38] that even before solving intersection equations, the method of working out the intersection between two arcs based on center point gives good results. However, this could not be reproduced in the simulations carried out in this thesis. Further, the equations presented by the authors in [38] were found to yield mathematically wrong answers as shown in section 4.3.3 of this chapter.

4.3.2 TBF Formulation

The geometry of TBF is shown in figure 69.

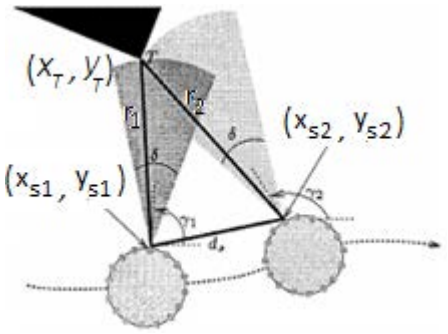


Figure 69: The triangulation geometry [38].

Assuming the sonar can be perfectly modeled as a circular arc, accordingly the intersection of circular arcs is likely to be due to an edge. This way the precise location of the edge can be

known even with wide sonar resolution. The edge satisfies equations 11 and 12 and solving these two equations simultaneously yields corner or edge location.

The following solution to this simultaneous equation problem is presented by reference [38] as:

$$\boldsymbol{x_T = x_{s_1} + \frac{1}{d_s^2}\left(d_{x_s} d_r^2 \pm |d_{y_s}| \sqrt{r_2^2 d_s^2 - d_r^4}\right)} \quad \textbf{(37)}$$

$$\boldsymbol{y_T = y_{s_1} + \frac{1}{d_s^2}\left(d_{y_s} d_r^2 \pm |d_{x_s}| \sqrt{r_2^2 d_s^2 - d_r^4}\right)} \quad \textbf{(38)}$$

Where:

$$\boldsymbol{d_{x_s} = x_{s_1} - x_{s_2}} \quad \textbf{(39)}$$

$$\boldsymbol{d_{y_s} = y_{s_1} - y_{s_2}} \quad \textbf{(40)}$$

$$\boldsymbol{d_s^2 = d_{x_s}^2 + d_{y_s}^2} \quad \textbf{(41)}$$

$$\boldsymbol{d_r^2 = \frac{r_1^2 - r_2^2 - d_s^2}{2}} \quad \textbf{(42)}$$

However, this solution was tested and was found to yield wrong solution. In MATLAB a function, CIRCCIRC can be used to find the intersection of two circles together and was used to confirm the wrong equations of [38].Therefore, a solution was searched for as the correct answer and the following solution was found for the intersection of two circles from the website [52]:

$$\boldsymbol{x_T = x_{s_1} - \frac{1}{\sqrt{d_s^2}}\left(d_{x_s} k \pm -d_{y_s} \sqrt{r_1^2 - k^2}\right)} \quad \textbf{(43)}$$

$$\boldsymbol{y_T = y_{s_1} - \frac{1}{\sqrt{d_s^2}}\left(d_{y_s} k \mp -d_{x_s} \sqrt{r_1^2 - k^2}\right)} \quad \textbf{(44)}$$

Where:

$$\boldsymbol{d_{x_s} = x_{s_1} - x_{s_2}} \quad \textbf{(45)}$$

$$\boldsymbol{d_{y_s} = y_{s_1} - y_{s_2}} \quad \textbf{(46)}$$

$$\boldsymbol{d_s^2 = d_{x_s}^2 + d_{y_s}^2} \quad \textbf{(47)}$$

$$\boldsymbol{d_r^2 = \frac{r_1^2 - r_2^2 - d_s^2}{2}} \quad \textbf{(48)}$$

$$k = \frac{d_s^2 + r_1^2 - r_2^2}{2\sqrt{d_s^2}} \quad (49)$$

4.3.3 Confirmation using Simulation trial

The algorithm by [38] was tested by data supplied from the robot simulator with the following environment shown in figure 70. The robot was run and the results were plotted in figure 71 with the sonar readings shown in red and the path of the sonar sensors shown in black. There are 10detectable corners in the figure71.

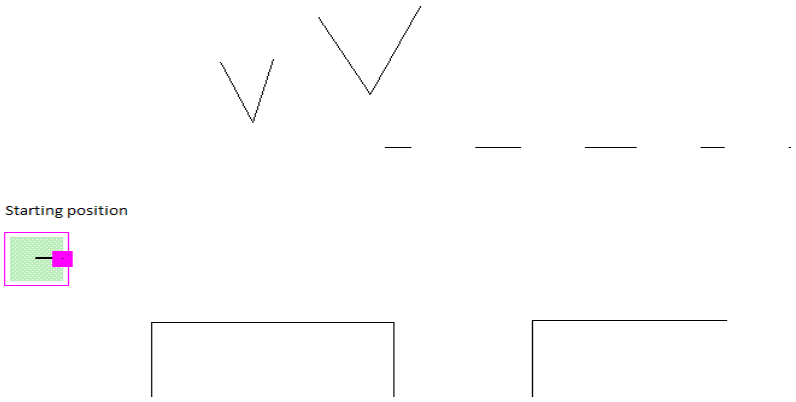


Figure 70: The simulation environment developed.

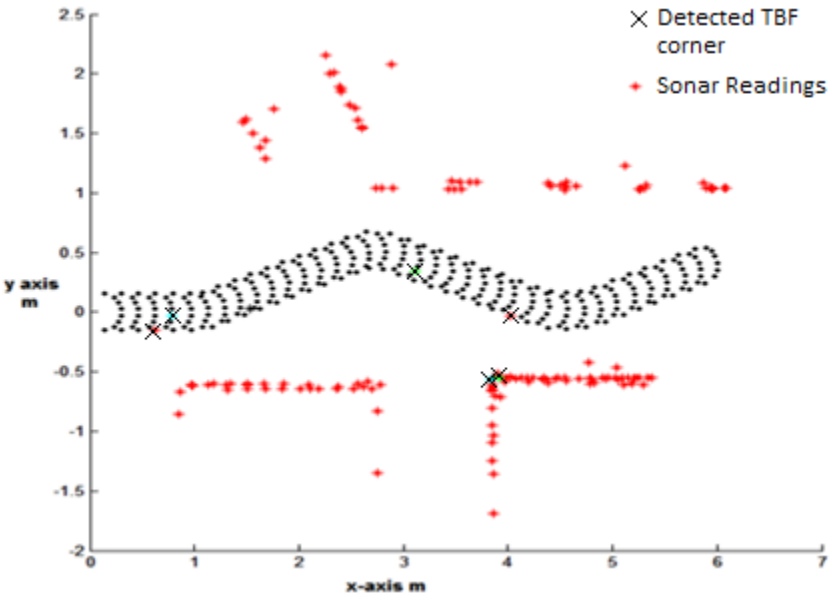


Figure 71: the environment under consideration for testing equations from [38].

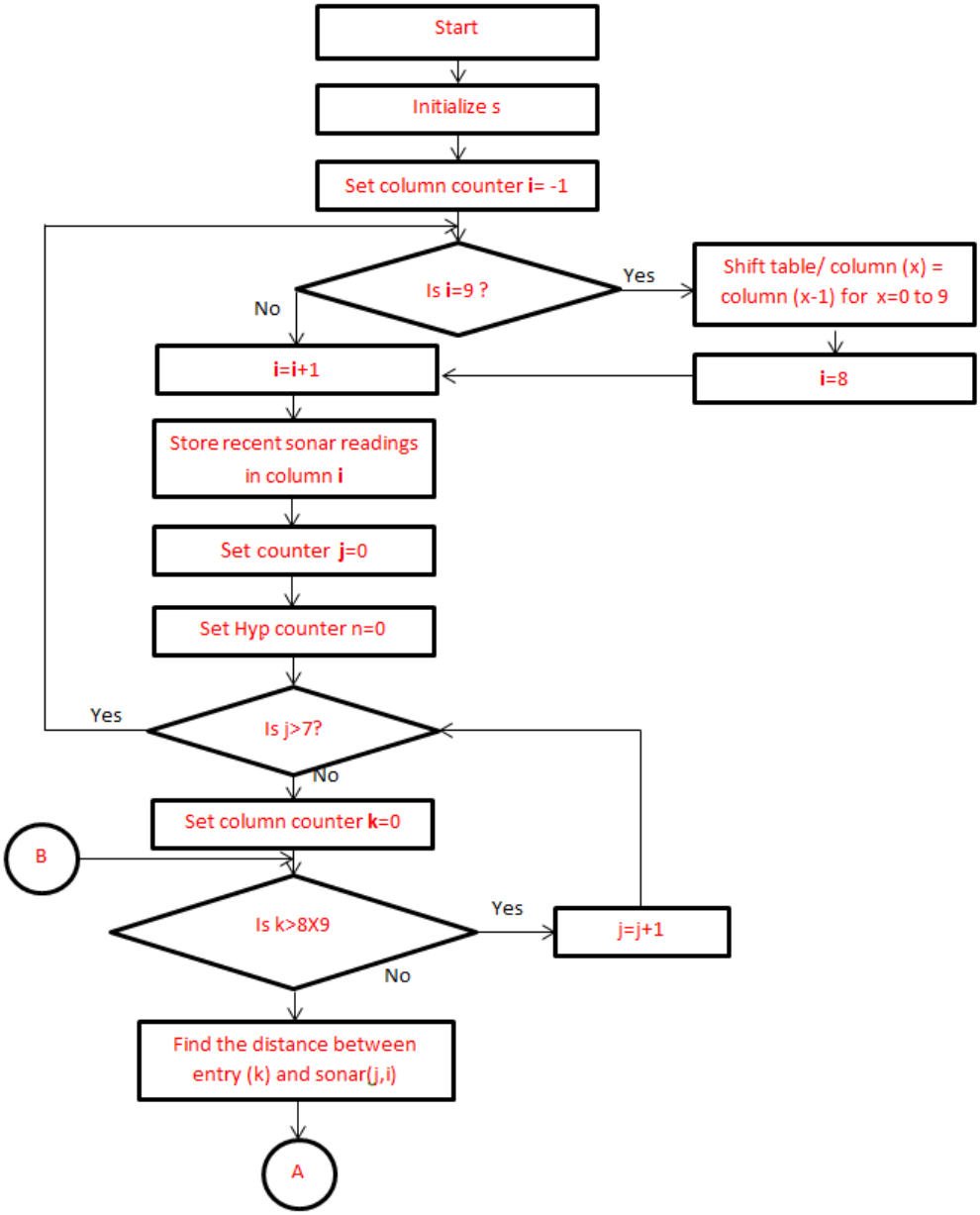
The results of corners were then displayed (as found by TBF equations of reference [38]) as:

corn =[2.3020 2.4028, 2.6003 2.1206, 2.7207 2.2877]

Only the first corner is close enough and the other two corners are wrong cornersnot all corners were detected as the equations by [38] were wrong so the trial failed to detect corners.

4.3.4. TBF algorithm flowchart

The flowchart for the TBF algorithm used is shown in figure 72.A sliding window is used to sort readings and the triangulation process is carried out if the sonar readings are within 300 mm of each other. If the result from the triangulation is within the sonar cone, a hypothesis counter, n, is incremented and the result of triangulation with hypothesis counter is stored as output.


Start
Initialize s
Set column counter i= -1
Is i=9 ?
Yes
No
Shift table/ column (x) = column (x-1) for x=0 to 9
i=8
i=i+1
Store recent sonar readings in column i
Set counter j=0
Set Hyp counter n=0
Is j>7?
Yes
No
Set column counter k=0
B
Is k>8X9
Yes
No
j=j+1
Find the distance between entry (k) and sonar(j,i)
A

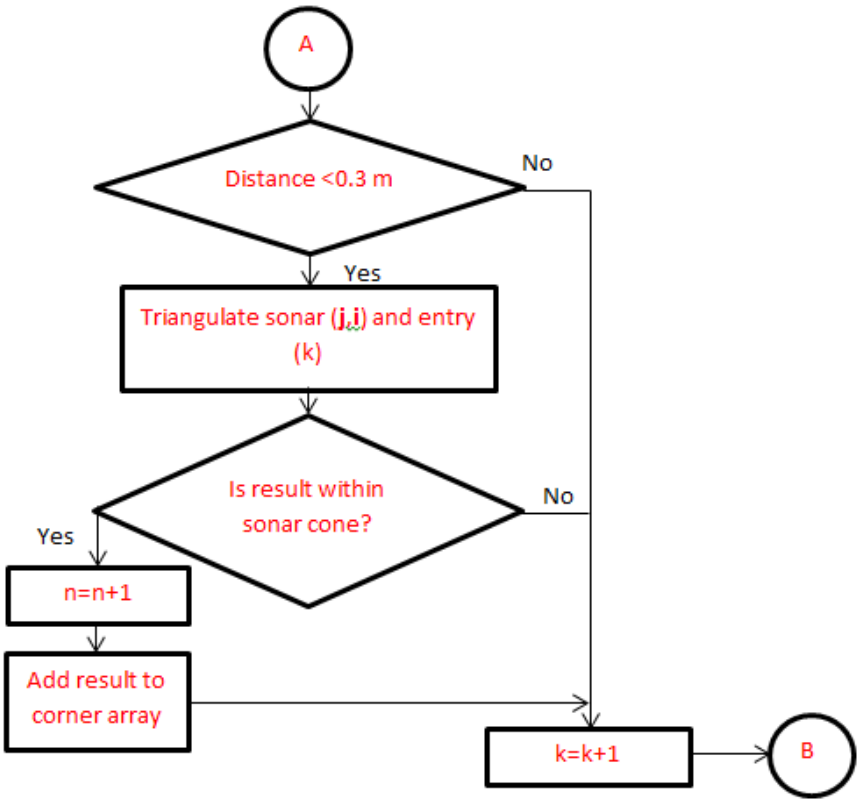


Figure 72: The flow chart for TBF for sonar data to extract corners.

In figure 72 counter k is compared to 8 by 9 which is the size of the sliding window sonar columns to triangulate with. The corner array then takes the triangulations with the corresponding counter in each triangulation and ensures that successive triangulations are closer to each other than previous ones. The triangulation process in the flow chart is simply the solution to the sonar arcs intersection equations.

4.3.5 Simulations

The algorithm was implemented in MATLAB using data from robot simulator Mobilesim in the environment shown in figure 73. The robot moved in a turning path while recording sonar readings and the pose at each reading.

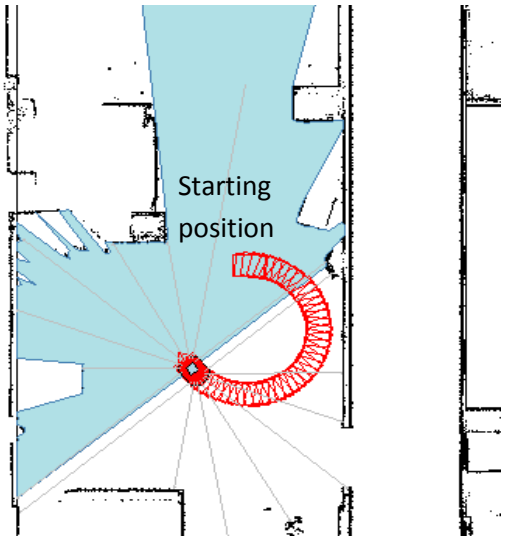


Figure 73: The path followed by the robot in the simulation.

The sonar readings obtained together with sonar pose were plotted for each individual sonar sensor. The red dots represent sonar sensors positions (on the robot) along the trajectory while the black dots are readings of those sensors and the sonar readings due to specular reflections are shown as cyan dots while the sonars that display them are blue. This is in figure 74.

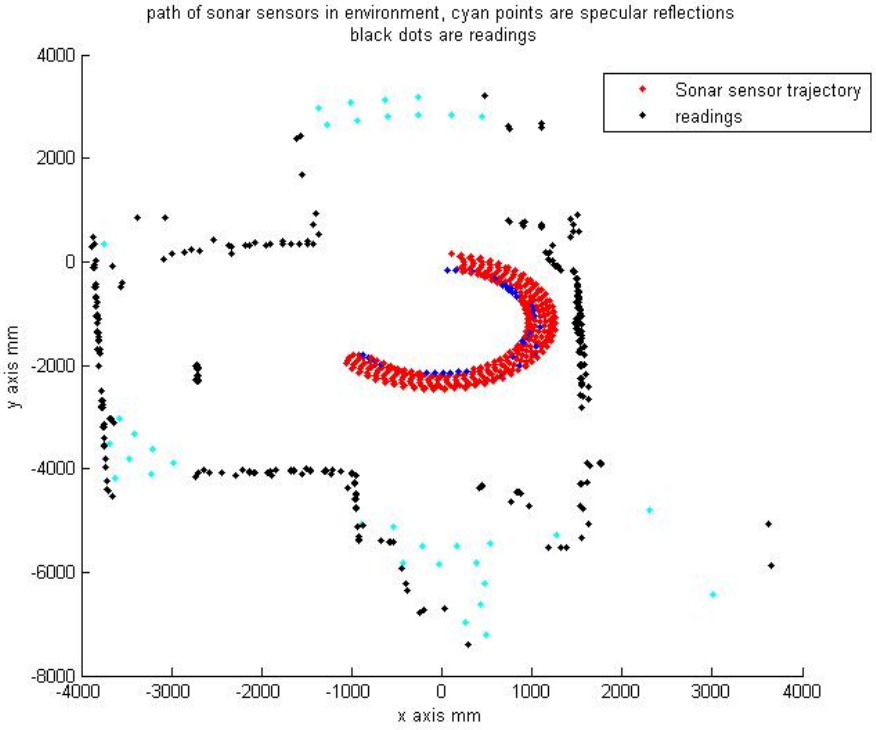


Figure 74: sonar path and the environment plotted in MATLAB according to data obtained from simulation.

The TBF algorithm was run however, only sonar intersections with angle differences (between triangulated corner and sonar beam axis) greater than 20 degrees were chosen as corners. This was mentioned by [37] as improvements to the basic TBF where intersections with higher angle difference between sonar sensors are called stable intersections. The corners from TBF were plotted as asterisk with color cycling as shown in figure 75.

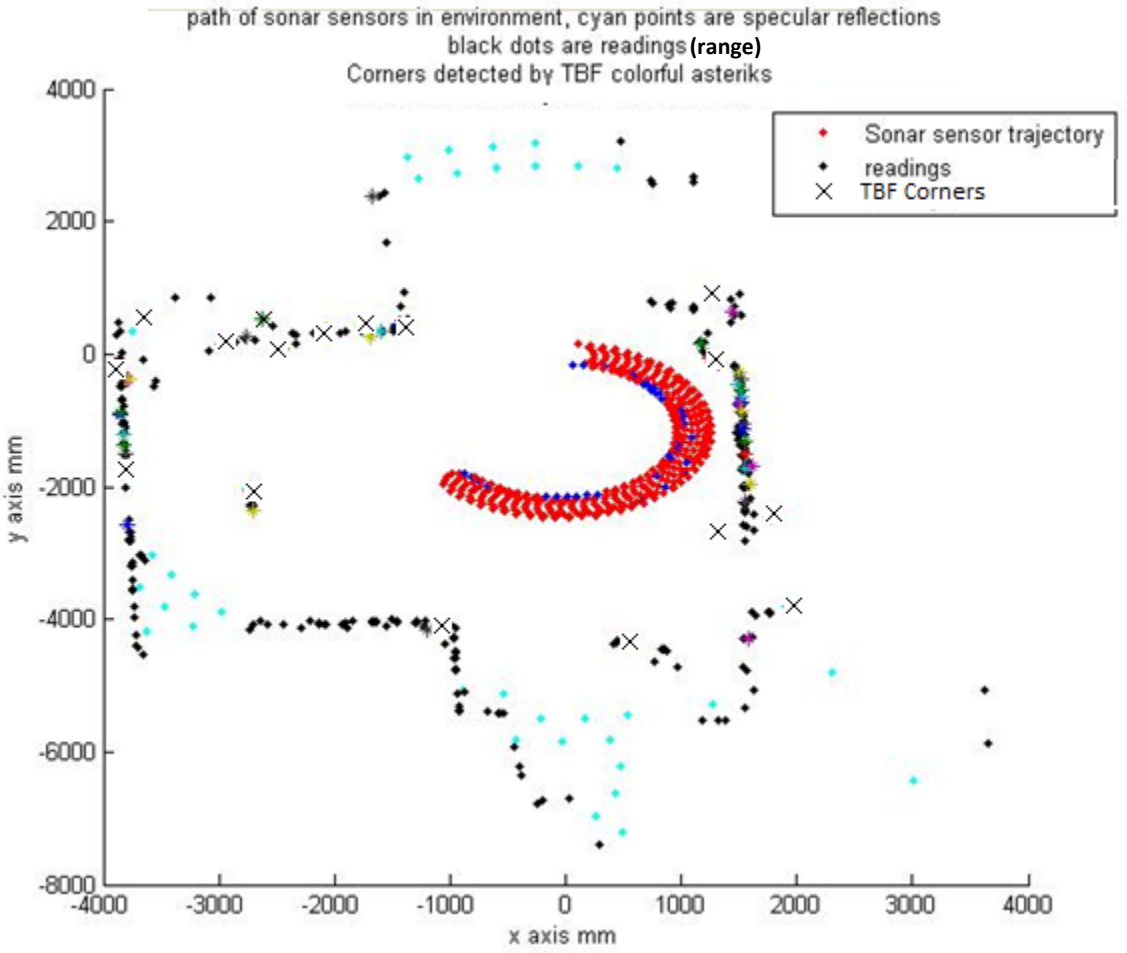


Figure 75: corners obtained from the TBF alogrithm for this simulation.

Some of the corners from TBF were matching the actual corners on the map but many readings on a line were mistaken as corners. To try to improve this, the angle of stable intersection was increased from 20 degrees to 40 degrees but the improvement in rejecting false corners was not significant.

To find out more, the map was zoomed in as shown in figure 76 using the same colors for various points as previously outlined.

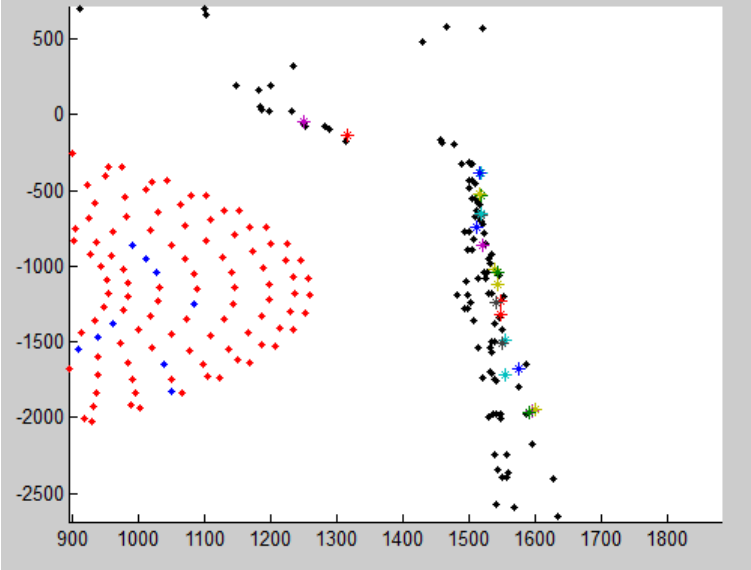


Figure 76: A zoomed in view from figure 75.

As can be seen, the sonar readings are not sharp but the line boundary is fuzzy thus making the algorithm display corners that do not exist. Therefore, a method is needed is to refine the corners. One way that was tested is to apply a low pass filter but this was found to distort the true readings. Another approach is mentioned by [38] using local grid maps to better localize corners. However, this is computationally intensive and nullifies the power of using sonar i.e. computational efficiency. Further, the corners produced from processing of laser data is much more accurate than that from sonar while not being computationally expensive.

4.3.6 Experimental trials

The robot was tested in a lab environment with the approximate layout shown in figure 77.

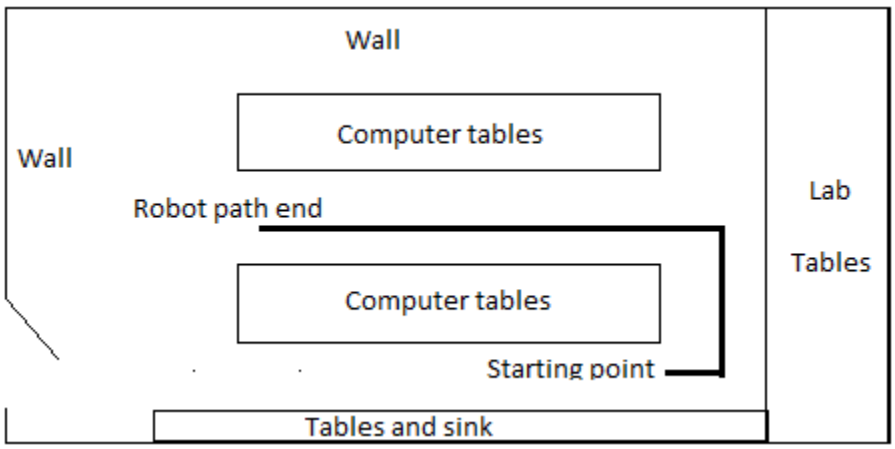


Figure 77: An approximate layout of the experimental environment.

As can be seen, the sonar readings (red dots) are noisy looking different from the actual environment. The robot path is shown in black and the readings in red. The robot path is drifting despite the very short distance (~8m) due to localization errors as the full SLAM was not operating.

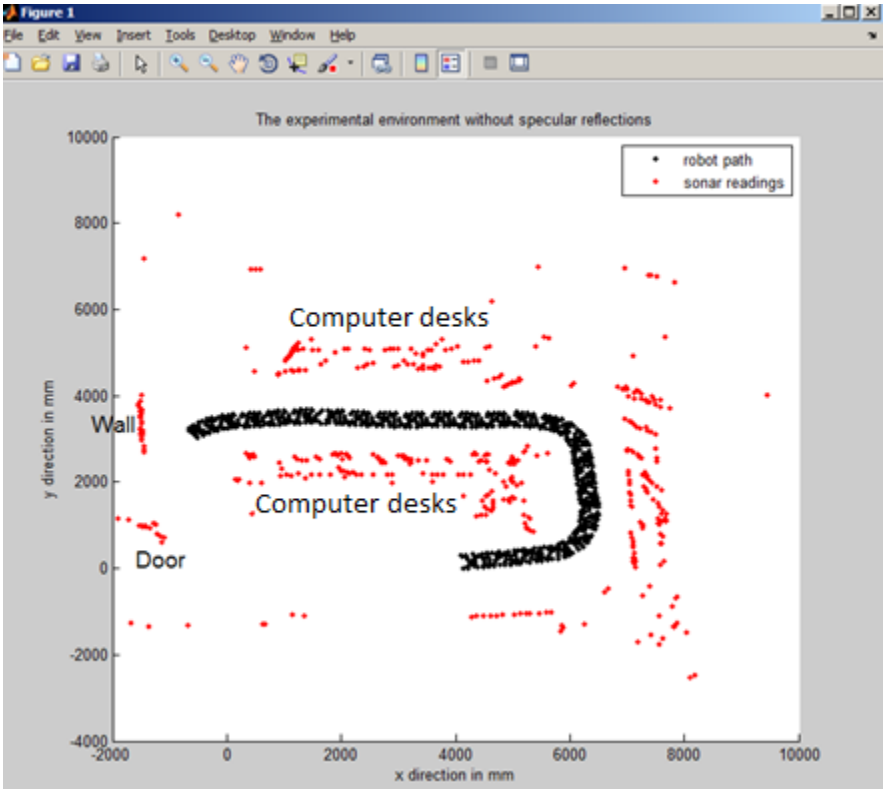


Figure 78: The experimental environment after removal of potential specular reflections

The chairs appear as points or as a row of inclined points. This set of data was used to extract corners and the results are shown in figure 79 where the corners are indicated by a green asterisk, the robot path in black, and the readings in red. It must be noted that the drift due to odometry will not affect the TBF algorithm as it operates over small distances.

Even after applying the stable intersection method, the sonar corners are not accurate and exact. Therefore, it was decided to fuse them with high weight laser sensor which has better resolution to benefit from both sensors. The fusion uses weights for both sensors favoring the laser corners.

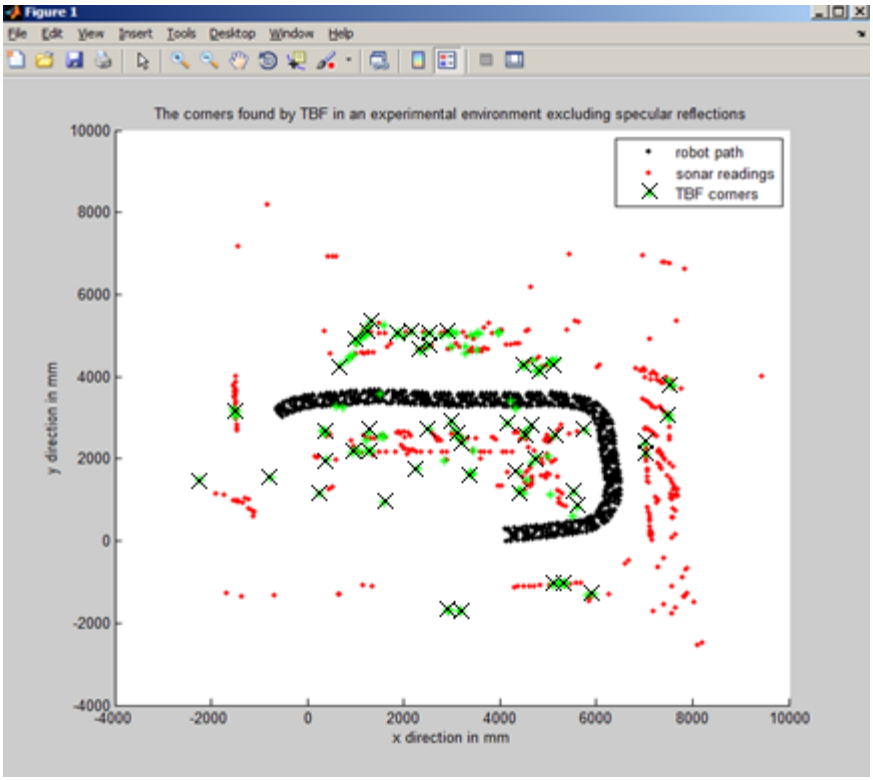


Figure 79: The corners extracted by TBF algorithm.

4.3.7 Discussion

Clearly the sonar data is noisy and there is uncertainty in the origin of a reading which shows itself in the corners found by TBF in figure 79. Many of the corners correspond to corners found in the actual environment but some of them are false positives. Additionally, many corners are re-extracted and identified as new corner only because they are slightly displaced from a previously found corner. This can be solved by applying an algorithm similar to the merge phase of the split and merge algorithm or by a clustering algorithm that clusters close readings together then represents them by a nominal corner that could be based on the centroid of the cluster. However, all those methods increase complexity of both the algorithm and the tuning of the algorithm for better performance.

4.4 Scan matching

4.4.1 Introduction

Scan matching is a technique used to find the relation (rotation and translation) between two sets of points such that maximum overlap occurs. The two sets can be a scan contour and a known map (position estimation) or a current scan contour and the previous scan contour (motion estimation). Dead reckoning can be employed to provide an initial guess for faster convergence [40].

Scan matching can be performed via three main approaches. These are:

1. Feature based approaches where certain features such as lines or corners are extracted and used for the scan matching,
2. Point matching techniques that directly find correspondence between points in the set of two scans, and
3. Data methods that extract certain mathematical properties from the two scans.

The first and second approaches are common in literature. Point matching techniques are best used with unstructured environments where there are no features available for extraction. Due to limitations imposed by the inaccuracies in sensors scan matching is an optimization problem to minimize error[40]. For point based techniques different methods have been proposed such as iterative closest point (ICP), gradient computation, simulated annealing, and genetic algorithm. The optimization can be carried out in polar or Cartesian coordinates. Given two laser frames or maps at two different poses, the aim is to pair the points in the first frame to the second frame for finding the transformation between frames.

Feature based method was used in this work but it was compared to one widely used point to point matching so point to point matching is discussed briefly.

The laser scan is made up of points that have range, d, and angle, α. The time step at which the frame is taken is denoted by a subscript, k and k+1 for the two consecutive frames. So the laser frame at time k is:

$$\boldsymbol{q_k} = \{\boldsymbol{d_k}, \boldsymbol{\alpha_k}\} \quad \textbf{(50)}$$

Where d is the range and α is the bearing (polar coordinates). The scan is arranged as a set of angles at discrete intervals separated by angular resolution ρ. The scanning sequence is represented by j where j is between 0 an N, N being the maximum number of scan angles (dependent on resolution). The angle of any point in the scan can be worked out if the resolution of the scanner is known using the equation:

$$\boldsymbol{\alpha_k(j) = \rho j} \quad \textbf{(51)}$$

This notation is summarized in figure 80 where the laser scanner is at the origin of the local coordinate frame, two points (j and j+1) from scan q_k are shown with their ranges and angle.

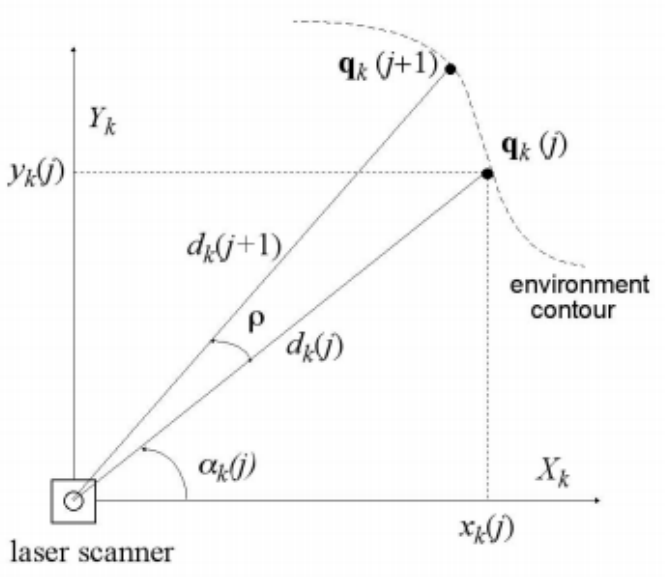


Figure 80: The coordinate system attached to the laser scanner [40].

Thus from figure 80, the coordinates of the polar point $q_K(j)$ in Cartesian system are:

$$\boldsymbol{x_k(j) = d_k(j)\cos\left(\propto_k (j)\right)} \quad \textbf{(52)}$$

$$\boldsymbol{y_k(j) = d_k(j)\sin\left(\propto_k (j)\right)} \quad \textbf{(53)}$$

When the robot moves with the scanner, two consecutive scans at time instants k and k+1 with coordinate reference systems are XY_k and XY_{k+1}will be recorded. The scans are called q_K, and q_{K+1} respectively. For scan matching, it is desired to find a coordinate transformation, T_k, composed of relative robot pose change (Δx, Δy, ΔØ) between the laser scanner frames at time instants k+1 and k.

Since the scan pose changes in opposite pose direction to robot pose change, T_k transforms from frame K+1 to frame k. Figure 81 shows laser scanner at time instant k which was transformed by an arbitrary pose at time instant k+1.

The readings from time instant k+1 can be projected back on the previous frame of time instant k i.e. XY_k using a coordinate transformation equation given by:

$$\begin{bmatrix} \hat{\boldsymbol{x}}_{\boldsymbol{k}} \\ \hat{\boldsymbol{y}}_{\boldsymbol{k}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\cos}\,(\Delta\emptyset) & -\boldsymbol{sin}(\Delta\emptyset) \\ \boldsymbol{sin}(\Delta\emptyset) & \boldsymbol{\cos}\,(\Delta\emptyset) \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{\boldsymbol{k+1}} \\ \boldsymbol{y}_{\boldsymbol{k+1}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\Delta x} \\ \boldsymbol{\Delta y} \end{bmatrix} \quad \textbf{(54)}$$

This equation allows the user to get the location of a point in the current scan at the previous scan if the transformation is known. Since the output from this equation is calculated based on the transformation, the output is labeled with hat symbol to indicate that it is a prediction. In a similar way angles and ranges of all scan points can be projected.

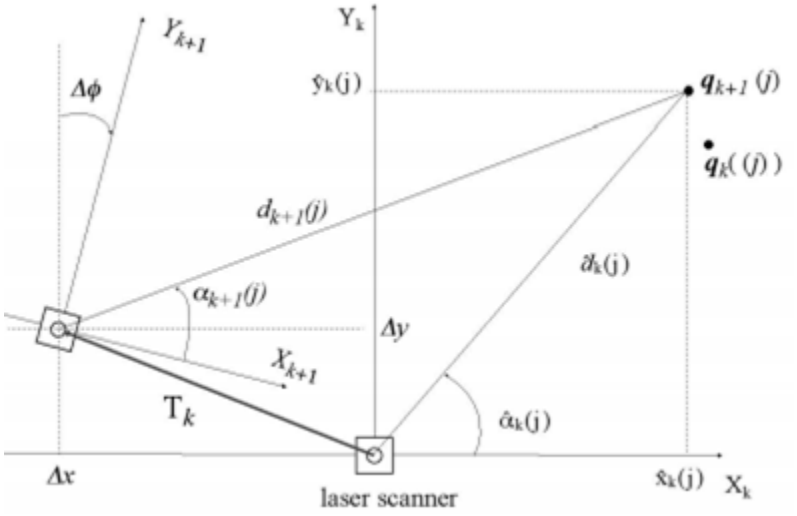


Figure 81: The geometry of the laser scanner at two time instances.

The predict parameters are compared with the actual measured parameters at time instant k and due to sensor noise, changes in the environment, and obstruction due to sensor movement some points will not match their prediction. Therefore, and error is defined as the difference between the position of the predicted point in Cartesian coordinates and the position of the actual point in Cartesian coordinates. This error will be worked out for every point. So the error for a point pair j is denoted by error(j). A large error indicates a potentially wrong match so a Boolean function p_T can be defined which produces true if the error exceeds a certain threshold

and false otherwise. This filters scan points to n_T valid pair matches and their ratio is the degree of overlap spanning 0 to 1.

This can be used to derive the equation that is iteratively minimized for finding the correct transformation:

$$\frac{\sum_{j=0}^{N} |\boldsymbol{p_T}(j)\boldsymbol{error}(j)|}{\boldsymbol{n_T P_{T_k}}} \quad \textbf{(55)}$$

The error threshold that is used to form the Boolean mask is critical for the ICP algorithm. Martinez et al [40] proposed starting with a maximum error threshold derived from odometry error and reducing the error threshold with each iteration to a minimum value. Feature based methods operate on the same principles but match feature properties instead of direct points.

4.4.2 Scan matching algorithm

The method employed for laser scan matching relies on extracted features, in this case line features. A line extraction algorithm was developed that produces an array of corner points where there is a significant change in gradient of the laser reading. The line extraction method is discussed in the next section (section i).

i. Line segmentation

The extraction of geometric features from data from the sensors of the robot describing an environment is of great use in SLAM since the use of geometric features requires less space and computation as well as enabling better recognition of similar data sets. Of those geometric features, a line feature is the simplest and most occurring one [55]. The line extraction algorithm is made up of two parts, the line segmentation and line identification parts.

Line segmentation

The method developed for segmentation relies on the observation that at line intersections, the gradient of the laser scan changes. Therefore, by differentiating the laser scan range with respect to angle it is possible to extract corners and hence beginnings and ends of line segments. The laser scan covers a range from 10 cm to more than 5 meters therefore to improve the outcome of the differentiation process, the log of the laser scan is differentiated. The finite difference method is used to differentiate the log of the laser scan twice to find the second derivative. This is then normalized by dividing by the maximum peak and the relative

spikes in the derivative are labeled as corners. An array called shod (static holder of derivatives) will be used to save peaks from second derivative that are above an absolute magnitude of 0.1. These points in the array are then tested to see if they are zero crossings or not. In order to mark them for further processing and to avoid the use of a low pass filter, the contents of the array neighboring the zero crossing point are set to zero while the point is set to 10. The flow chart in figure 82 is executed for detection of zero crossings associated with the corners.

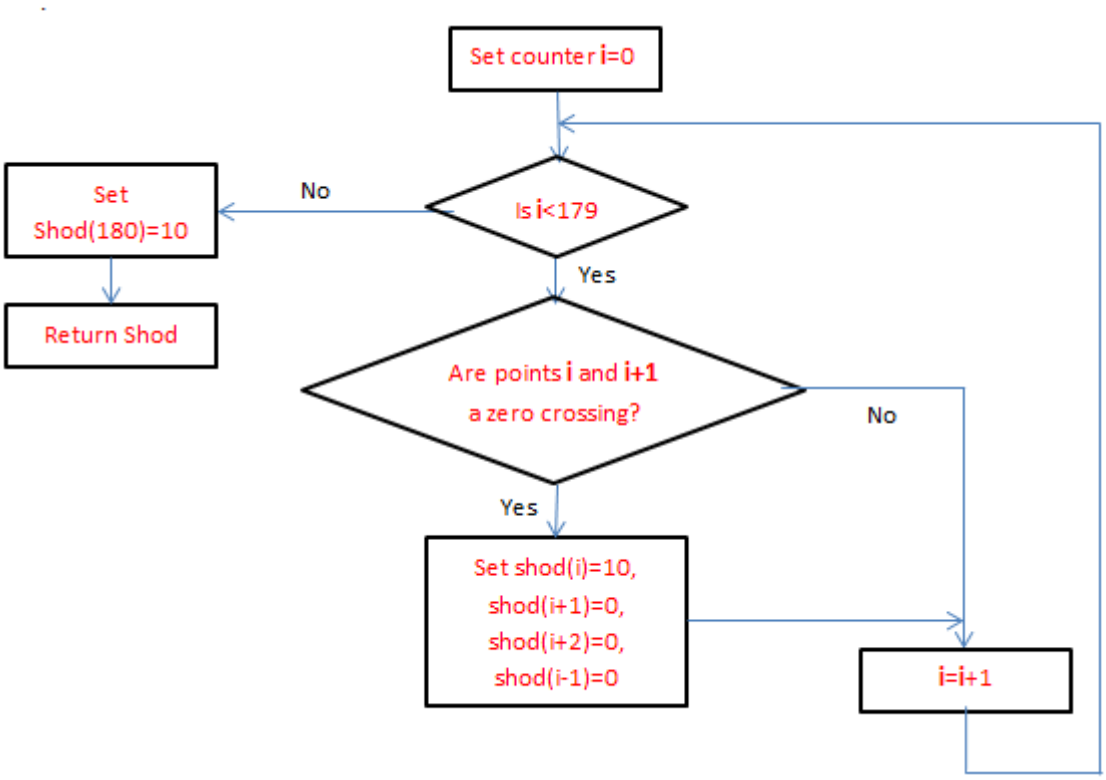
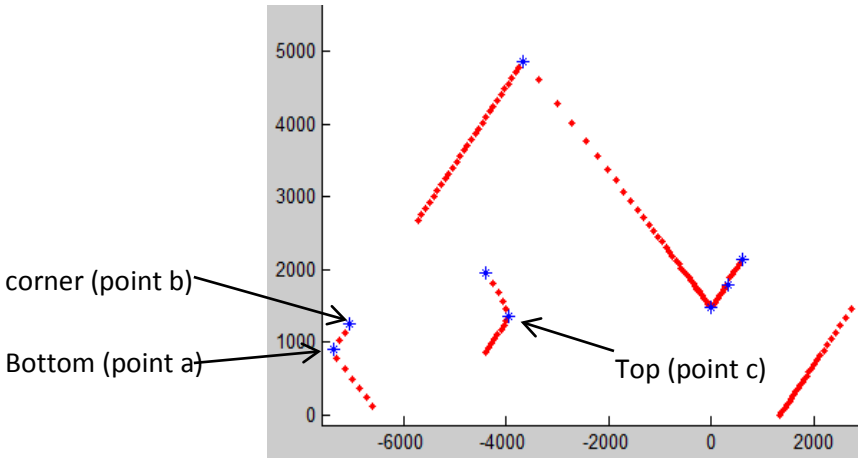


Figure 82: Extracting zero crossings for corners .

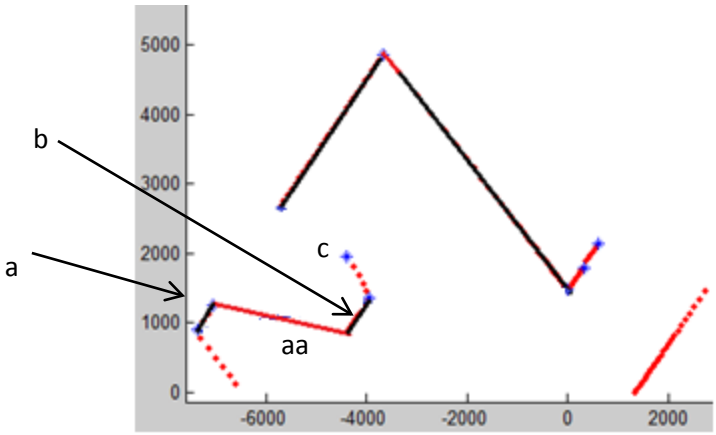
This produces a set of peaks that represent all possible edges/corners. However, as is the case with any differentiation approach, noise will appear as **false peaks**. Therefore, a method was used to filter the false peaks.

Dealing with noise

The approach to filter false peaks is based on two criteria that are determined heuristically. The first criterion is that a false peak separating two lines must not show a gradient difference (between the two lines) more than 10 degrees. The second criterion is shown in figure 83 and explained below.



(a)



(b)

Figure 83: the concept of the distance criterion used.

In figure 83, the edges are separated by blue asterisk (shown in 83(a)). When considering any two lines, the first point on the first line is called the bottom point (point a in figure 83a), while the last point on the second line is called the top point (point c in figure 83a), and the point separating the two lines is called the corner (point b in figure 83a).The line joining the corner point to the first point on the second line (shown in red in figure 83 (b)) is called *aa*, the first line and the last line are shown in black are called a, and b respectively. Fora peak to be a false corner:

$$\boldsymbol{Distance\ criterion} = \frac{\boldsymbol{aa+a+b}}{\boldsymbol{a+b}} > 1.05 \qquad \textbf{(56)}$$

This means a gap between lines must exceed 5% of their combined length.

The flow chart for application of this filter to remove false peaks is in figure 84. The corners produced by algorithm in figure 82 are used as inputs.

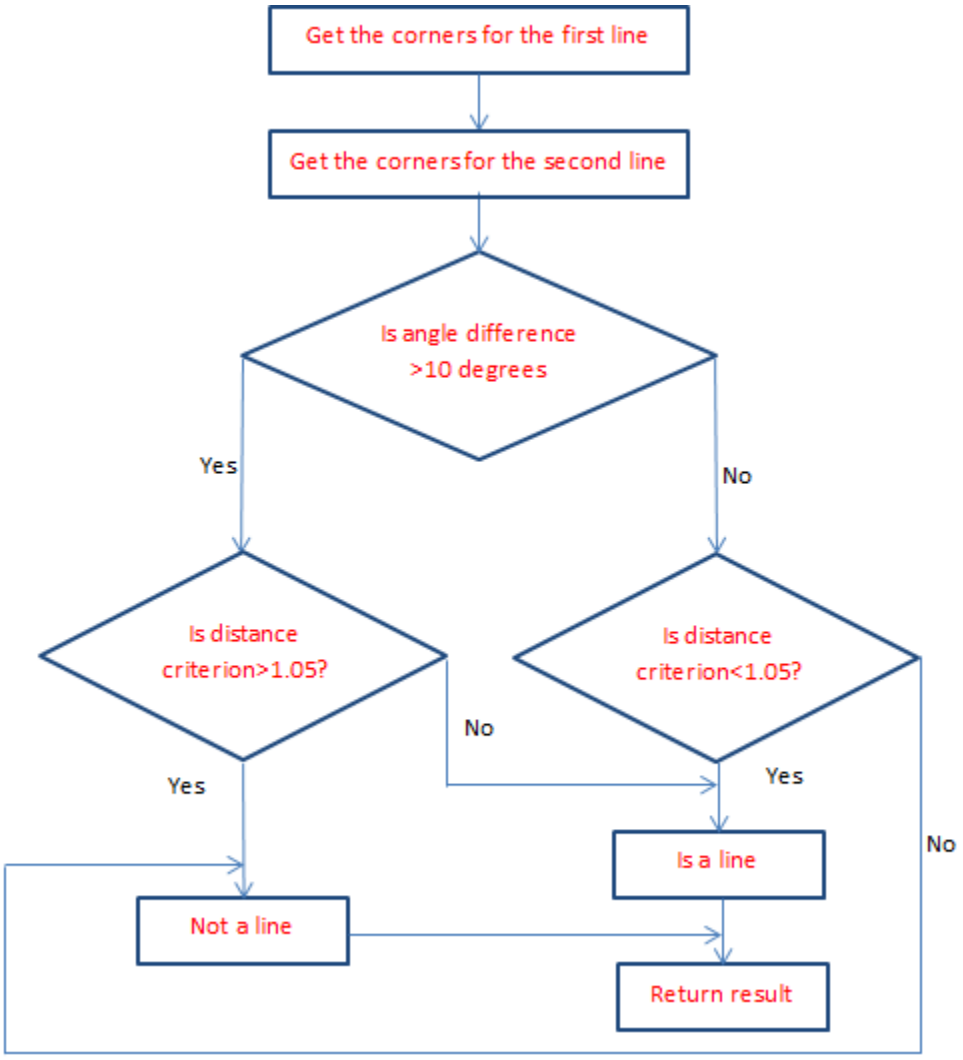


Figure 84: The flow chart for separating noises from actual lines.

The flow chart in figure 84 is repeated for every segmented line pair produced by the algorithm in figure 84 in shod array and the result is used to merge the lines if the peak is a true line (false zero crossing) or split the lines if the peak is a false line. The merging occurs by setting the appropriate value in shod array to zero and the splitting occurs by setting the split point to 10.

Results of Application of line segmentation algorithm

The segmentation algorithm developed was applied to the environment in figure 85.

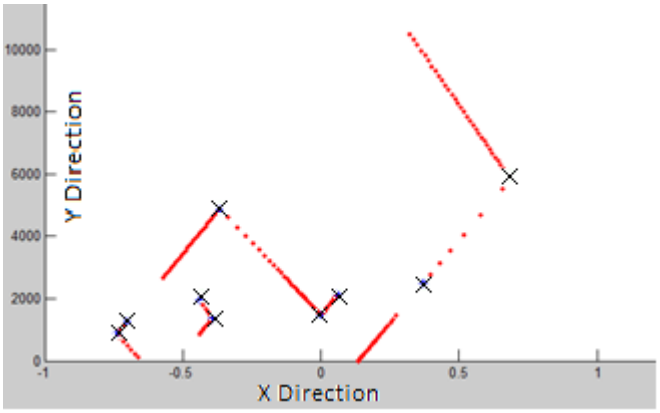


Figure 85: The environment considered for application of line segmenting function.

The crosses show the edge location as discovered by the algorithm. Figure 86 shows the actual peaks in shod array (before applying the algorithm in part ii) which include one false peak due to noise.

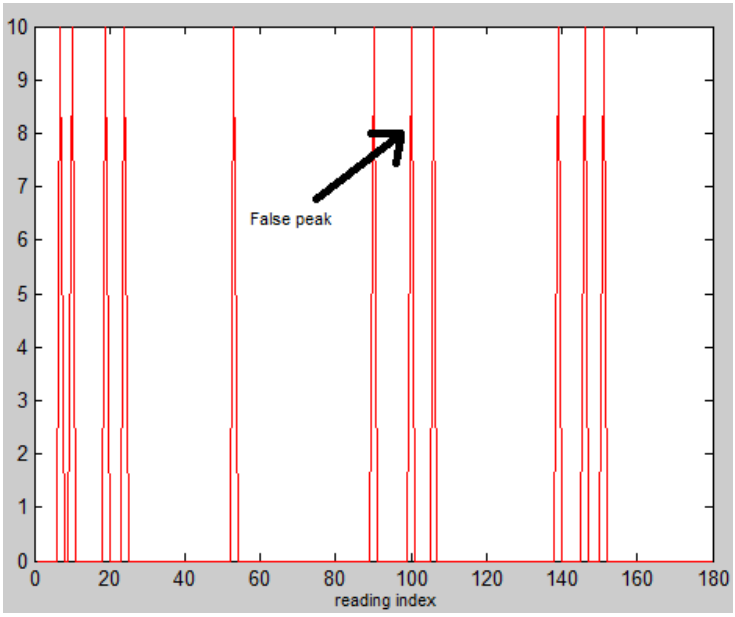


Figure 86: The peaks before filtering.

Now applying the algorithm removes the noisy peak with the results in figure 87. As can be seen, the false peak has been eliminated and its segments have been joined together into a single line.

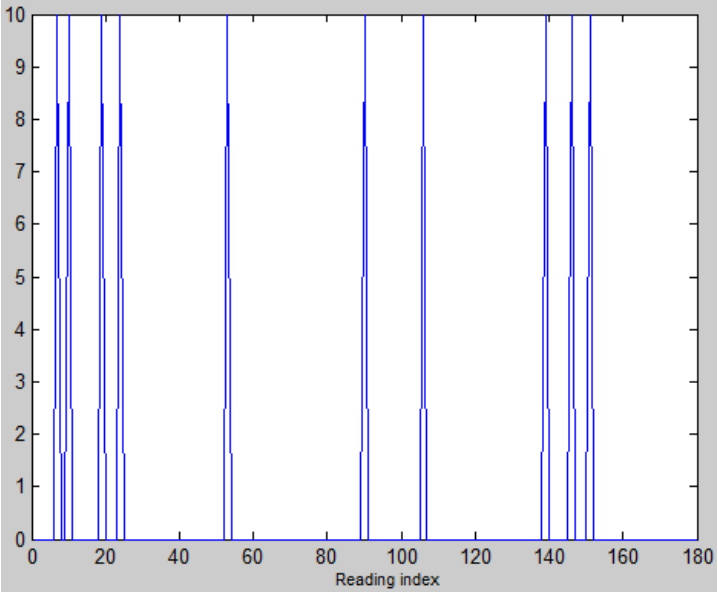


Figure 87: The noisy peak was successfully removed.

Different simulation environments featuring lines of different intersection angles were tried and the algorithm was successful in segmentation. Therefore, the next step was to make use of that segmentation result to identify the line segments via the two line parameters.

Discussion of line segmentation

From the results and analysis it was concluded that the use of a method based on differentiation to segment line can succeed and will likely be affected by noise. The noise error could be reduced or eliminated if an empirical criterion is used to separate lines from non-lines. Further, the differentiation approach is linear in time with increase in input size which corresponds to increased laser resolution. However, the algorithm is not fully immune to noise but the performance was found acceptable. The algorithm was found to be successful in eliminating most noise and segmenting the data for identification of lines parameters. The next step is to take the segmented laser sets to identify the lines by finding their gradient and intercept.

ii. Identification of lines

After the data was segmented, it is now possible to find the lines in the scan. To do this the total least squares method is used. Consider the following straight line to best fit *N* points:

$$\boldsymbol{Y = mX + c} \quad \textbf{(57)}$$

Where Y and X are vectors. The total least error can be found as:

$$\boldsymbol{error} = \frac{\boldsymbol{Y-mX-c}}{\sqrt{\boldsymbol{1+tan^2\theta}}} = \frac{\boldsymbol{Y-mX-c}}{\sqrt{\boldsymbol{1+m^2}}} \quad \textbf{(58)}$$

And he error squared becomes:

$$\boldsymbol{e^2} = \frac{\boldsymbol{(Y-mX-c)^2}}{\boldsymbol{1+m^2}} \quad \textbf{(59)}$$

Differentiating partially with respect to m and c and equating to zero yields two simultaneous equations:

$$\frac{\boldsymbol{\partial e^2}}{\boldsymbol{\partial m}} = \frac{\boldsymbol{(-2Y-2c+2mX)X}}{\boldsymbol{1+m^2}} - \frac{\boldsymbol{2m(Y-mX-c)^2}}{\boldsymbol{1+m^2}} \quad \textbf{(60)}$$

$$\frac{\boldsymbol{\partial e^2}}{\boldsymbol{\partial c}} = \frac{\boldsymbol{-2Y-2mX-2c}}{\boldsymbol{1+m^2}} \quad \textbf{(61)}$$

Equations 60 and 61 can be equated to zero and arranged to yield [81]:

$$\boldsymbol{0 = K_2 m^2 + K_1 m + K_2} \quad \boldsymbol{and} \quad \boldsymbol{c = \overline{Y} - m\overline{X}} \quad \textbf{(62)}$$

Where lower case letters are scalars, uppercase indicate sum of a vector with variables of this type. The bar symbol indicates the average of the sum of the quantities (X and Y), and the constants K_2, K_1, are given by:

$$\boldsymbol{K_2} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} (\boldsymbol{x_i - \overline{x}})(\boldsymbol{y_i - \overline{y}}) \quad \textbf{(63)}$$

$$\boldsymbol{K_1} = \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \left[(\boldsymbol{x_i - \overline{x}})^2 - (\boldsymbol{y_i - \overline{y}})^2\right] \quad \textbf{(64)}$$

The value obtained for m, the gradient, is then used to obtain the y intercept of the line joining those points by using equation 62 [81]. The total least squares method was used instead of the least squares method to enable correct identification of vertical lines which otherwise would be wrong. Further, it is more relevant to this data since both the x and y are not dependent on one another and variance exists in both quantities that stems from laser range variance.

Once the lines have been segmented and identified, the scan matching based on line features can be used as explained next.

iii. Feature based scan matching

Figure 88 shows image of a simulation laser scan. The line extraction algorithm was run and produced the corners shown by the cross points.

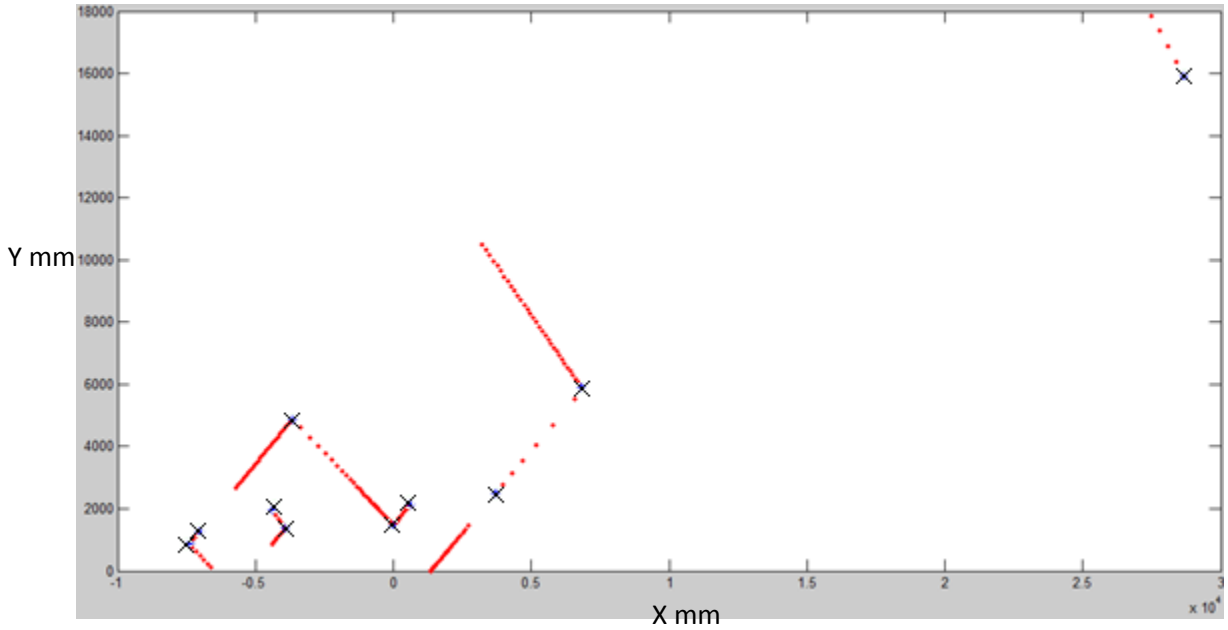


Figure 88: Line extraction using differentiation.

The laser sensor position in figure 88 is at the origin point (0, 0) and the positive y axis points in the same direction as the heading of the laser sensor. The corners representing ends and beginnings of the lines are used for scan matching. Scan matching requires two frames from laser scans with two arrays shod1 and shod2. The scan matching works by comparing lines of same lengths and positions in the two frames.

The algorithm has the following steps:

- First generate an array of relations (called rel) that has the line length for each line. This array is created for the two frames of lines from the two laser scans thus we have rel1 and rel2,
- The next step is to find the pairs of lines that have similar length. Due to sensor limitation a threshold of 0.05 m is allowed i.e. lines must be less than 0.05m different in length, and position
- The paired lines are then compared based on their gradients to find the rotation with respect to the initial frame,

- Once the rotation is found, the coordinate transform equation that relates two coordinate frames rotated by an angle is used to rotate frame one to frame two for finding the translation,
- This is repeated for every pair of lines, and
- The results are summed either as a weighted average by comparing these to odometry estimate.

The flow charts that describe the operation of the algorithm are shown in the following pages where the output of each flow chart serves as the input of the flow chart of the next stage.

The flowchart for the first part of scan matching in figure 89 finds the lengths of the lines.

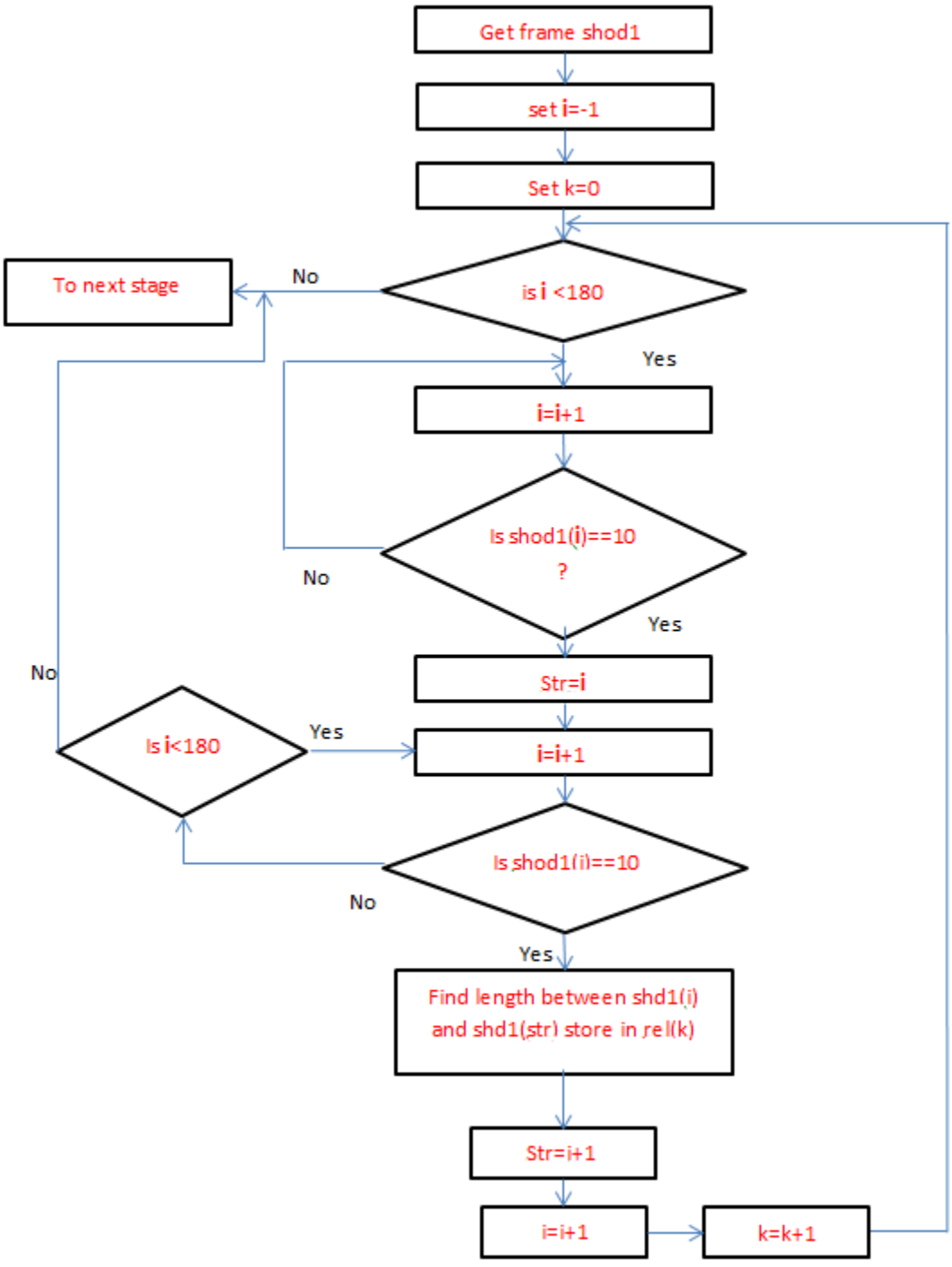


Figure 89: Flow chart for the first part of scan mathing algorithm.

The second part of the algorithm where the related lines are paired together this is in figure 90. The criteria used for pairing is the line length, and the location which is a property in the laser scan since it is a sequential scan.

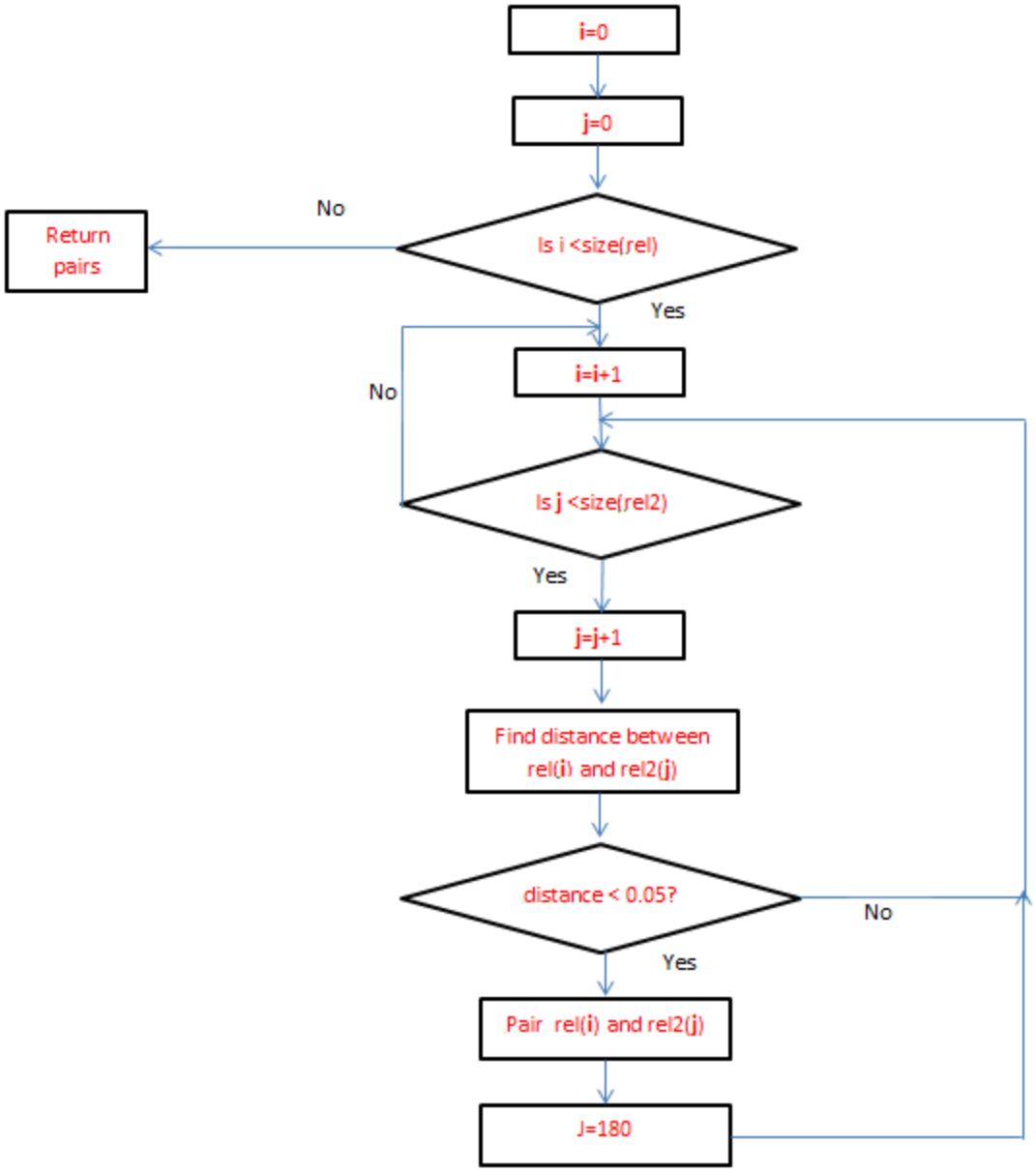


Figure 90: The second part of algorithm where data association (pairing of lines) is carried out.

The distance used in figure 90 to pair lines is the ratio of their lengths minus one i.e. lines should not show length variation more than 5%. This was empirically chosen based on experiments and simulation and is equivalent error threshold in the point to point scan matching. Following this, the rotation of the two frames is first found from gradient change as the gradients of two paired lines, m_1 and m_2 do not depend on the translation but only rotation. That is, any lines in the environment found at time instant k should have the same gradient as lines found at time instant k+1 in a different laser scan if the robot did not rotate. If the robot rotated, then the

observed change in angles of the lines in the local frames will depend on the angle of rotation regardless of the translation. Two methods were tried for finding the rotation. The first is by finding the average of all rotations:

$$\frac{\Delta\theta_1+\Delta\theta_2+\ldots..+\Delta\theta_n}{n} \qquad where \quad \Delta\theta_n = \mathbf{atan}(m_1) - \mathbf{atan}(m_2) \quad (65)$$

This method gives equal weight to all lines. However, in the real world some lines could be erroneous and some false rotations can appear due to different segmentation results. Therefore, it is better to have an individual weight for each estimate that is based on the predicted change due to odometry. This leads to the second method using the weighted average. The weight is related to how far from the odometric estimate is the rotation found. A good weight to use is a Gaussian weight since random errors can be modeled as Gaussians. The weight is given by:

$$w = e^{-\frac{(\Delta\theta-\Delta\theta_0)^2}{18}} \quad (66)$$

Where $\Delta\theta_O$ is the rotation change estimate from robot odometry, and the constant 18 is the width of the Gaussian function (standard deviation). All gradients are expressed as angles. A plot of the weight function is in figure 91 for an odometry estimate of 10 degrees. It has maximum weight at the expected odometry estimate and falls in a Gaussian manner as result deviates. The weight function can be empirically tuned to have different width if needed.

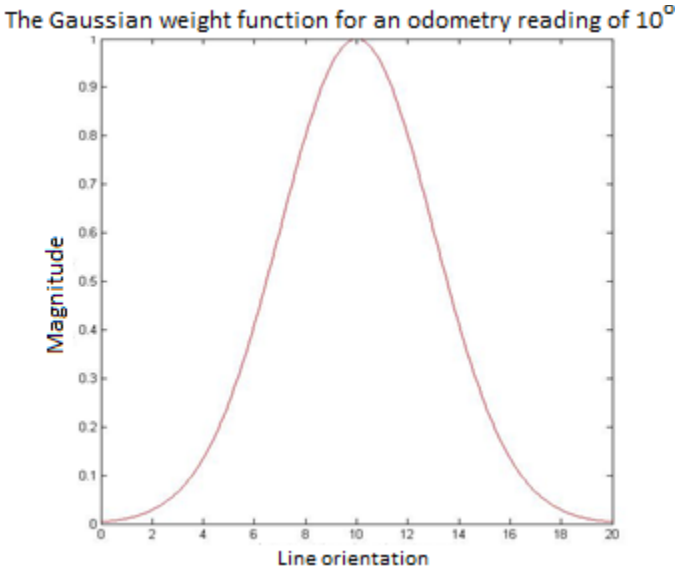


Figure 91:A plot for the weight function at different orientations.

Once the rotation concluded by the robot is found, the translation in X and y directions is needed. Two approaches were developed to complement one another. In the first one, the points from the laser scan before the robot movement and after the robot movement i.e. two frames are used for computing the translation. Using the rotation calculated by the algorithm, all the points of the previous frame are rotated to yield an expected estimate. The rotation is given by:

$$x_{expected} = x_2 \cos(\theta) - y_2 sin(\theta) \quad (67)$$

$$y_{expected} = x_2 \sin(\theta) + y_2 cos(\theta) \quad (68)$$

Where subscript 2 indicates the previous frame (relative to current frame). This is then compared to the actual scan at the new frame (x_1, and y_1) and a Gaussian weight function is used to rule out bad estimates. However, this method was found to yield wrong estimates in cases of inclined lines. The solution to this problem is to rotate the lines itself rather than the points. Equations 67, and 68 are used for this but applied to the line parameters (gradient and intercept) instead of the line points to predict the modified line parameters due to pose change.

In the ideal condition this should match the observed gradient and y intercept in case of no robot motion. In case of robot translation, the observed gradient will not change but the observed y intercept will change as demonstrated in figure 92. Here, the same line is observed from frame X_kY_k and $X_{k+1}Y_{k+1}$ after it was moved an arbitrary distance, dx, in the x direction. The y intercept changed as a result of this motion. Using the equation for a straight line:

$$y_1 - y_0 = mx_1 - mx_0 \quad (69)$$

$$y_1 - y_0 = m(x_1 - x_0) \quad \rightarrow \quad \Delta y = m\Delta x \quad (70)$$

Thus the change in y intercept is related by the gradient to the change in x direction. Additionally, the y intercept changes directly with motion in the y direction thus the overall change in y intercept is given by:

$$\Delta c = m\Delta x - \Delta y \quad (71)$$

Where Δx is the robot motion in x direction, and Δy is the robot motion in y direction.

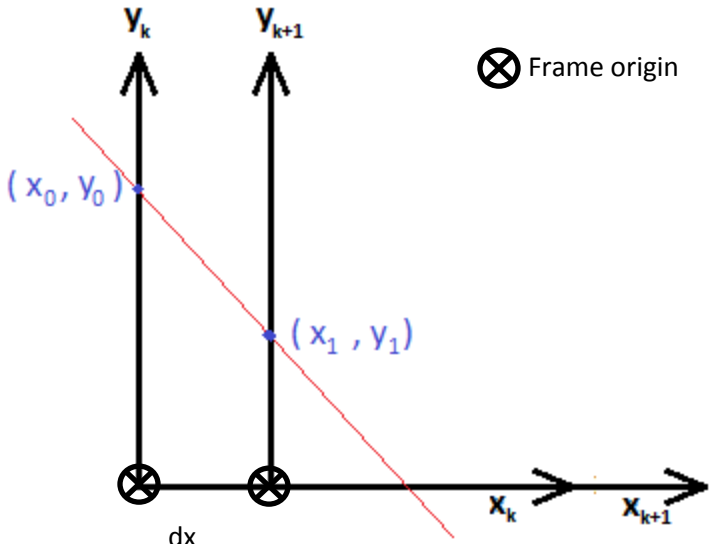


Figure 92: The change in y intercept due to motion in x direction.

Equation 71 has two variables (Δx, and Δy) that affect the observed change in y intercept. To solve this equation at least two lines are needed. The equation can be solved usingleast squares method (effectively finding Δx and Δy of best fit).

In the implementation a weighted least squares method was used where the weight for the lines was derived from equation 66 and normalized. Further, if the number of lines is low (2 or less) or the difference between worked out (Δx, Δy) and actual odometry is large i.e. error greater than error due to method 1 then method 1 is used. The weighted least squares method has the general form given by [56]:

$$\boldsymbol{SSE} = \sum_{i=1}^{n} \boldsymbol{w_i}\left[\boldsymbol{y_i} - \left(\widehat{\boldsymbol{\beta}}_{\boldsymbol{o}} + \widehat{\boldsymbol{\beta}}_{\boldsymbol{1}}\boldsymbol{x_{i,1}} + \widehat{\boldsymbol{\beta}}_{\boldsymbol{2}}\boldsymbol{x_{i,2}} + \widehat{\boldsymbol{\beta}}_{\boldsymbol{k}}\boldsymbol{x_{i,k}}\right)\right]^2 \quad \textbf{(72)}$$

Where SSE is the sum of squared errors, w_i is the *i*th weight, y is the actual value of function, $\hat{\beta}_k$ is the estimation of the *k*th parameter, and $x_{i,k}$ is the value of the *k*th variable of point *i*. In the case of straight lines, there are only two parameters (gradient and y intercept) that can minimize the equation.

iv. Results from scan matching algorithm

The algorithm was run on the simulation environment shown in figure 93. As can be seen, the line extraction algorithm has segmented the lines (crosses indicate beginnings and ends of lines).

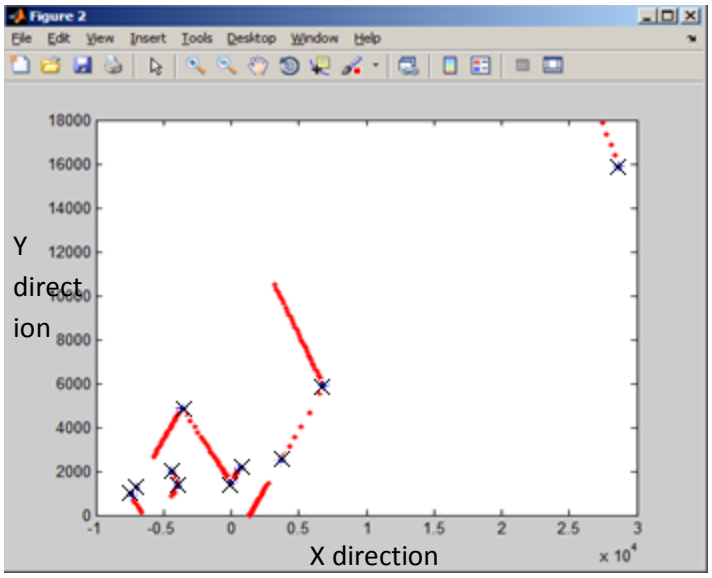


Figure 93: The simulation environment.

The data was rotated 10 degrees anticlockwise. The original environment (blue) and rotated (green) are shown in figure 94.

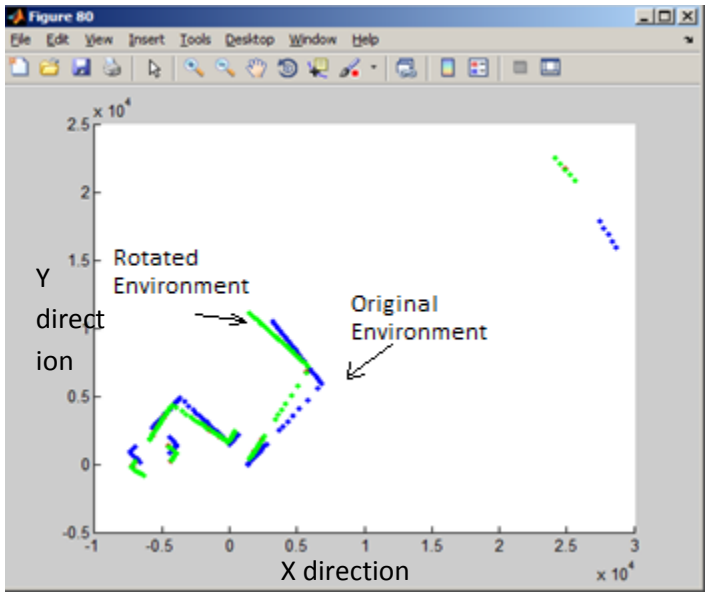


Figure 94: The original environment (blue) and the rotated environment(green).

The results were matching to the expected rotation and translations where the algorithm yielded an output of 9.8 degrees rotation, 10mm translation in x, and -21mm translation in y.

This compares with the actual transformation of 10 degrees and zero translation in both x and y. Following this, the algorithm was applied in a lab environment to test its performance.

To conduct this, actual readings were taken in a lab environment using a SICK LMS200 laser on a P3DX robot. The robot was rotated 5 degrees and translated 100mm while readings were taken. Figure 95 shows the lab environment. The line extraction algorithm succeeded in segmenting long line segments correctly but there are some inaccurate segments. The red dots represent laser readings, and the asterisk represent the line segment boundaries discovered by the algorithm.

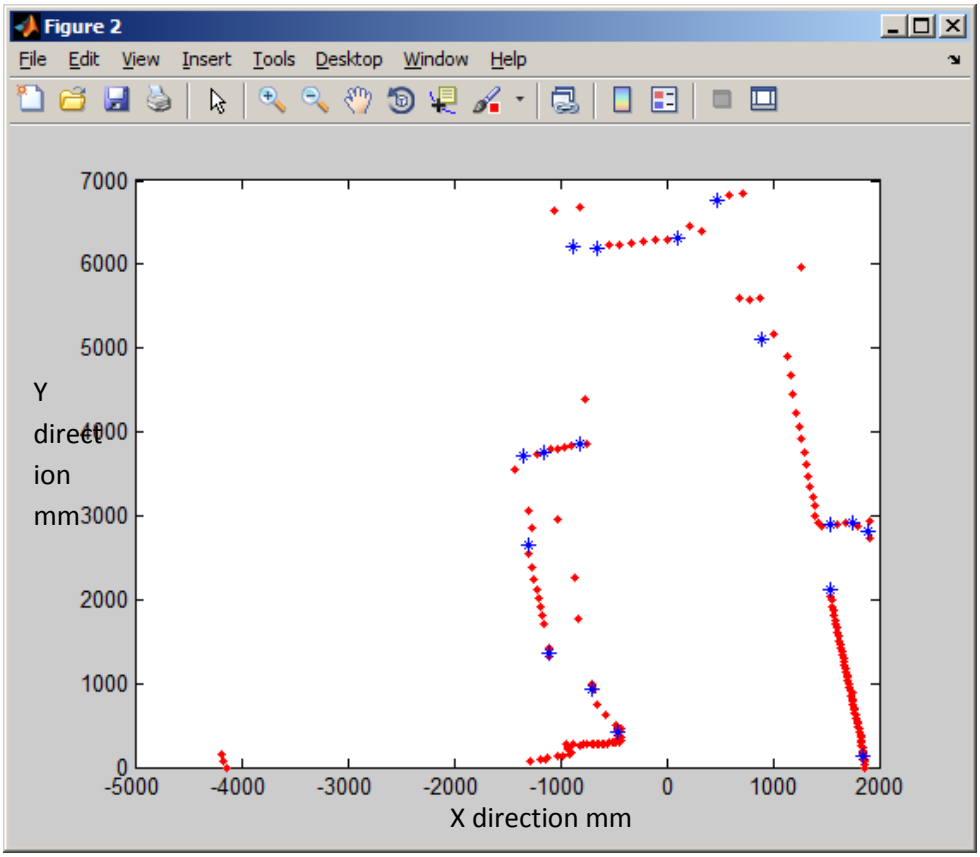


Figure 95: The lab environment where the actual readings were taken.

In the above image, the robot was then rotated and the new scan read again and stored for processing. The robot was then moved forward by a distance of 100mm i.e. 0.1m to test the effect of translation. The resulting scan is shown in figure 96 on the next page. The algorithm was run on the data from combined translation and rotation.

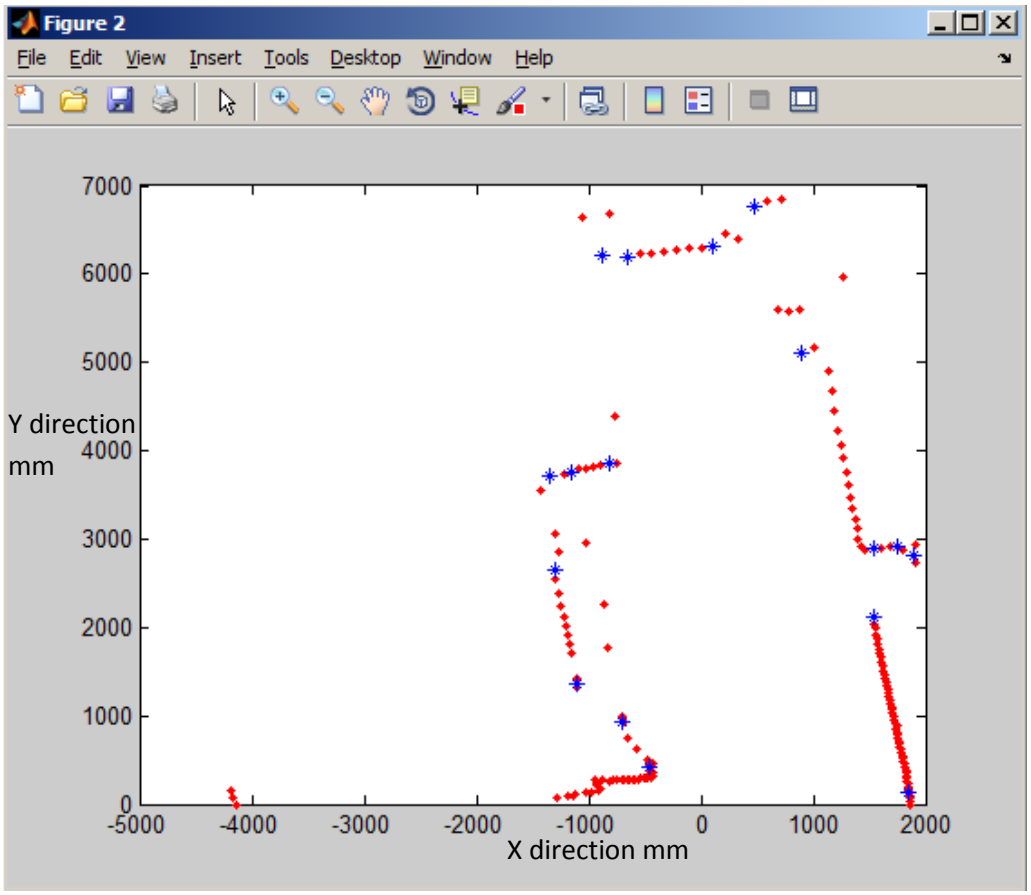


Figure 96: The scan of the same environment after the robot has moved 100mm forward.

For the translation only run, the algorithm was run and the following results were obtained.

rott = 0.1815

dx = -100.7236

dy = 1.8426

As can be seen the results are close to those expected from odometry. Therefore scan matching can be used to correct the robot motion path. Another trial was also made where the robot rotated instead of pure translation and was moved in y direction (manually an estimated 100mm).The results are shown in figure 97. The standard estimate of error (SEE) for the detected lines is shown too.

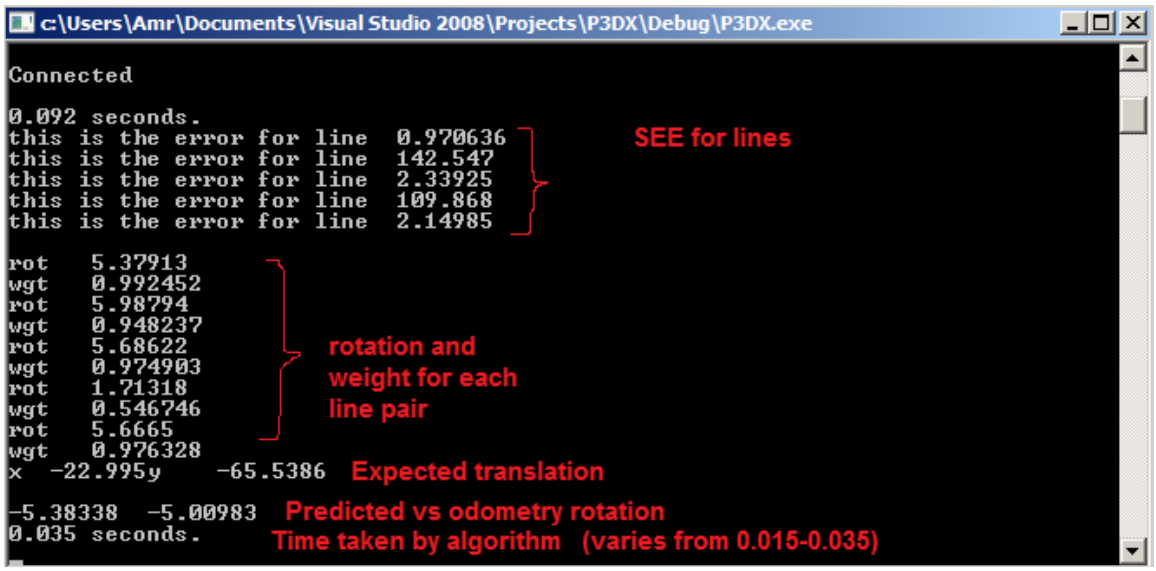


Figure 97: The results of scan matching algorithm compared to odometry.

The expected translation should be zero in x and around -100mm in y but given the presence of noise and obstruction of lines a small error appears of the order of few cm (2 and 4 in x and y respectively). Notice further how the function is not heavily biased to odometry estimate in that it produced an observed change in y despite the robot odometry having 0 y change.

The function was tested in different scenarios and the time taken varies from 0.015 to 0.035 seconds making it suitable for real time algorithm. Given the fact that the laser sensor operates at 10 to 15 hz i.e. ~ 0.1 seconds then the algorithm can process all laser scans. However, the accuracy is badly affected in corridors and heavily cluttered environments with many chair legs and dynamic objects or if line obstruction is likely to occur. Corridors are very challenging since only two different lines do not always give any margin for errors.

4.4.3 Comparison with other line extraction methods

The developed scan matching method will be compared to some of the commonly used methods. Those methods will be discussed first. Any line extraction algorithm has to address three main issues; to find how many lines are in a 2D laser scan, which points belong to which line, and given the classification of the points to different lines how to estimate the line parameters [55]. There are many different line extraction algorithms that deal with those three problems and are discussed below. In the next sections (D, E) they are compared to the developed algorithm.

i. Split and merge

Split and merge algorithm works by iteratively fitting a line to a set of points then finding the point with largest distance from the fitted line above a predefined threshold. The set is divided into two new sets at that point and the process is repeated. The pseudo code clarifies this in the following steps [55]:

1. Initial set of all points, S_i, has N points where i initially is 1
2. Fit a line to set S_i and save it in array L
3. Find the point, P, with maximum deviation from the line fitted in point 2
4. If distance of point P from the line is less than a threshold go to point 2 else continue
5. Split set S_i at point P to get sets S_{i1} and S_{i2}
6. Go to point 2 Until point 3 can no longer be applied
7. Merge collinear sets in L

When fitting a line, the least squares method is used. Another variation of the algorithm joins the first and last points in a set to fit the line in which case it is called the iterative end point fit algorithm [55]. The performance of the split and merge algorithm tends to be sensitive to the threshold chosen in step 4 [57]. The iterative end point algorithm on the other hand is less sensitive to the threshold since it recursively splits a set into subsets for lines [57].

ii. Line regression

Another method used for line extraction is the line regression method based on Hough transform. The method utilizes a sliding window with a size which is dependent on the environment [55].The pseudo code is given:

1. Initialize the sliding window, N_f, of predefined size
2. Fit a line for every N_f points
3. Compute fidelity array as the sum of distances between every three adjacent windows
4. Scan the array in point 3 for consecutive elements with a value less than predetermined threshold
5. Merge overlapping line segments and recomputed parameters of the lines

iii. RANSAC algorithm

The Random Sample Consensus algorithm (RANSAC) is an algorithm for fitting a model to a set of data. The algorithm is good with outliers and can be used with different features other than lines if the model for those features is known [55]. The pseudo code for use of RANSAC in line extraction is given:

1. Choose a sample of 2 points randomly from the set N
2. Fit a line through the points using least squares method
3. Compute the distances of other points in set N to the line computed in 3
4. Gather points close enough to belong to line in step 2 to be the inlier set
5. Remove inliers from the set N
6. Repeat step 1 until few points remain or iteration limit is reached

For point two any fitting method could be used but normally least squares is used. Total squares can be used if both x and y vary.

iv. EM algorithm

The expectation maximization algorithm (EM) in this context is a probabilistic tool used for the extraction of line segments in a 2D scan using a probabilistic approach [55]. The pseudo code is given by:

1. For a set, N, generate random line parameters
2. Initialize weight for points in set
3. Compute the weights of points from the expected line parameters obtained in step 1
4. Recomputed the line parameters based on weighted points
5. Repeat steps 3 and 4 until convergence or maximum number of steps reached
6. Repeat steps 2 to 5 until maximum number of trials is reached or a line is found
7. Remove points belonging to the line found in in step 6 from set N
8. Repeat steps 1 to 7 until no more points remain in set N or no more lines can be found.

The EM algorithm has several drawbacks such as it could get trapped in a local minimum while searching, and the initialization of random values is difficult to ensure good starting values as well as not being deterministic [55].

Experiments were conducted by [55] in an indoors environment to find the speed and accuracy of various line extraction algorithms. For RANSAC algorithm 1000 iterations were used and a 671 by 401 array was used for the Hough transform yielding a line parameters resolution of 1 cm. for the EM algorithm 50 trails were used and 200 iterations were allowed before convergence. According to the results, EM performed worse speed wise at a speed of 0.6 Hz, Hough transform at 8 Hz then RANSAC at 30 Hz followed by split and merge which was the fastest at over 1000 Hz. However, RANSAC and Hough transform displayed the highest accuracy in terms of the line parameters. Moreover, the EM algorithm had the highest percentage of false positive lines.

From this data, it was decided to use the split and merge algorithm and RANSAC algorithm as candidates for comparison with the developed algorithm.

For comparison purposes the iterative end point algorithm was used with a distance threshold of 100 mm and RANSAC algorithm was used with the condition that a point is considered an inlier if it is within 50 mm of the proposed line. Proposed lines with five or more inlier points were chosen for fitting else they were rejected. The number of iterations was varied for different scenarios to see the effect on performance and time. The results of simulation are shown in section 4.4.4.

4.4.4 Comparing Line segmentation algorithms

i. Results with RANSAC algorithm

The RANSAC algorithm was used with the environment in figure 98.

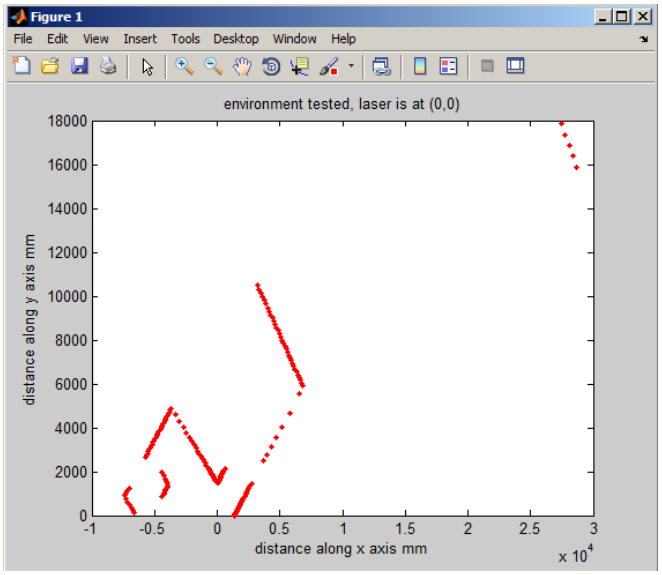


Figure 98: First simulation environment for RANSAC.

RANSAC was run first at 10 iterations and the results were processed to remove repeating line segments. The lines found by the algorithm before and after the processing are shown in figure 99 as blue lines superimposed on laser scan points. The time taken was 0.02 seconds (20 ms) but not all lines were found.

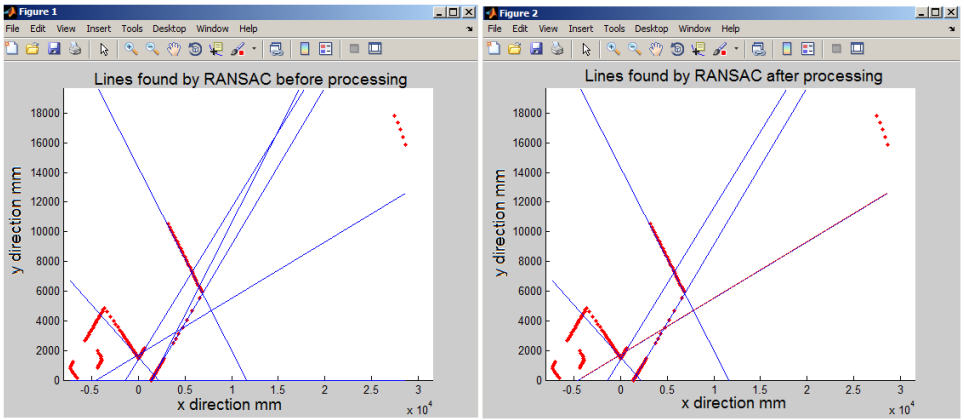


Figure 99: The lines found by RANSAC before filtering (left) and after (right) at 10 iterations.

Of the total 10 lines 5 were found and one false positive (shown in red). The number of iterations was increased to 50 and this took a time of 0.06s (60 ms) to execute. The result at 50 iterations is shown in figure 100.

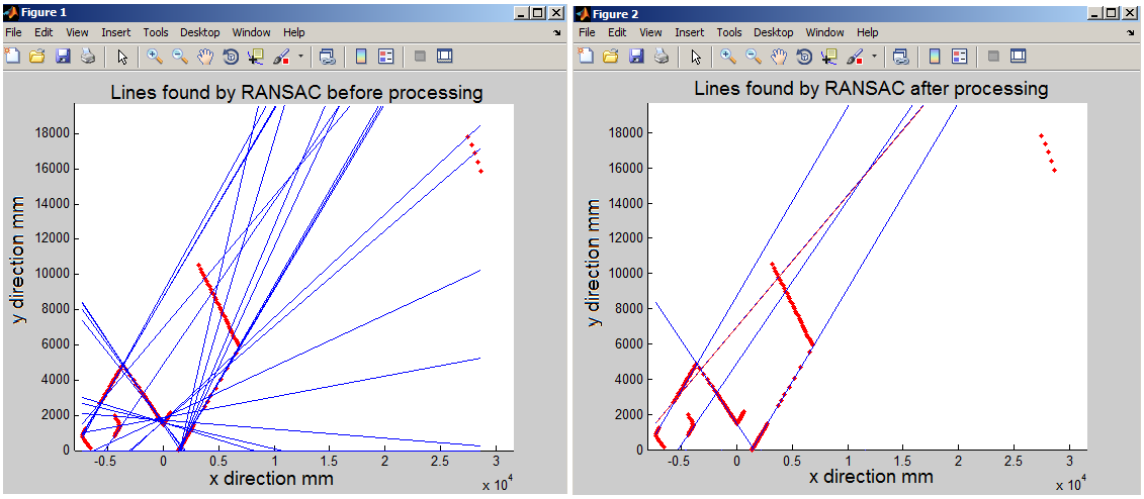


Figure 100: The lines found by RANSAC before filtering (left) and after (right) at 50 iterations.

Of the total 10 lines 6 were found and one false positive (shown in red). Therefore the number of iterations was increased to 100 and this took a time of 0.15s (150 ms) to execute. The result indicated that out of the total 10 lines 8 were found and one false positive. It was found that since RANSAC is not deterministic the results varied from iteration to another but at 100 iterations the best obtained was 9 lines out of 10 and one false positive. The repeating lines were omitted based on the number points, similar gradient and y intercepts, and low line quality given by the number of points i.e. lines with higher number of points were favored. Further increases in iterations above 100 did not improve results but significantly increased algorithm time.

Another simulation environment was tried with two lines forming a corner. The environment is shown together with the results of EANSAC algorithm at 10 iterations in figure 101. The algorithm took 0.03 seconds (30 ms).

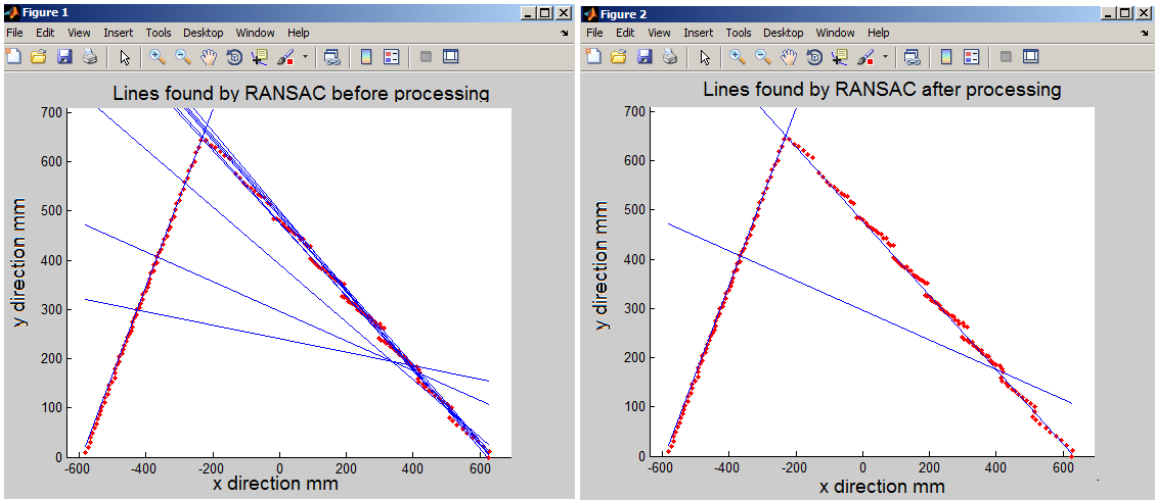


Figure 101: The results from RANSAC algorithm before processing to omit repeated lines (left) and after (right).

All lines (2) were successfully found with one false positive. The algorithm was tired at 50 iterations and the time taken was 50ms and more iterations did not yield different results. Therefore, another simulation environment was tested as shown in figure 102. The results were after 10 iterations and took about 0.03 (30ms) with the RANSAC algorithm.

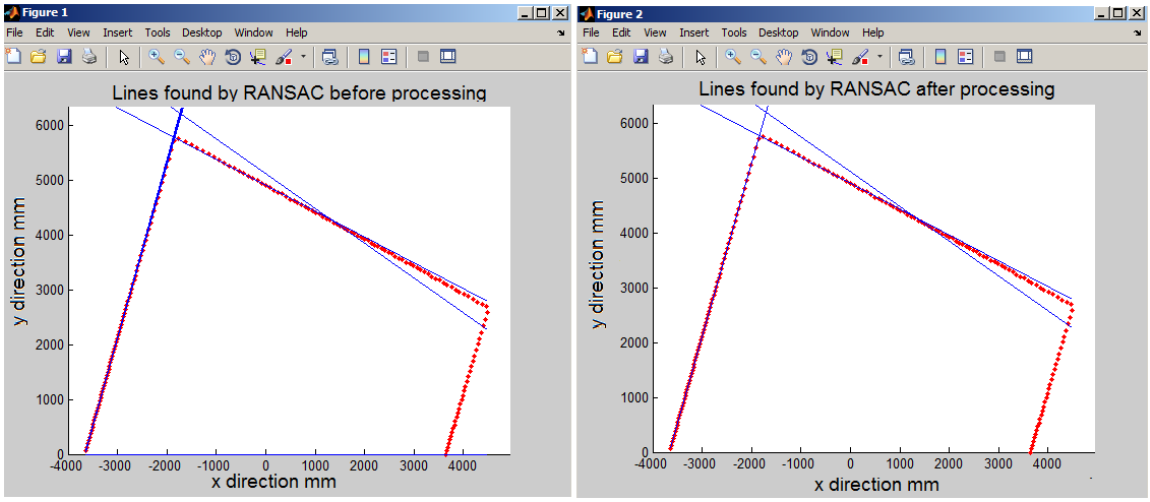


Figure 102: The lines of RANSAC algorithm applied to the environment shown at 10 iterations.

The algorithm took 0.02 seconds (20ms). However, only two lines out of 3 were detected and one false positive. The algorithm was tried several times with no noticeable improvements. Iterations were increased to 50 but not a change. At 100 iterations, the algorithm successfully finds the three lines taking 0.12s (120ms) to do so.

ii. Results with IEP algorithm

The iterative end point algorithm (IEP) from the split merge group of algorithms was applied to the same environments to compare the results. The results for the first environment are shown in figure 103.

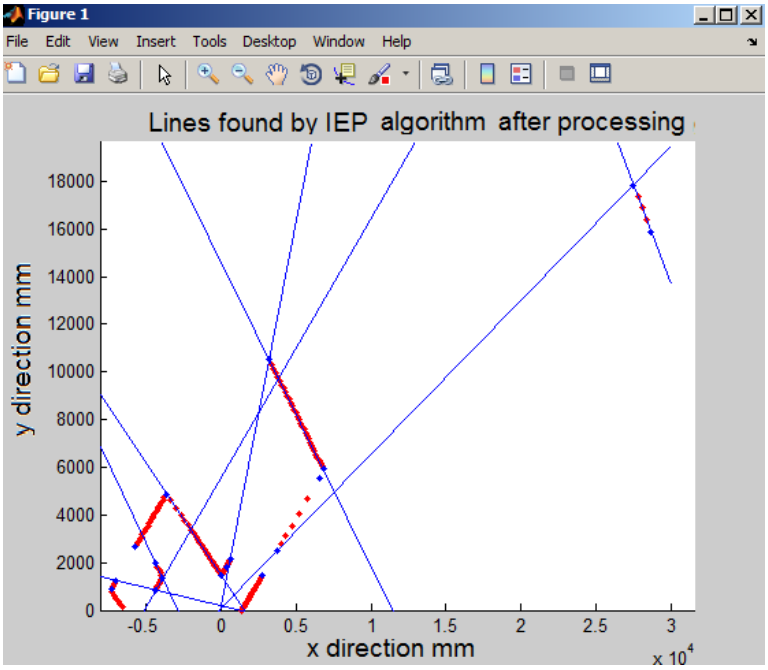


Figure 103: THe IEP algorithm applied to the first simulation environment.

The time taken by the algorithm was 0.1 seconds (100 ms) and all the lines were found correctly with one false positive. The endpoints and starting points of the lines are shown in blue dots. The lines drawn in blue do not reflect the actual lines due to step gaps between lines mistakenly drawn as a line. This could be avoided by a simple test such as that the line has more than two points.

In the results for the second simulation environment (same as figure 101) only two lines were successfully detected. Despite the noisy lines on the right, the algorithm successfully identified the segments as one line. No false positives were found and the time taken was 0.04 seconds (40 ms). The IEP algorithm was applied to the last simulation environment and yielded no false positive with all lines correctly identified and the algorithm took 0.06 seconds (60 ms) to conclude this process.

iii. Results with Developed algorithm

Finally the line extraction algorithm that was developed by this thesis was tested with the same three simulation environments. The results of applying this algorithm to the first environment are shown in figure 104. The algorithm only segments lines and thus the start and end points of lines are shown as crosses.

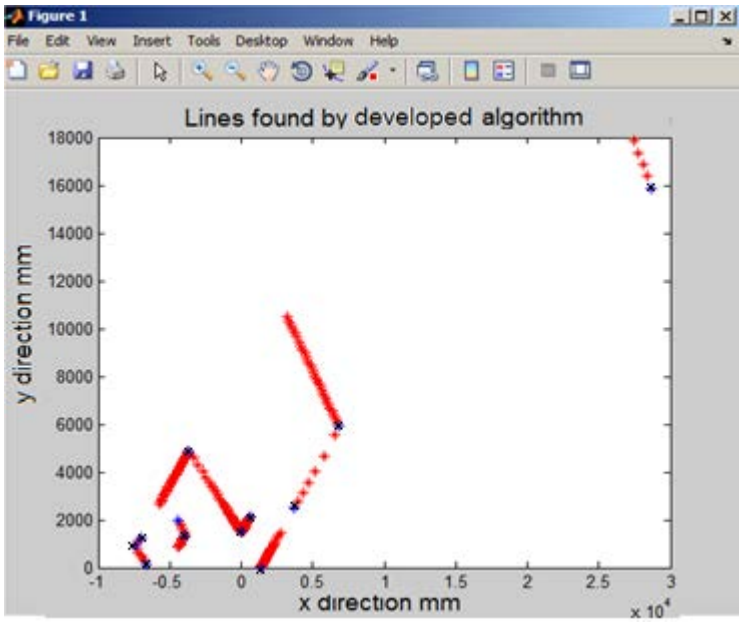


Figure 104: Results of applying the developed algorithm to the first simulation environment.

The algorithm took between 0.008s and 0.01s (8ms to 10ms) and successfully found all the lines with one false positive. The lines were not plotted to show clearly the beginnings and ends of the lines (corners) that were used as input to total least squares. Clearly all lines are segmented from one another by the algorithm. The algorithm was then tested with the second simulation environment. The results are shown in figure 105. The algorithm took about 0.01s (10ms) to conclude its operation. All lines were found but one line (on the right) was mistakenly (due to noise) split as two different lines. Thus it can be considered as one false positive.

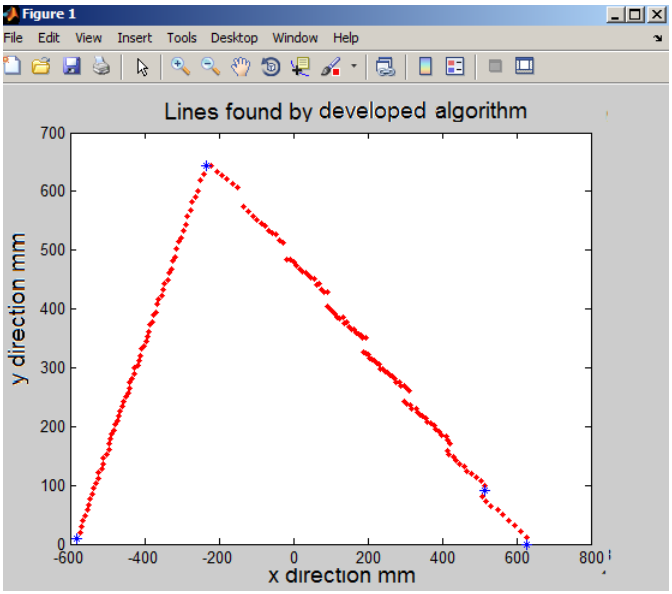


Figure 105:Results of applying the developed algorithm to the second simulation environment.

Finally, the third simulation environment was tested by the developed algorithm and the results are in figure 106. The time taken was about 0.0094s (~10ms). No false positive was found and three lines were correctly segmented as shown by the position of corner asterisk.

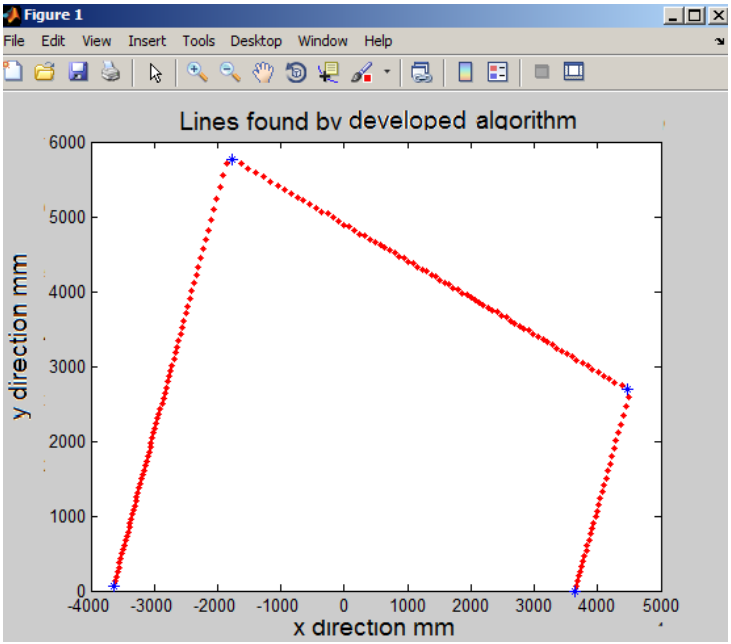


Figure 106:Results of applying the developed algorithm to the third simulation environment.

iv. Discussion

A summary of the comparison between the different algorithms for the three simulation environment tested is shown in table 3. For RANSAC the number of iterations needed is shown in brackets.

Table 3: A summary of the comparison between the different algorithms developed.

Environment	Environment 1			Environment 2			Environment 3		
Algorithm	Lines found	False positives	Time taken	Lines found	False positives	Time taken	Lines found	False positives	Time taken
IEP	10/10	1	10	2/2	0	40	3/3	0	60
RANSAC	9/10	1	150 (100)	2/2	1	30 (50)	3/3	0	120 (100)
Developed algorithm	10/10	1	10	2/2	1	10	3/3	0	10

It must be noted that RANSAC is not deterministic and results vary from run to run even with the same inputs and variables and this made the comparison difficult. However, extensive time was consumed testing to ensure the best results from RANSAC were obtained. Moreover, notice that the time taken by the algorithms should only be used as a rough guideline and is not meant to be an exact reflection of the algorithm complexity. This is because the time was obtained from MATLAB tic toc functions on a laptop computer with various background processes taking place. However, care was taken to test all the algorithms in the same session to rule out the effect of other processes in affecting the processor time due to overloading or different number of background threads etc.

Although the developed algorithm performed well, the scan matching relying on it will be affected by environments having low number of features i.e. lines such as corridors for example.

4.4.5 Comparing scan matching algorithms to the developed algorithm

To test the various algorithms for scan matching, a simple simulation environment was made using line gradients and y intercepts. The environment was then rotated by any desired amount and translated by any desired transformation to yield the transformed environment. This is done by predicting the gradient and y intercept of the current lines at the given transformation.

A simple simulation environment was created and rotated by different amounts with one version that was of pure translation in y direction. The first case was a rotation of 5 degrees as in figure 107 (not to scale) where the laser sensor is at (0,0) and the old environment is in black and the rotated result is in red. The points on the edges were not included since they will move out of field of view.

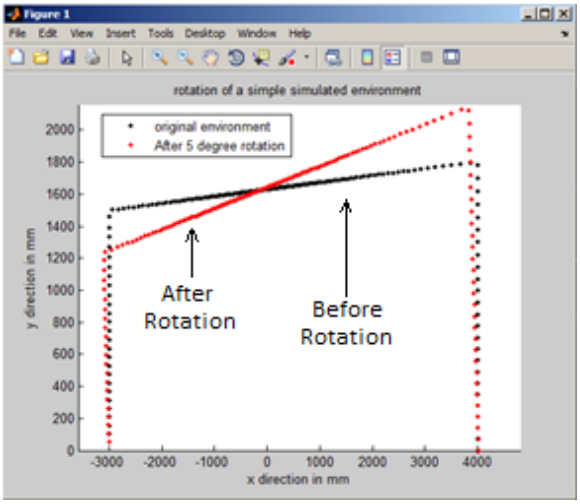


Figure 107: The rotation of a simple simulated environment by 5 degrees.

The second case was a rotation by 1 degree. This is shown in figure 108.

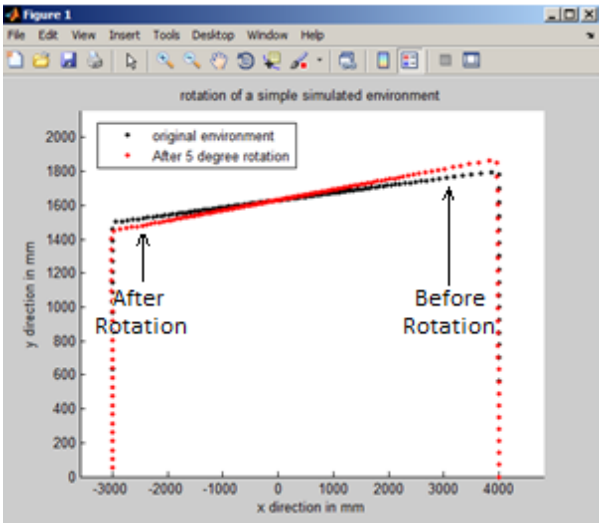


Figure 108: The rotation of a simple simulated environment by 1 degree.

A third case was the translation in y direction by 200mm. This is shown in figure 109.

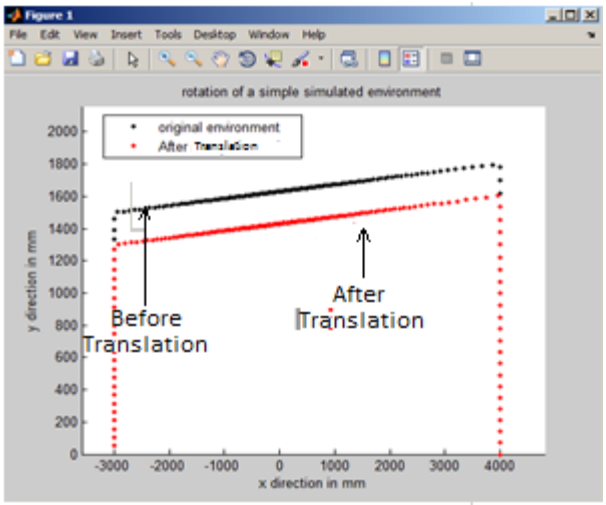


Figure 109: The translation of a simple simulated environment by 200mm in y direction.

Those three environments generated at different pose change were used to simulate the various scan matching algorithms.

i. Results with Developed method

The developed algorithm was run using two sets rotated by 1 degree apart and the algorithm succeeded in extracting lines although in set 2 two points are mistakenly marked as a different line. The following transformation was found:

(dx, dy, dth) =(3.195mm, 0.905mm, 1.2401 deg)

The error in displacements in x and y is only few mm and rotation was found as 1.24 degree the error is due to the fact that two lines were at almost 90 degrees which makes estimating the small change in rotational displacement difficult. A second set was tried at 4 degree rotation. The lines were successfully identified as shown in figure 110.

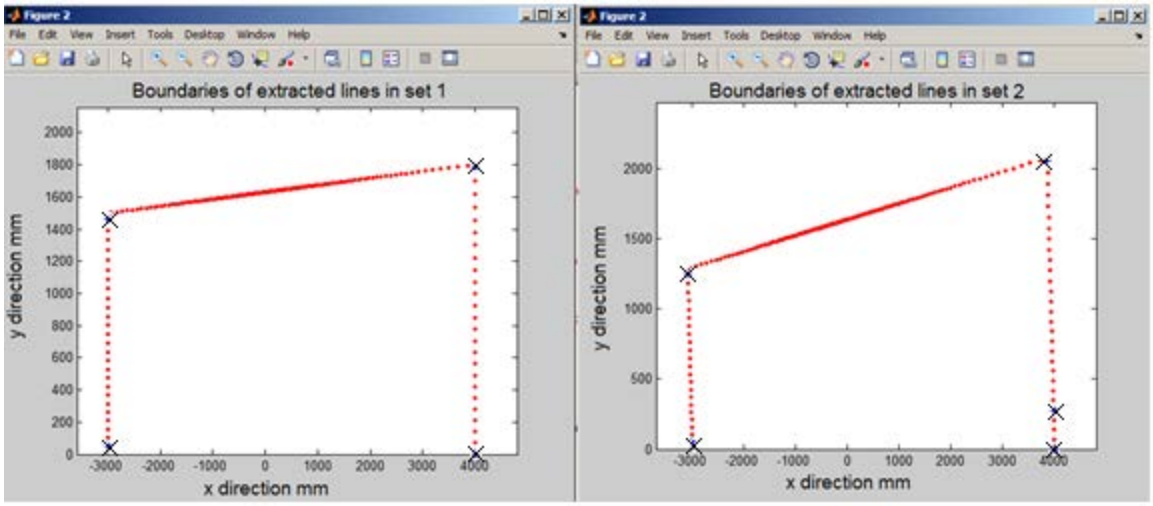


Figure 110: The result of line extraction algorithm on the two sets prior to scan matching.

The output found was : (dx, dy, dth) =(0.1565mm, 7.13mm, 4 deg)

This correctly gets the transformation between the two frames. The final trial was at 200mm translation in y direction. The following output was obtained:

(dx, dy, dth) =(-0.0123mm, -198mm, -0.0123 deg)

Note the negative sign which occurs as the apparent motion of lines is in opposite direction to actual motion.

ii. Results with GA

The genetic algorithm from MATLAB was used with a designed cost function based on the cost function in the paper by the reference [40]. The cost function works according to the pseudo code:

1. Map new scan into old scan given the transformation from the genetic algorithm
2. Work out the correspondence of points in the two sets
3. Work out the errors between the points matched in step 2
4. Return the sum of errors squared

The GA from MATLAB produces a transformation and tests it by passing it to the cost function developed (that follows steps 1-4 mentioned above) to minimize the error squared. The default options in GA from MATLAB were used but the population range was set between -200 and 200 for Δx and Δy and -10 to 10 for Δth. Further, the number of chromosomes was set to 100.

The set rotated 5 degrees was first tested. The GA algorithm produced the result:

```
>> Transform=GA(@ICPGA,3)

Optimization terminated: average change in the fitness value less than options.TolFun.

Transform =

   0.6056    0.5022    0.1216
```

Where the output has the form (Δx, Δy, Δth). This is far off from the actual result. However the algorithm was run several times (despite the significant times it takes) as it is stochastic in nature. The second run generated:

```
>> Transform=GA(@ICPGA,3)

Optimization terminated: average change in the fitness value less than options.TolFun.

Transform =

   0.4578    0.9626    3.5867
```

Which still shows some error. The algorithm was tested with the sets rotated 1 degree apart. The results are shown.

```
>> Transform=GA(@ICPGA,3)

Optimization terminated: average change in the fitness value less than options.TolFun.

Transform =

0.3763    5.1517    0.9835
```

A second run yielded:

```
>>>> Transform=GA(@ICPGA,3)

Optimization terminated: average change in the fitness value less than options.TolFun.

Transform =
```

0.6422 0.6065 0.6300

The algorithm was run with the set displaced 200mm in the y direction and yielded the results:

>> Transform =GA(@ICPGA,3)

Optimization terminated: average change in the fitness value less than options.TolFun.

Transform=

-0.6263 -0.2873 2.0825

The results from GA were not correctly matching to the transformations. It was not possible to reproduce any work similar to that of reference [40] as each run was different due to stochastic nature. Further, some errors are definitely introduced which is attributed to the correspondence problem where wrong points are matched together as possible candidates.

iii. Results with ICP

The iterative close point algorithm was run as described in [40] to find the transformation that minimizes error. Figure 111 shows the two environments with the matched points shown in red.

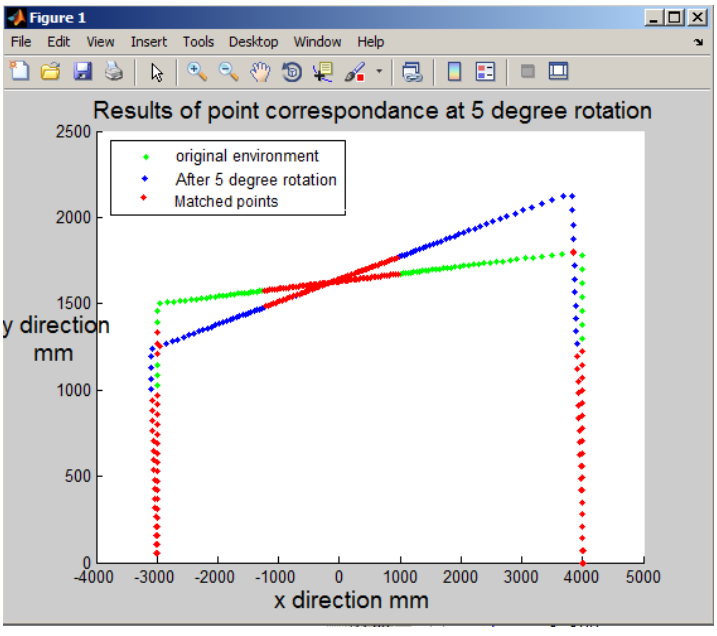


Figure 111: The result of point correspondance in ICP for two frames.

It is clear some points are mistakenly thought to belong to other points. This correspondence error is responsible for the wrong answers observed here and in the GA section. The

correspondence step is done by finding the distance of each point from all other points. The nearest neighbor is chosen as the corresponding point. However, this may not be correct in many cases.

At 5 degrees the percentage overlap was found to be 60%. The transformation found on first iteration was:

Δx =-27.0252

Δy =-537.0326

Δth = -0.0902

The second iteration yielded:

Δx = -878.2137

Δy = -6.1379e+003

Δth =2.7117

Clearly, the algorithm is diverging. At one degree rotation the algorithm showed good matching. The matching points led to a high degree of overlap at 99%. However, not all points were correctly paired. The first iteration gave errors of 90 mm in x and y with 0.5 errors in degrees. The second iteration diverged and no useful result was obtained. A Third run produced close enough results with errors about 9mm in displacement and 0.1 in angle. In the third case, the environment was displaced by 200mm in the y direction. The corresponding map is shown in figure 112.

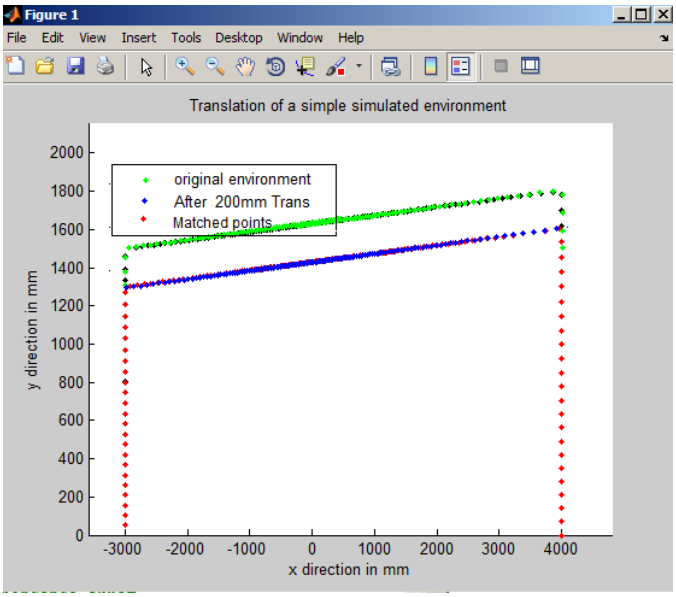


Figure 112: The result of point correspondance in ICP for two frames.

Only 23.2% of the points were matched and the results obtained as expected were erroneous.

$\Delta x = 1.6511$

$\Delta y = 1.3243$

$\Delta th = 0.0018$

The second iteration diverged and the desired output was not obtained. It must be noted that the iterative point algorithm (ICP) was repeated with different thresholds for correspondence matching a well as outlier detection but improvements were slight. The reasons for the failure to reproduce results by [40] could be due to the nature of points used since they have used an outdoors environment where geometric variation is plentiful making the correspondence of points more accurate. Further, point to point scan matching work best when in an unstructured environment that will not lead to correspondence ambiguity.

4.4.6 Discussion

The scan matching algorithm developed based on features performs better than algorithms based on point to point correspondence from a time and processing speed point of view. It was compared to ICP scan matching, and GA optimized scan matching with the results from the developed algorithms more accurate and performing faster. The point to point scan matching may fail if there is a significant change in pose as some points may get obscured or disappear or

new points appear that can be mistaken for previous points resulting in errors. Further, GA and ICP based GA methods are stochastic and will not yield the same result every time making testing a lengthy process.

On The other hand the developed method relies on good estimates of extracted lines used as features and thus performs badly in environments with poor lines or no lines. Further, the developed method is sensitive to noise since the segmentation relies on a differentiation approach despite the filtering operations to minimize this effect. Moreover, the algorithm may produce wrong results with lines that are very close to 90 degrees gradient with respect to the robot local frame and the reason is not known but could be due to the use of the Cartesian properties of lines instead of polar. The results from the scan matching algorithm are used with unscented Kalman filter to fuse them with predicted robot motion. This is discussed in section 4.5.

4.5 UKF algorithm

The Kalman filter is an important tool for data fusion algorithms in use today with important properties such as being the optimal estimator for systems with Gaussian error statistics and its recursive nature [59]. In the process of state estimation, measurements are used with a system model to obtain optimal estimate using Kalman filter. However, an estimate of the process and measurement noises is needed. These are generally not known which requires empirical estimates or use of tuning techniques that modify the covariances of measurement and process noises until the filter performance is optimized [61]. Most of the time in robotics either the EKF or UKF are used since state equation is non-linear and Kalman filter is valid for linear systems only.

4.5.1 The EKF

The EKF in many applications is the standard tool for non-linear state estimation and machine learning problems. The EKF overcomes the limitations of the Kalman filter by using a linearization approach [60]. This effectively makes the EKF a linearized version of the Kalman filter. However, the EKF is only valid for systems which are not highly non-linear and which are almost linear in behavior on the time scale of the updating intervals as it will diverge if the approximated function by the filter is not approximating linearly behaving during updates [64].

To overcome those limitations, the unscented Kalman filter was used (UKF) for better estimation of non-linear processes.

4.5.2 Unscented Kalman filter (UKF)

i. introduction

The UKF overcomes the problems associated with the EKF as it avoids linearization all together and instead of that represents the Gaussian random variable using a minimal set of specifically selected sigma points that capture the statistics of the random variable making it suitable for approximating the mean and covariance of any non-linearity accurately to the third order [60, 64].

ii. Equations of the UKF

The number of sigma points is chosen according to dimensionality of the problem [65]. Thus for 1 dimensional state vector three sigma points are needed. The first sigma point is the mean, the second and third points serve to capture the spread (variance) of the variable and are given by:

$$\boldsymbol{x_{2,3} = \bar{x} \pm \sqrt{(n+\lambda)\sigma}} \quad \textbf{(73)}$$

Where $\bar{x}$ is the mean, n is the dimensionality of the state, σ is the variance of the GRV (covariance in multidimensional states) and λ is a scaling factor given by:

$$\boldsymbol{\lambda = \alpha^2(n+k) - n} \quad \textbf{(74)}$$

Where α and k are scaling parameters [65]. The mean and covariances of sigma points are computed using predefined weights. The first set of weights is used to compute the mean and the second is used to compute the variance. For the first sigma point (mean) the mean weight and the variance weight are:

$$\boldsymbol{w_{m_0} = \frac{\lambda}{n+\lambda}} \quad \textbf{(75)}$$

$$\boldsymbol{w_{\sigma_0} = \frac{\lambda}{n+\lambda} + \left(1 - \alpha^2 + \beta\right)} \quad \textbf{(76)}$$

Where β is chosen according to the distribution type. For a Gaussian distribution the optimum value is 2 [65]. For the rest of points the two weights (the mean and the variance weights) are equivalent and given by:

$$\boldsymbol{w_{\sigma_i} = w_{m_i} = \frac{1}{2(n+\lambda)}} \quad \textbf{(77)}$$

The sigma points can be propagated through any non-linear function and the properties estimated accurately [65]. The choice of parameters k, α, and β determine the spread and nature of the distribution β is set to 2 for Gaussian distribution while α is usually a small number.

4.5.3 UKF Algorithm for fusion of odometry and scan matching

The scan matching algorithm developed is used to update the robot position using the unscented Kalman filter. A function called UKF was developed for this purpose and the function will need two laser scans at least to match. Thus the first step is to find the current number of scans to know whether a new scan must be read or two scans compared.

Once two scans are obtained and segmented, the properties of the segmented lines must be found for data association. These are the number of laser point, the distance between the line endpoints, the gradient, and the error. Once paired, the orientations of the lines are worked out and subtracted to find the expected rotation. However, consider figure 113 where the line changes quadrant due to the robot rotation. The atan function will return about 80 degrees for figure 113 (a) and -80 for 113 (b). This will yield a wrong rotation of 160 degrees. To correct this issue, if the gradient is found to be a negative angle 180 is added to the result.

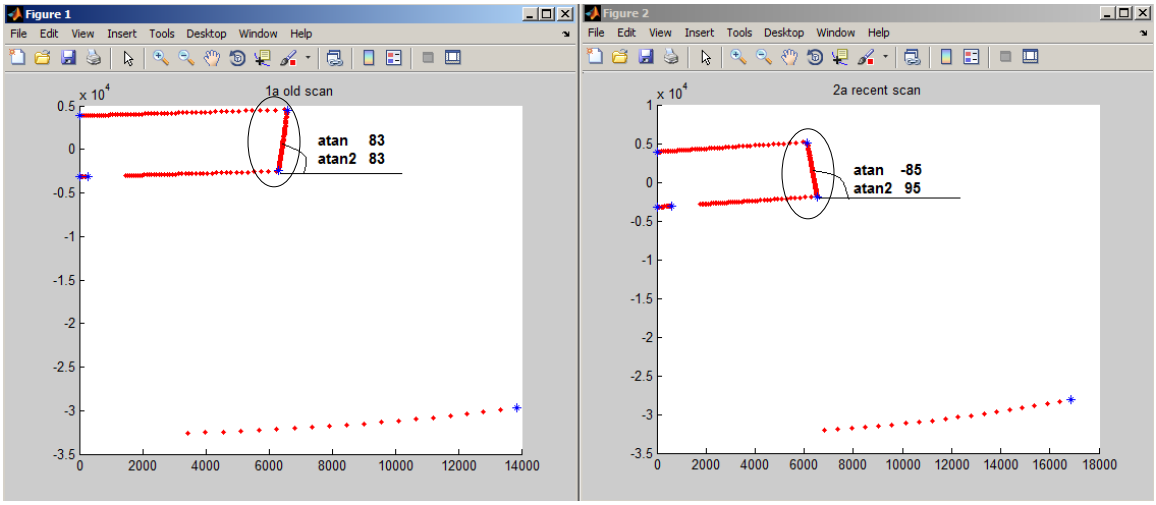


Figure 113: a problem that could arise due to line orientation and use of atan function.

The flowchart for this function is shown in figure 114. The function is called by the main code periodically every time the robot moves more than 0.1m or rotates more than 5 degrees.

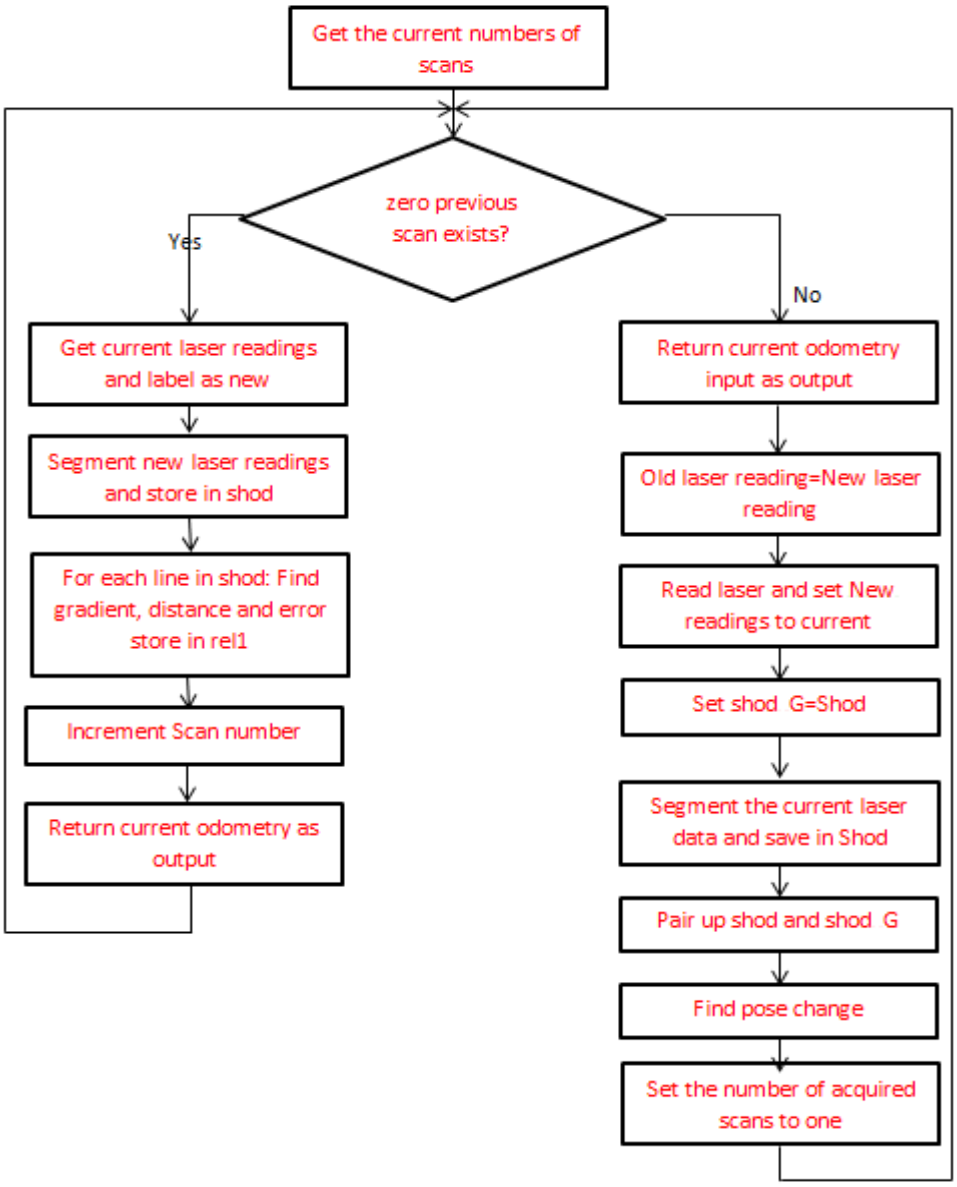


Figure 114: The flow chart for the scan matching producing inputs for UKF function.

The process of pairing up the two segmented arrays shod and shod_G was explained in the flow chart in figure 90 while the process of finding pose flow chart is in figure 115. Figure 90 is the data association step which is very important for the success of the algorithm as wrongly paired lines will give rise to wrong scan matching. Although the weight function helps to lessen this effect, it cannot continuously accept errors in data association. Environments which show highly similar line features or ambiguous for the robot can cause a failure of the scan matching algorithm increasing the odometry error. This is because in case of algorithm failure (high errors

in lines or low association weight), the robot switches to believing its internally predicted state more compared to measurements due to the operation of the UKF.

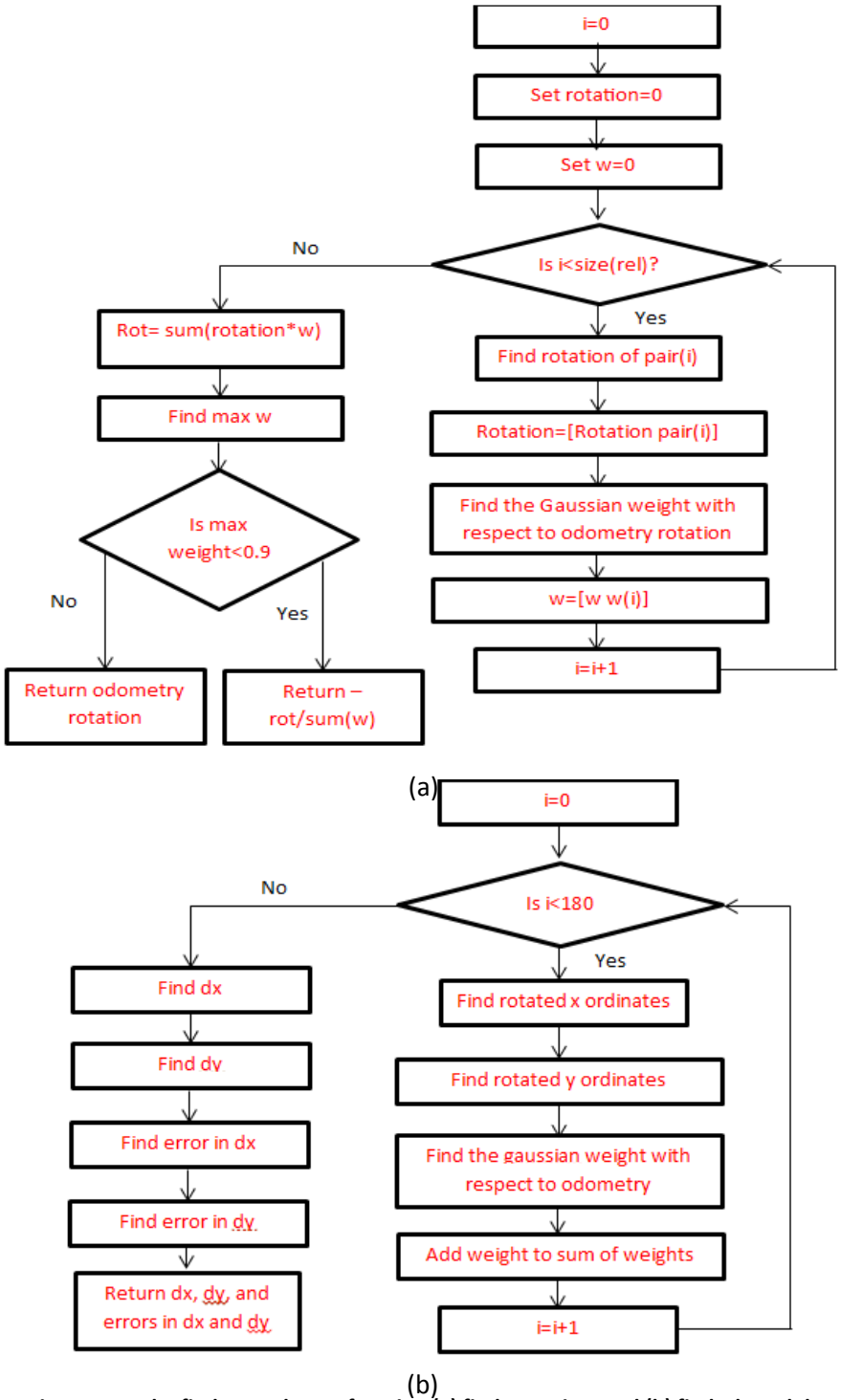


Figure 115: The find pose change function (a) finds rotation, and (b) finds dx and dy.

As can be seen, if the maximum weight is less than 0.9 then the estimate is likely to be erroneous and thus is rejected in favor of odometry. The second part of figure 115 is applied also to rotated line features and results are preferred if they are closer to prediction than point based method.

i. Estimation of noise and measurements covariance

The Kalman filter needs an estimate for process noise and measurement noise. For the process noise, the robot used is the P3DX of the differential drive type that moves only in the x direction, and rotates around its centre or a centre of rotation. Thus, the translation (and hence) the error in y direction are dependent on both the x direction translation and the rotation from the last time step. The errors in x and y forward motion (x direction) are independent.

Experiments were conducted to measure the variation in x, y, and angular position of the robot with various commands and the results were used to find a relation:

$$\boldsymbol{var(y) = 0.01dx + 0.5dth.dx} \quad \textbf{(78)}$$

The multiplication with dx is due to the fact that the error in orientation is propagated to the y direction only when the robot moves. The variance in x position was found to be smaller than that in y position and was hard to measure accurately over a distance of 5 meters but was found to be:

$$\boldsymbol{var(x) = 0.5dthdx} \quad \textbf{(79)}$$

The Variance in orientation is simply:

$$\boldsymbol{var(th) = 0.5dth} \quad \textbf{(80)}$$

For the measurement noise, the measurement noise covariance is built in already in the Gaussian weight function that was used.

The variances developed serve only as empirical estimates for the UKF algorithm. Alternatively, the use of fixed variances showed greater stability in operation. The use of these variable variances sometimes led to poor behavior of UKF although the reason for this is not known. Therefore, fixed variables based on an average change of 10 cm and 5 degrees were used.

4.5.4 Simulation results and discussion

MobileSim was used to test the UKF algorithm with the robot. The robot was moved in a straight line but at an inclined starting angle and the path is shown in figure 116.

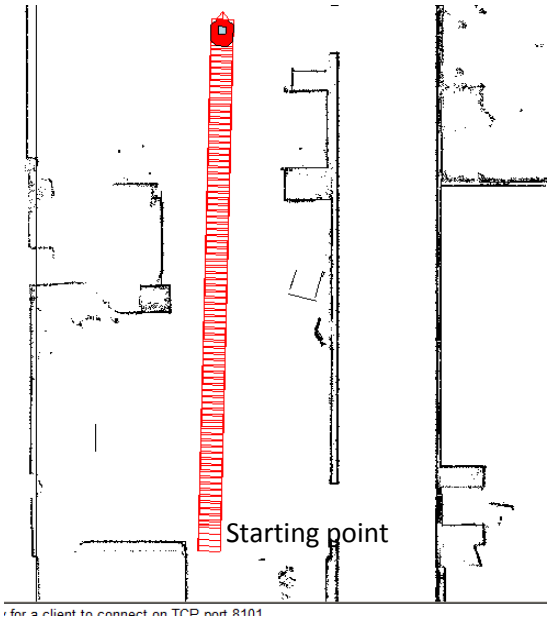


Figure 116: The environment used for first simulation.

The environment is an indoors environment with large dimensions and few observable lines at any point in the environment to makes the scan matching algorithm based on features likely to produce low number of lines and hence will have low accuracy. However, through the use of the UKF the noisy measurement is combined with the odometry estimate to a more accurate estimate. The estimate of x and y positions is shown in figure 117 while the estimate or orientation is shown in figure 118. The output from scan matching at different time steps is shown as well in the same figure. Notice how noisy the scan matching output is due to corridor like environment.

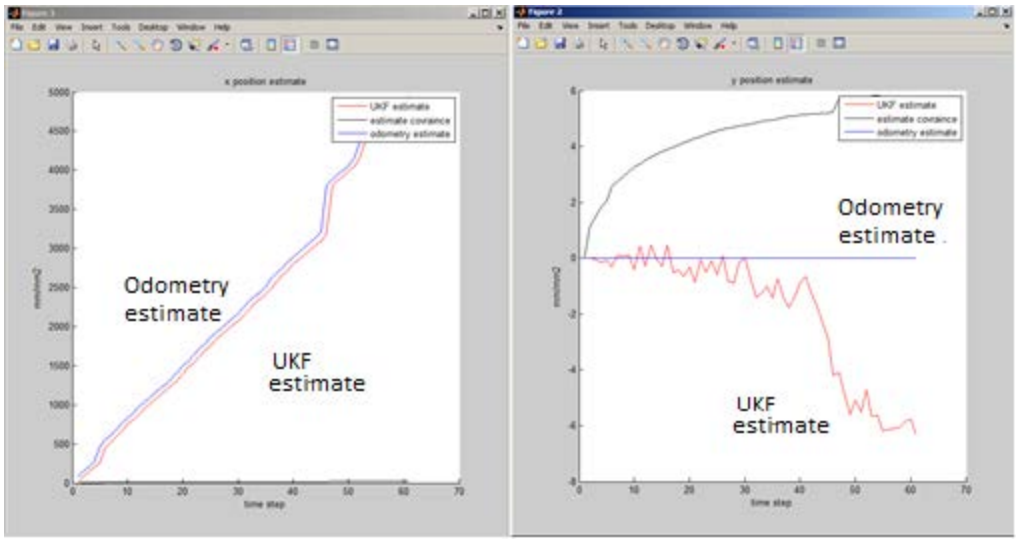


Figure 117: The estimates of x and y positions from UKF versus odometry.

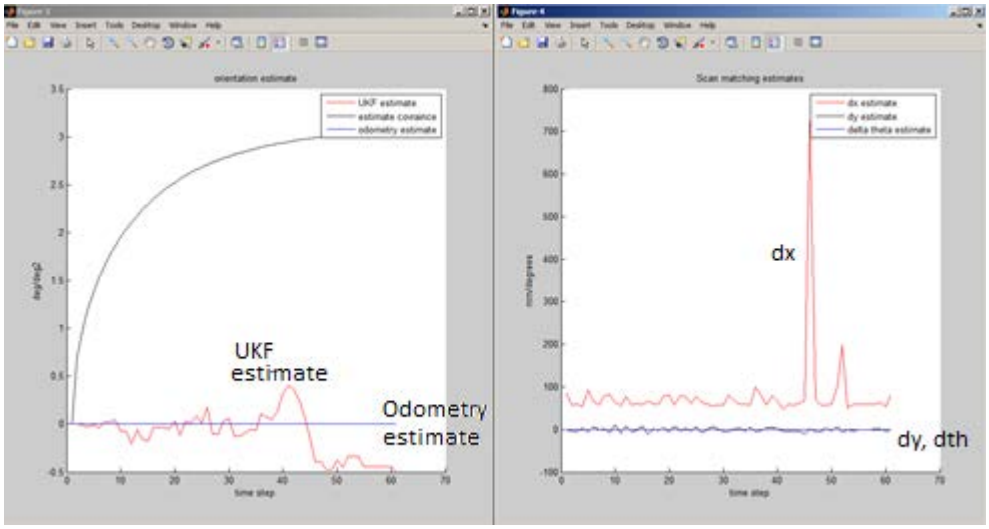


Figure 118: The orientation estimate from UKF and odometry (left) and the output of scan matching function (right).

Notice the how the covariances start at zero and increase but to a steady state value. The y position estimate shows slight negative result in accordance with inclination of the robot. The peak in figure 118 is attributed to wrong data association and decreases as soon as the lines are correctly associated. In the second simulation the robot was moved in a straight line but the inclination angle was increased considerably to see if it affects the accuracy of scan matching and UKF algorithms. The path and the environment are show in figure 119.

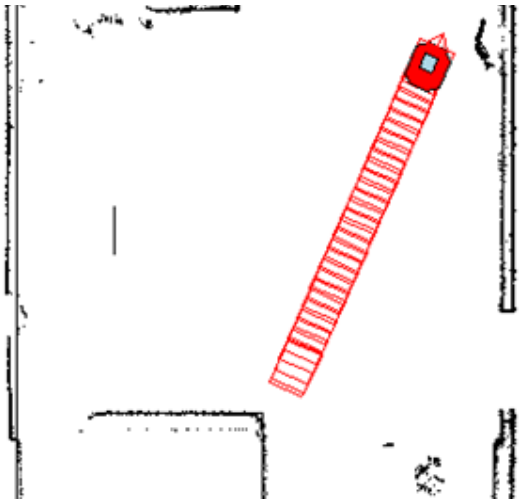


Figure 119: The environment used for the second simulation.

The estimate of x and y positions is shown in figure 120 while the estimate or orientation is shown in figure 121. Further, the output from scan matching at different time steps is shown as well in the same figure. Notice how noisy the scan matching output is.

Notice the how the covariances start at zero and increase but to a steady state value. The y position estimate shows slight negative result in accordance with inclination of the robot. However, the value is larger due to the larger inclination.

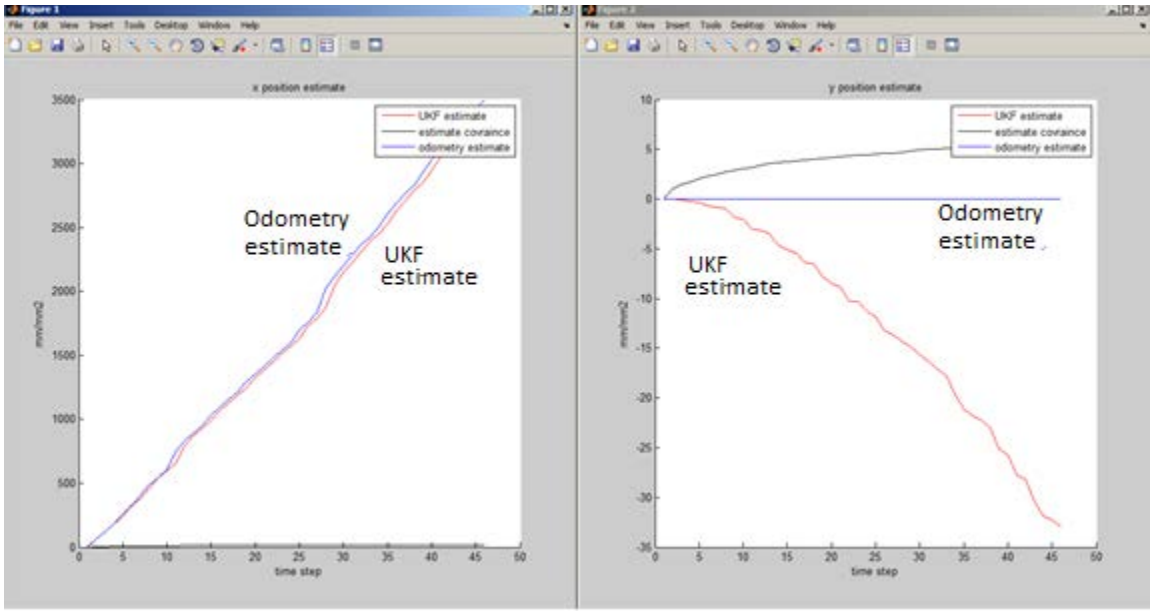


Figure 120: The estimates of x and y positions from UKF versus odometry.

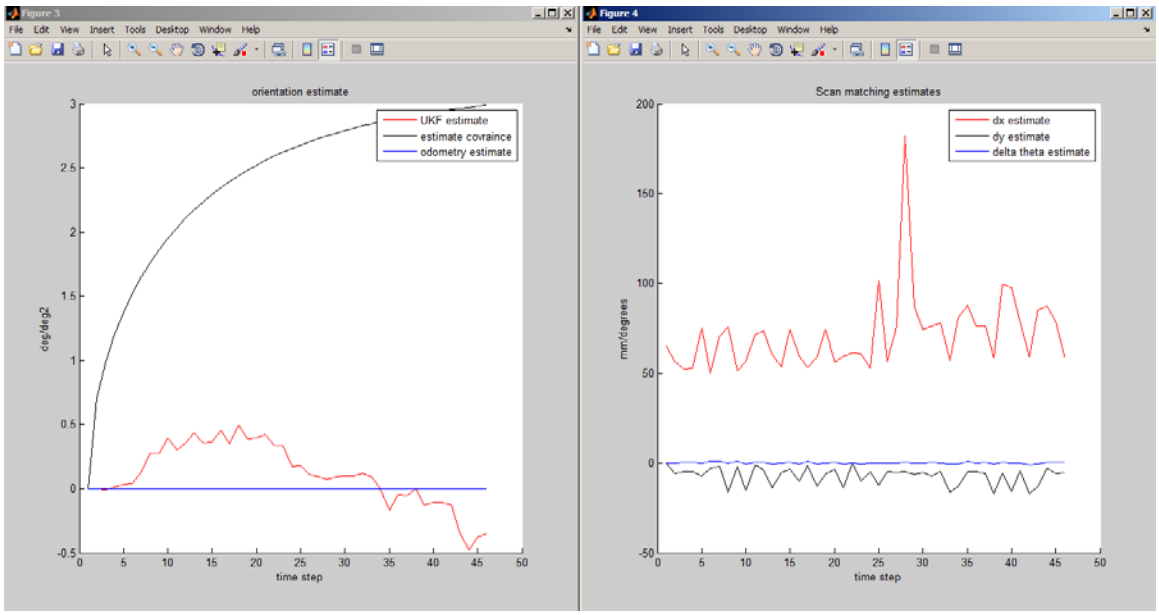


Figure 121: The orientation estimate from UKF and odometry (left) and the output of scan matching function (right).

The robot was then tested in a purely corridor environment with only two lines on the side. This environment is difficult compared to typical indoors environments where many lines are found.

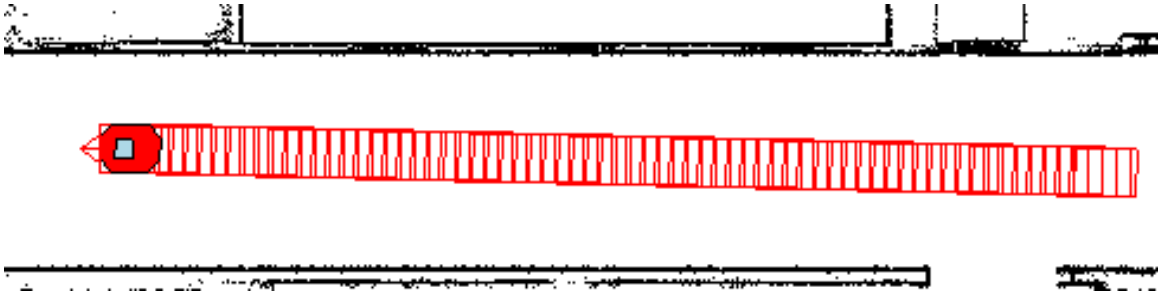


Figure 122: The corridor environment used for the third simulation.

The estimate of x and y positions is shown in figure 123 while the estimate or orientation is shown in figure 124. Further, the output from scan matching at different time steps is shown as well in the same figure. Notice how noisy the scan matching output is. Notice the how the covariances start at zero and increase but to a steady state value. The y position estimate shows variation around zero in accordance with motion of the robot.

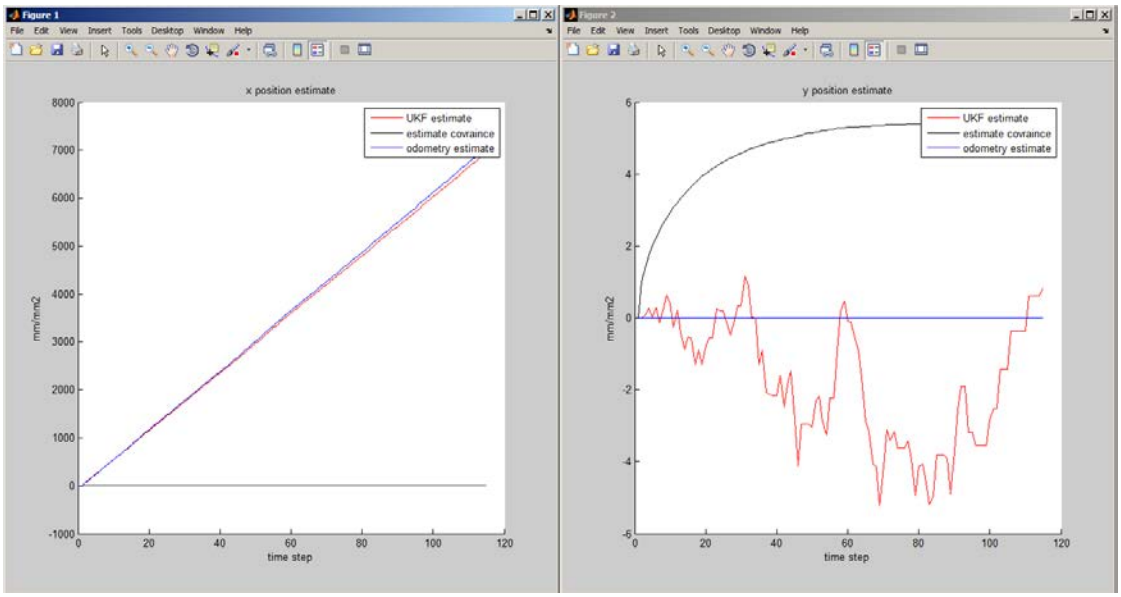


Figure 123: The estimates of x and y positions from UKF versus odometry.

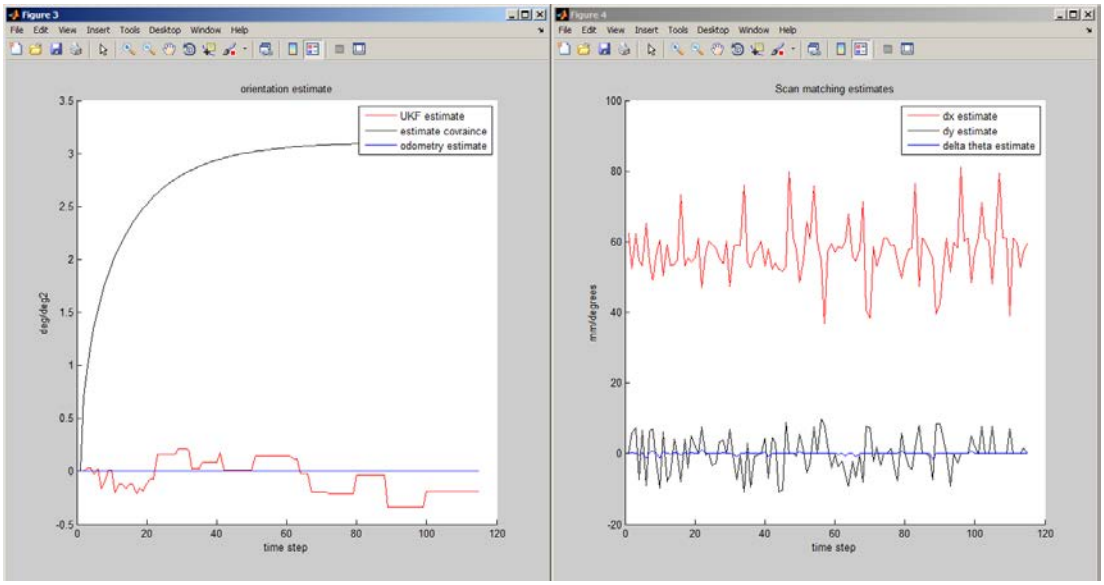


Figure 124: The orientation estimate from UKF and odometry (left) and the output of scan matching function (right).

The robot was tested this time by executing circular motion where the rate of change of orientation is significantly larger than the previous cases. The path followed is shown in figure 125 with the starting position marked with black circle. The robot executed one and half rotation around the circle. The estimate of x and y positions is shown in figure 126 while the estimate or orientation is shown in figure 127 Further, the output from scan matching at different time steps is shown as well in the same figure. Notice how noisy the scan matching output is.

Figure 125: The robot in circular motion

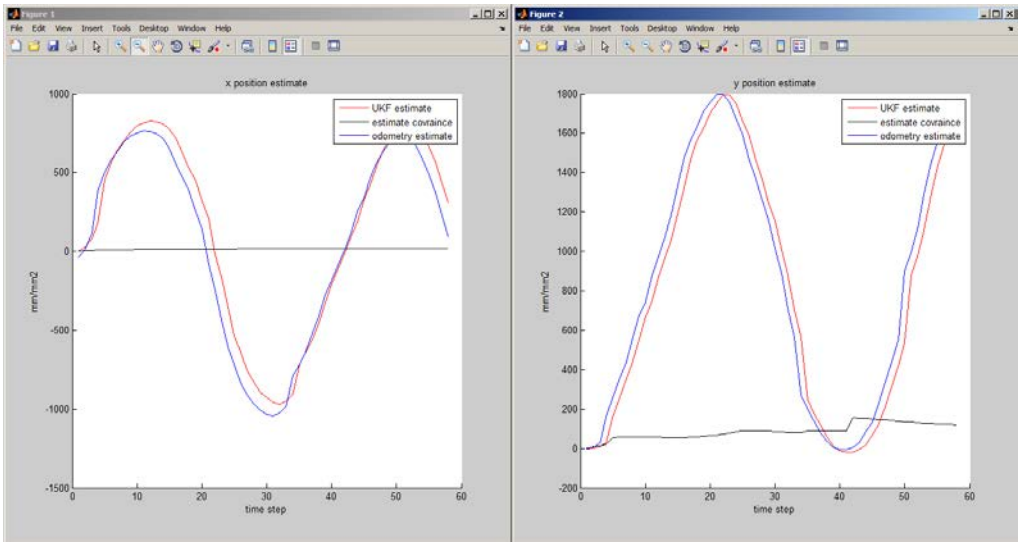


Figure 126: The estimates of x and y positions from UKF versus odometry.

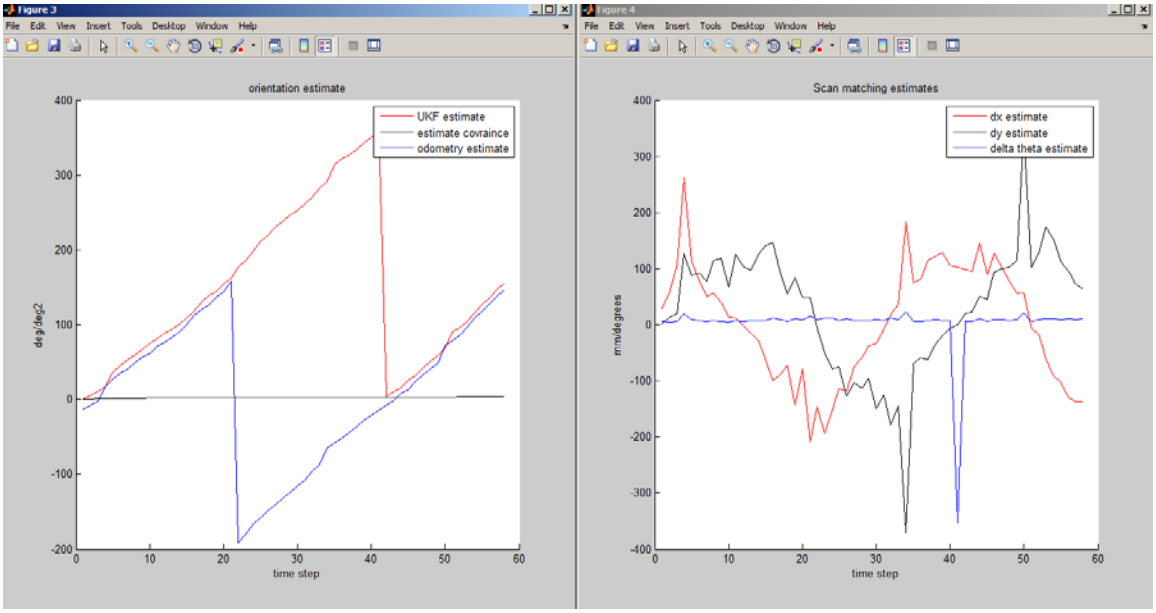


Figure 127: The orientation estimate from UKF and odometry (left) and the output of scan matching function (right).

Notice the how the covariances start at zero and increase but to a steady state value. At time step 40 there is increase in covariance due to change in view angle as the robot enters the new rotation. However, following that the covariance falls again. Further, note that the orientation estimate of the UKF varies between 0 and 360 degrees while that of the odometry varies between 180 and -180 since it considers angles above 180 negative. The slight drift upwards is due to the fact that UKF starts one "frame" later than odometry since it needs two consecutive laser frames to produce its estimate. This can be solved by adding the dx, dy, and dth for thr first frame.

4.5.5Discussion

The use of filters to fuse data from different sources can overcome noisy sensor data which would otherwise lead to noisy estimates of robot pose. Further, the use of scan matching based on features and local change is fast to process making it real tie process. However, the use of features means that scan matching fails to provide enough data in featureless spaces or in some environments may detect only few features reducing the accuracy of pose estimate change. Therefore, an approach combining point to point and feature to feature scan matching algorithms would be more robust and faster than either. In its basic form it would start using feature matching and a variance threshold is used to switch to point to point matching if needed.

The presence of inclined lines in the environment for prolonged periods of time may lead to slight errors in the y estimate of position. This is due to the use of the local frame of reference for scan matching. However, the effect may be overcome by the use of an algorithm to detect line gradients and find values for change in x and y that minimize the error between expected and actual y intercept. This was implemented but the results usually showed large errors mainly due to the lines with angles close to 90 degrees as these have infinite (practically very large) value of y intercept.

However, the effect of inclined lines is likely to be less obvious in small local environment especially with the presence of dynamic objects that could block the view of the lines from the robot. The UKF algorithm used was proven n the experimental grid map building in section 3.9.

4.6 Grid map

4.6.1 Grid map structure

The grid map developed was intended to be scalable and growing with the robot motion. However, the problem of increasing computational load will eventually be encountered rendering the algorithm non real time. Therefore, a method was developed where the map is segmented into tiles of equal size (number of grid cells within each tile is the same) and as the robot moves on and explores new areas new tiles are added as shown in figure 128 it can be seen that when the robot navigates within a tile, it only considers the current tile for the updating of grid cells i.e. updates only the local tile. However, if a ranged sensor reading extends into another tile then the robot shifts from the current tile to upgrade grid cells in the tile where the obstacle is then shifts back the active to the tile it is located into.

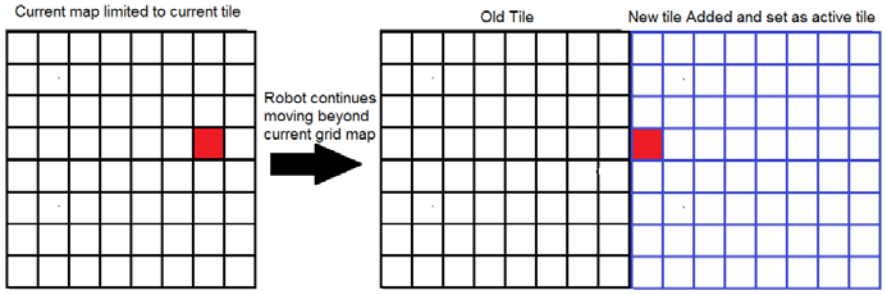


Figure 128: The process of extending the grid map by adding new tiles as the robot moves on.

Another difficulty to be addressed is to reduce the computational cost of updating grid cells within the local (active) tile. Even after localizing the update process, the computational load of updating a large number of grid cells within the same tile cannot be carried out in real time. Therefore, a fast sensor model has to be developed for both the laser and sonar sensors. The segmentation of global map will be discussed first then sensor models afterwards.

i. Map segmentation

Map segmentation is carried out to enhance the computational efficiency of the algorithm as well as render the algorithm scalable. The segmentation is carried out by breaking the grid map in to a series of smaller grid maps which are not predefined but rather are added as the robot navigates. The robot starts in the first tile (home tile) positioned in the center of the tile. The tile has the dimensions shown in figure 129.

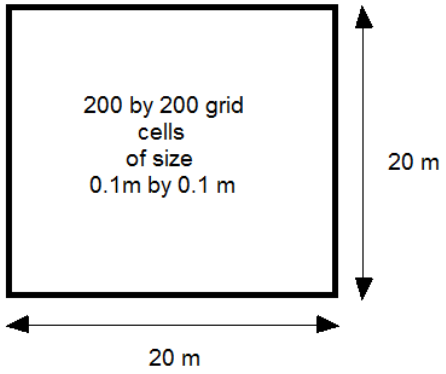


Figure 129: The dimensions of the tile.

The center position of each tile is given the coordinates of (100,100) i.e. cell number 100 in x and 100 in y (center of tile). Given a robot position (x_r, y_r), the grid cell in which the robot exists is given by the function:

$$x_{cell} = 100 + \frac{ceil(x_r - 50)}{100} \qquad y_{cell} = 100 + \frac{ceil(y_r - 50)}{100} \qquad (81)$$

Where the function ceil is the ceiling function that rounds up variables to the nearest whole number, subscript r indicates robot global variable (pose) in metric space in mm, and subscript cell indicates the grid cell domain. Millimeters are used since the robot odometer outputs readings in mm. If a ranged reading (from a laser or sonar) to an obstacle has to be mapped to a grid cell, then the obstacle global position is first obtained from the robot global position and its position in the robot's local frame and the following function is used:

$$x_{obstaclecell} = 100 + \frac{ceil(x_p - 50)}{100} \qquad y_{obstaclecell} = 100 + \frac{ceil(y_p - 50)}{100} \qquad (82)$$

Where subscript p indicates the global position of the obstacle in the metric domain in mm.

Equations 81 and 82 are only applicable to grid maps of size 20 meters by 20 meters with grid cells of the size 0.1 m by 0.1 m.Furthermore, the opposite conversion from grid cell domain to metric domain is desired in parts of the algorithm. In order to carry this out it is assumed that each grid cell location is represented by the metric coordinates of its center thus if the robot exists in cell (100,50) then its metric location in tile can be found using:

$$x_r = 100 x_{cell}\, mm \qquad y_r = 100 y_{cell}\ mm \qquad (83)$$

However, equation 83 is modified (to be equation 198) due to the segmentation of grid maps into tiles.In equations 81 and 82 the grid cells is considered to be in a different tile if its

coordinates are outside the range of the local tile. For example, if the robot in tile 1 senses and obstacle and using equation 83 finds that the coordinates of the obstacle are (120,230) then the y ordinate is not within the limits of the tile (0-200) and hence the obstacle lies in a tile northern to the current robot tile. Thus, the robot needs to keep a stack of tiles ordered labeled with their spatial positions relative to one another.

ii. Stacking of map tiles

The grid map is expanded as the robot moves by the addition of tiles. Those tiles are stacked and indexed according to their position relative to the starting tile. Essentially, the grid map becomes a stack of tiles arranged in the order in which they were created and indexed by their location. The stack of tiles is shown in figure 130 where the subscript indicates the order in which the tiles were created. Therefore T_1 is the home tile and all other tiles are relative to it.

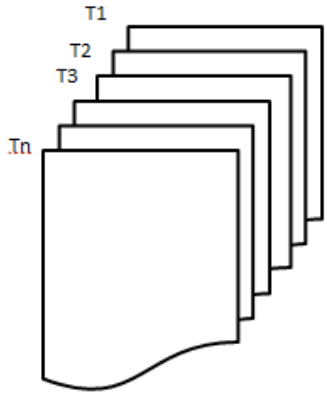


Figure 130: The grid map is a set of tiles arranged in the order in which they appear.

The index which is assigned to a tile depends on its geometrical relation to the starting tile which is given an index of (0,0) i.e. zero x position and zero y position.

The robot knows it has reached the end of the current tile if local cell address from equation83 exceeds the limits of the tile (0-200) in which case a new tile is created in the appropriate location and set as a working tile. Therefore, when converting from grid cell domain to metric domain the working tile has to be taken into account and this modifies equation 83 to become:

$$x_r = 200T_x \times 100x_{cell}\ mm \qquad y_r = 200T_y \times 100y_{cell}\ mm \quad (84)$$

Where T_x is the x ordinate of tile index, and T_y is the y ordinate of tile index while other symbols have the same meanings as in equation 83.

4.6.2 Sensor model

i. Laser sensor model

The laser model used was developed with the size of grid cells in mind as the 10cm by 10 cm cell size means that the probabilistic model within the individual grid cell can be replaced by a different model where the obstacle cell is considered blocked and cells in between the obstacle cell and the laser sensor as unblocked. To limit the computation and enhance the speed of processing, only local readings less than 3000 mm away from the laser were. The laser used is the LMS-100 laser with a range of 15m and update rate of 10 Hz. Figure 131 part (a) shows an example of a laser scan superimposed on the grid cell. The blue lines indicate the laser scan. In this example only one obstacle is found in the cell marked with an x. in thiscase that cell will be marked as blocked and updated using Bayes rule if its distance from the laser is less than 3000 mm and all cells lying on a line from the laser sensor to the obstacle cells will be declared as unblocked and updated as well. The result is shown in figure 131 part (b).

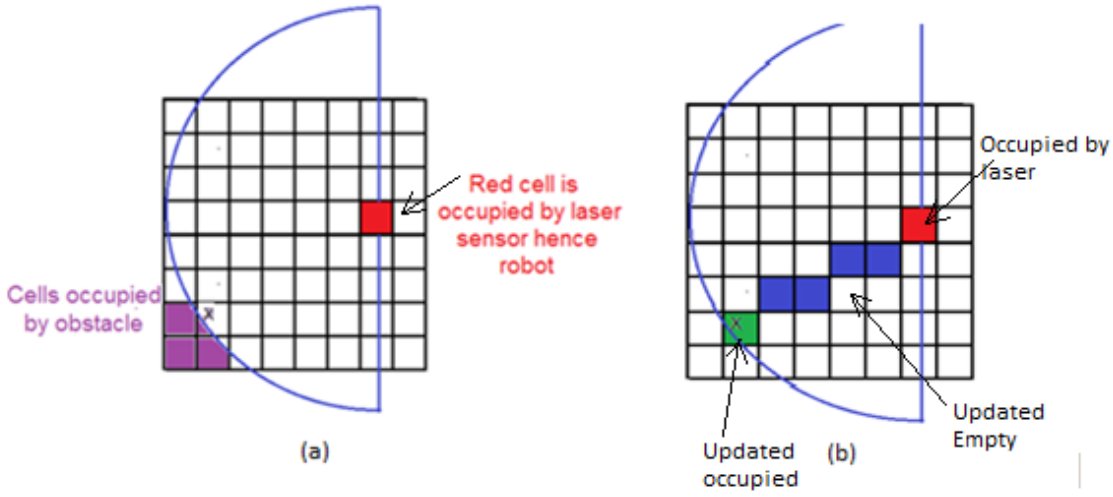


Figure 131: The result of grid cells update. Blue indicates updated empty and green indicates updated full.

Cells in which the obstacle exists are labeled with a probability of 1 of being full. While cells that are to be vacant (blue cells in figure 113) are updated using Bayes rule based on their prior probability and a sensor probability of 0.3. Thus,

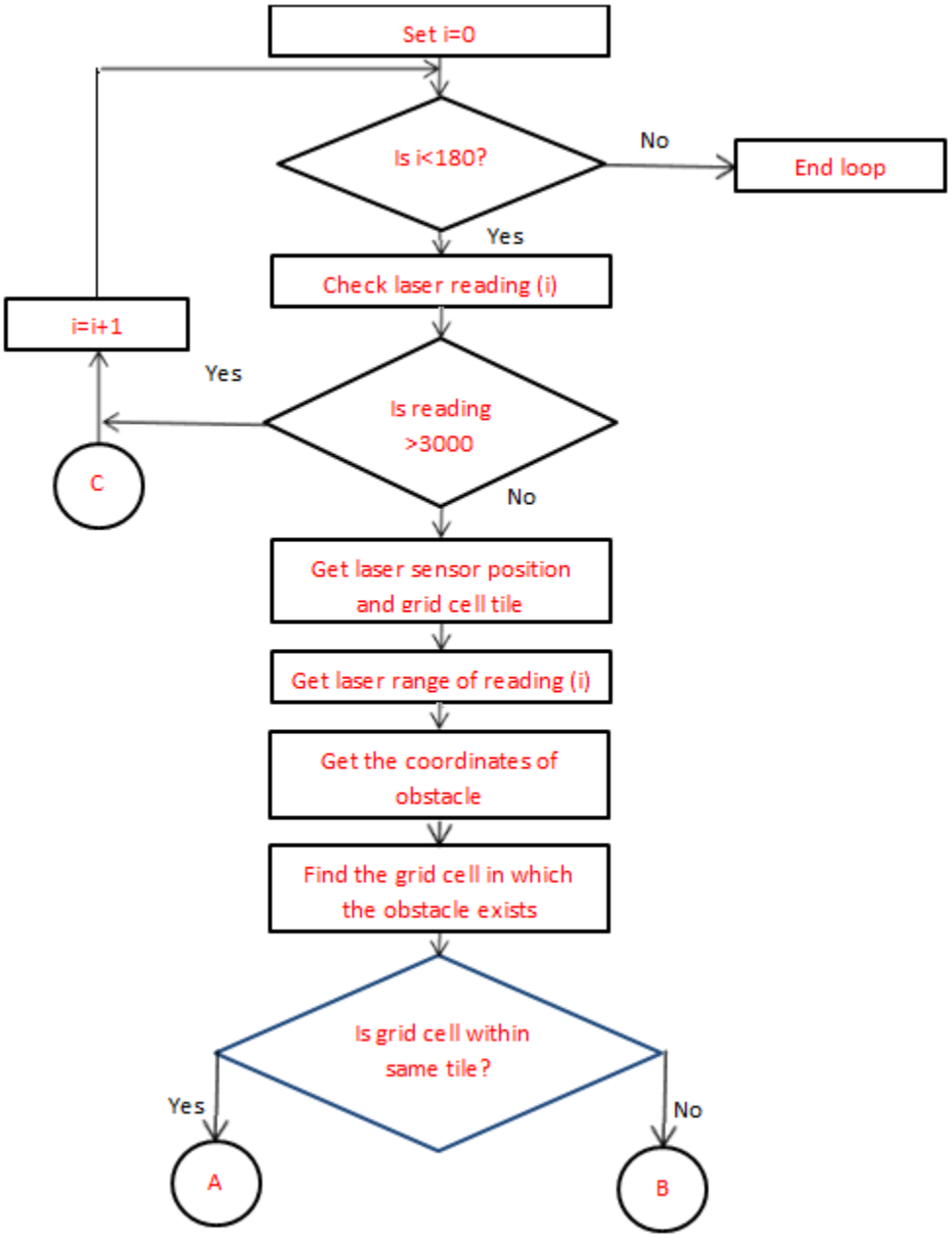
$$\boldsymbol{P(Cell\ occupied|\ Reading)_{m,n} = \eta \times P(Reading|cell\ occupied)_{m,n} \times P(occupied)_{m,n}} \quad \textbf{(85)}$$

Where subscript (m,n) indicates grid cell (m,n). This procedure is repeated 181 times for the 181 laser rays. Although the discretization of the laser rays superimposed on the grid map creates

distortion, the use of probabilistic update overcomes this distortion as the robot moves. This method of update is efficient as it does not update all the grid cells in all the tiles of the map nor does it update al the grid cell in the current tile but instead updates the cells likely to be affected by the laser ray

ii. The flow chart for the laser sensor updating

Figure 132 shows the flow chart describing how the laser sensor data is used to update cells.



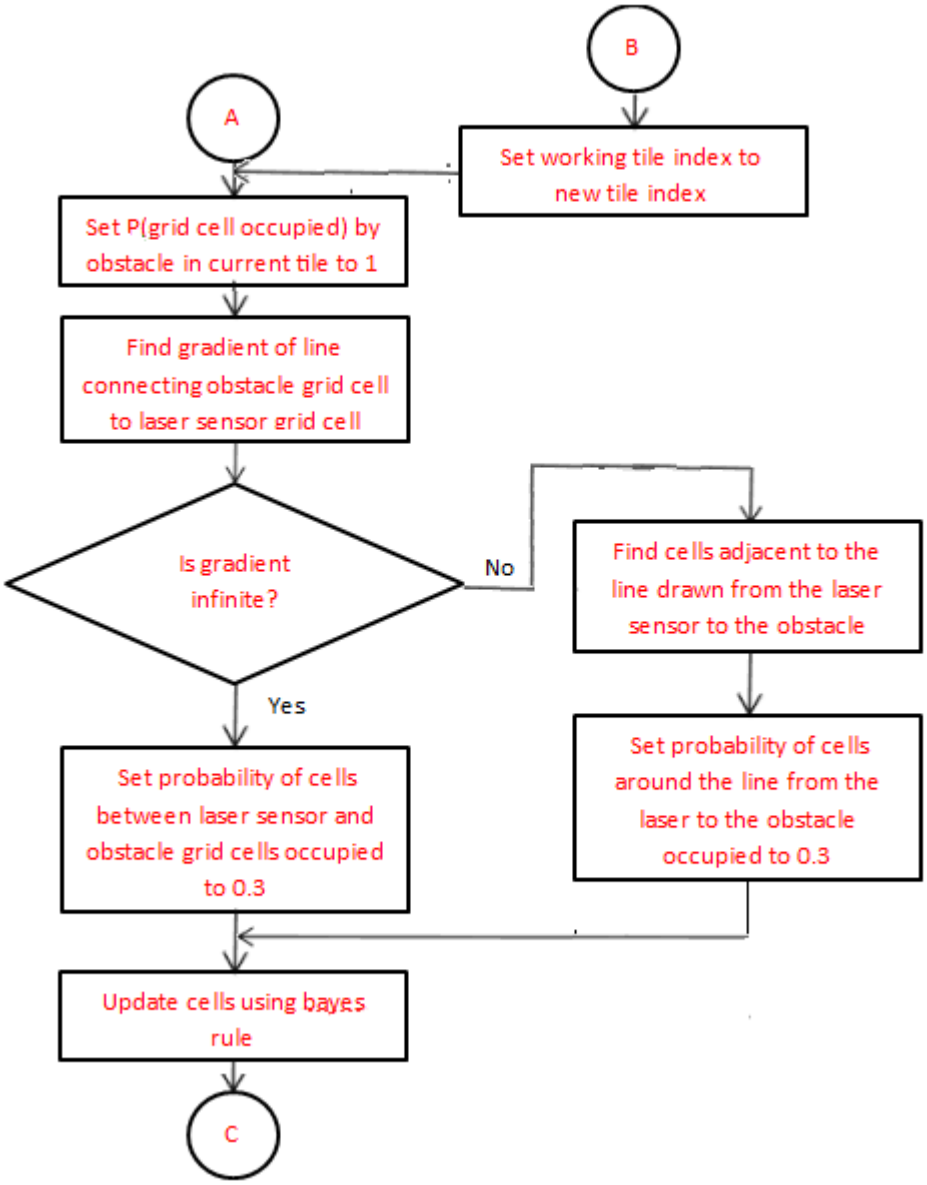


Figure 132: The flow chart for the grid map function that updates the occupancy propabilities of grid cells based on laser sensor.

The flow chart in figure 132 has to check if the grid cell that is to be updated is in the current operating tile or not. Therefore, a function called allocateTile was developed to find the tile in which the grid cell is operating and to convert from global position to local position in a different tile. The desired tile is not found in the stack then it is created by the allocateTile function.

For the map to be more robust to features that cannot be conveniently detected by the laser sensor e.g. glass, the sonar sensor must be used. The fusion of the laser and sonar sensor occurs

in the grid map based on the probabilistic update while giving higher weight to the data presented by the laser sensor.

iii. Sonar sensor model

For the sonar sensor consider figure 133 which shows a sonar beam superimposed on a grid map. The sonar beam is divided into two regions, a green region (region 1) where the obstacle is likely to be and a blue region (region 2) that is likely to be empty.

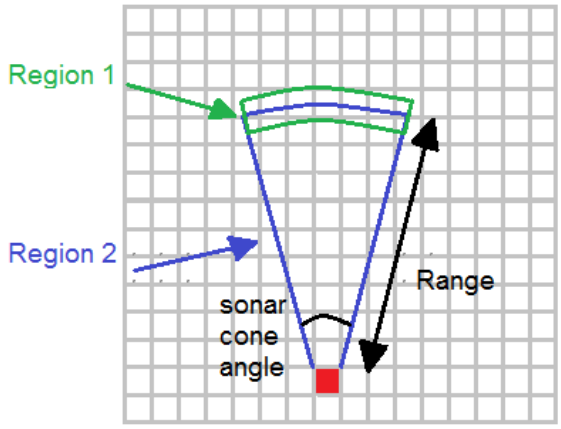


Figure 133: A sonar beam superimposed on a grid map.

The cells are updated as shown in figure 134 in two regions using two equations 86 and 87.

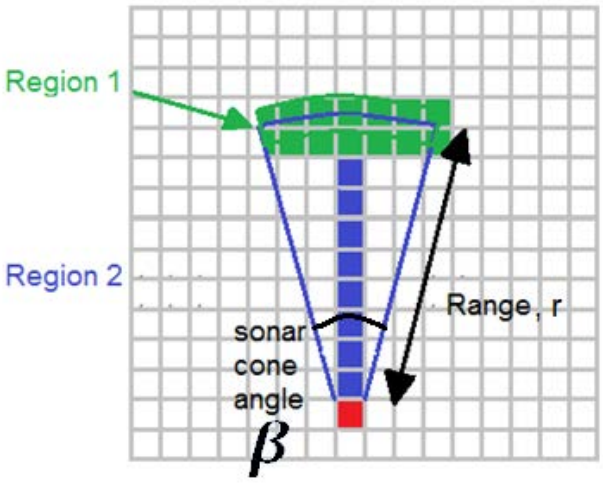


Figure 134: The updating of cells using the devised sonar model.

The only cells updated in region 2 are the cells highlighted in blue. The cells are updated with a probability being occupied according to the equation:

$$\boldsymbol{P(occupied) = 1 - \frac{\frac{R-r}{R} + \frac{\beta-\alpha}{\beta}}{2}} \quad \textbf{(86)}$$

Where R is the maximum sonar range, r is the current range to obstacle, β is the sonar cone angle, and α is the angle of the grid cell with respect to the axis of the sonar cone. This probability is fused with the prior of each cell using Bayes rule. The rest of cells in the blue region are not updated. The cells in region 1 (green in figure 134) are updated using the equation:

$$\boldsymbol{P(occupied) = \frac{\frac{R-r}{R} + \frac{\beta-\alpha}{\beta}}{2} \times Max_{occupied}} \quad \textbf{(87)}$$

Where $Max_{occupied}$ is a predetermined value used to limit the maximum certainty in a grid cell and is set to 0.98. When dealing with grid cells for update with sonar sensor, the allocateTile function is called just as it is called in the laser updating function to check if the obstacle exists in a different tile from the active tile. This is because the sonar sensor is a ranged sensor.

iv. Flow chart for the sonar sensor updating

There are 8 sonars on the P3Dx robot used since there was no back sonar array and the front array has 8 sonar sensors thus the updating process was repeated for the 8 sonar sensors. The sonar model update algorithm works exactly like the laser flow chart but with 8 sonars updated instead of 181 laser readings.

4.6.3 Simulations and comparing the developed sonar model to standard model

i. First trial

The Developed sonar model was compared to the model commonly used from [19]. The robot was moved in a simulation environment for the comparison. The sonar data and robot odometry were collected and used to simulate the grid map in MATLAB. The first simulation environment tested is shown in figure 135.

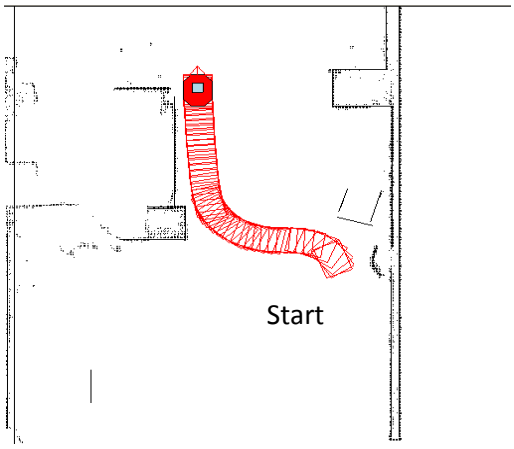


Figure 135: The first trial of the robot in a simulation environment.

The robot moved a distance of about 3.6m and therefore it is expected that the algorithm developed for grid mapping will use only 1 tile. Therefore, the environment was set as a grid map of size 20m by 20m and 200 by 200 cells for proper comparison. First, the standard sonar model algorithm was run where every single grid element is updated for every sonar sensor. This process makes it slow and computationally intensive. The pseudo code is: For every grid cell:

1. i=1
2. if i<9
3. find range and angle to the grid cell from sonar sensor
4. if range and angle are within region 1:
5. upgrade cell using equation 87
6. if range and angle are within region 2:
7. upgrade cell using equation 86
8. increment i
9. if i>9:
10. end

The result of the map produced by the commonly used sonar model that is proposed by [19] is shown in figure 136.

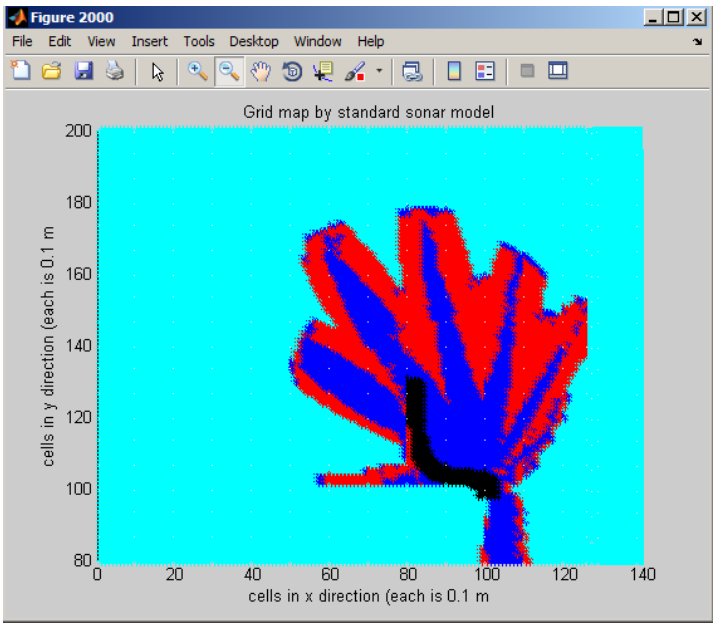


Figure 136: the map for the enviornment in figure 135 using standard sonar model.

The mean time taken for 80 updates was 0.196 seconds (196 ms). After that the developed sonar model was tested. The result with the same environment is shown in figure 137.

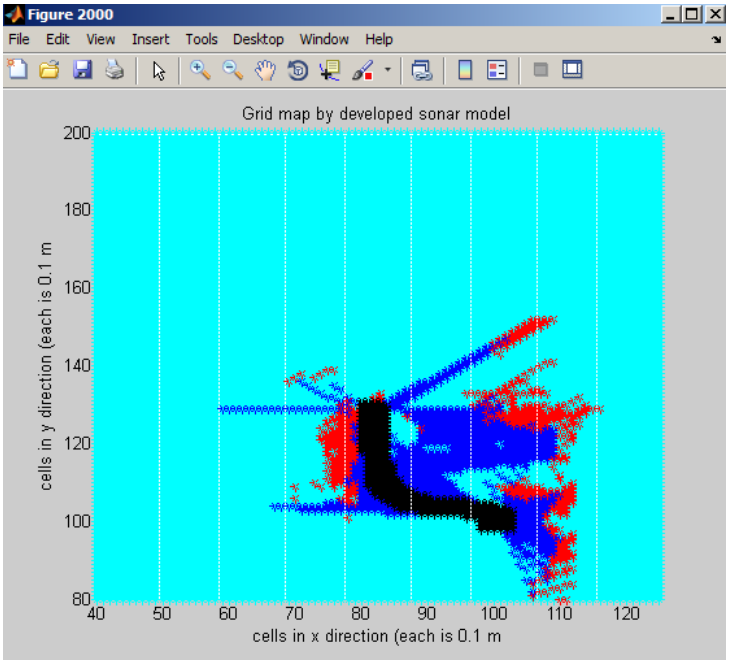


Figure 137:A zoomed version of the map for the enviornment in figure 135 using developed sonar model.

The developed sonar model took on average 0.0078s (7.8 ms) for 80 runs. Thus it was faster than the original model although it missed more details in the environment. However, the general layout of the environment and close obstacles are clear.

ii. Second trial

A second trial was set with the robot exploring more of the simulated environment. The path followed by the robot in the environment is shown in figure 138. The grid map produced by the standard sonar model is shown in figure 139. This map was produced by the standard sonar model this time a larger environment.

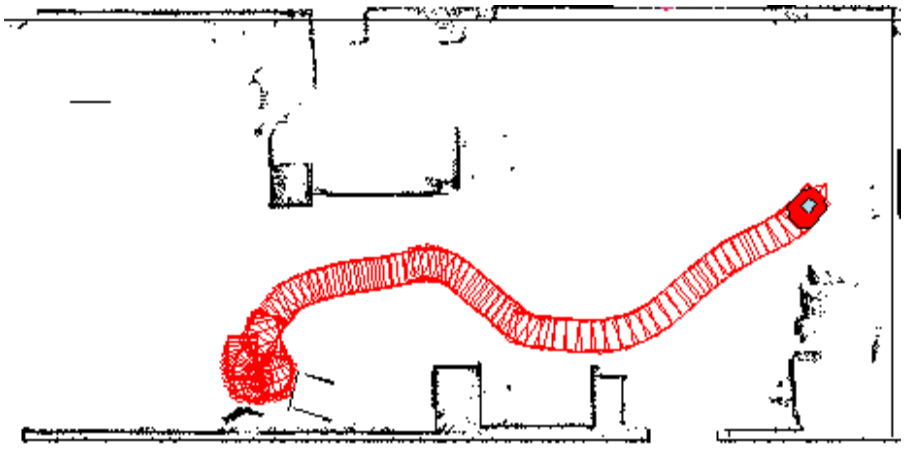


Figure 138: The second trial in the same environment.

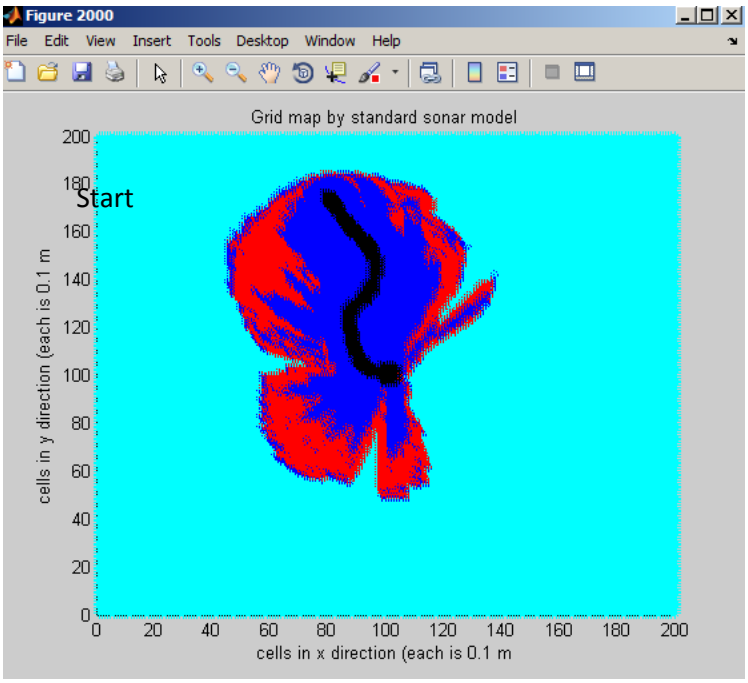


Figure 139: The map for the enviornment in figure 138 using standard sonar model.

The map is made up of 148 runs which on average took 0.1645s (164.5 ms) per run to update all sonar sensors. The developed sonar model was tested and the result is shown in figure 140.

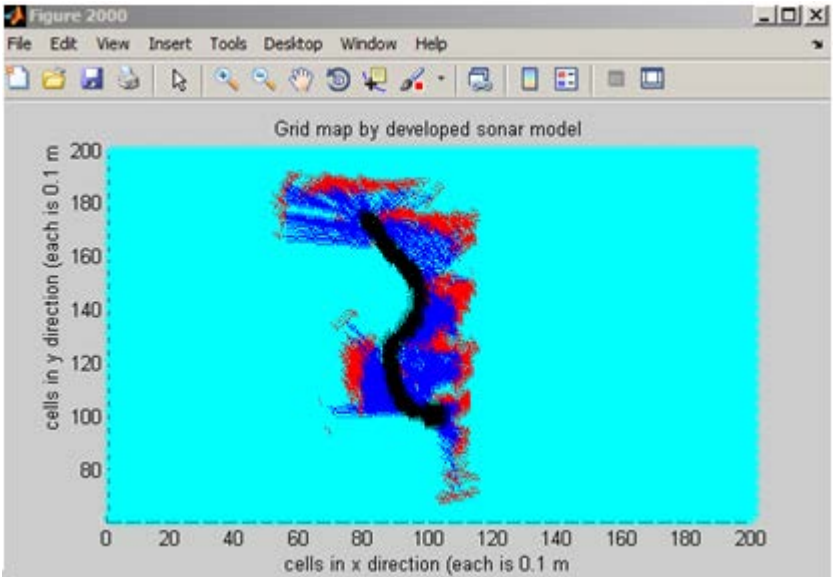


Figure 140: The map for the enviornment in figure 138 using developed sonar model.

It is made up of 148 runs which on average took 0.0058s (5.8 ms) per run. The resulting map looks slightly "empty" as the sonar readings were limited to 3000 mm from the robot. If longer sonar readings were included, the resulting map would be as shown in figure141. However, the time it takes to build this map per scan increases so a compromise distance of 3000 mm is used.

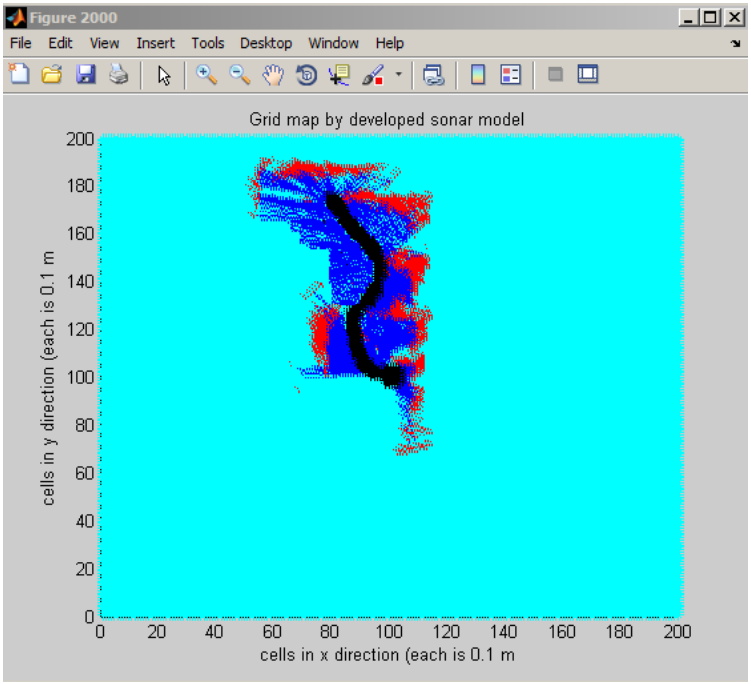


Figure 141: The map with develoed sonar model if longer readings are taken ino aacount.

As can be seen in figure 141 the vacant areas in the map are now considered by the algorithm but previously they were vacant, here implies having a probability of 0.5, i.e. not modified yet by algorithm (Cyan cells).

4.6.4 Experiments

To prove the concept of fusing laser and sonar data in the grid map a well as the use of tiles to extend the map as the robot moves on, experiments were conducted in the mechatronic lab. The robot used was the P3Dx differential drive robot equipped a laser sensor (LMS 200) and a sonar array. The robot was manually driven using joystick with the obstacle avoidance function not functioning but odometry reckoning was functioning. Further, the dynamic objects detection was not functioning too. The robot map was saved and later retrieved for displaying using MATLAB. The map is made of two tiles, the first being tile (0,0) the center tile and is shown in figure 142.

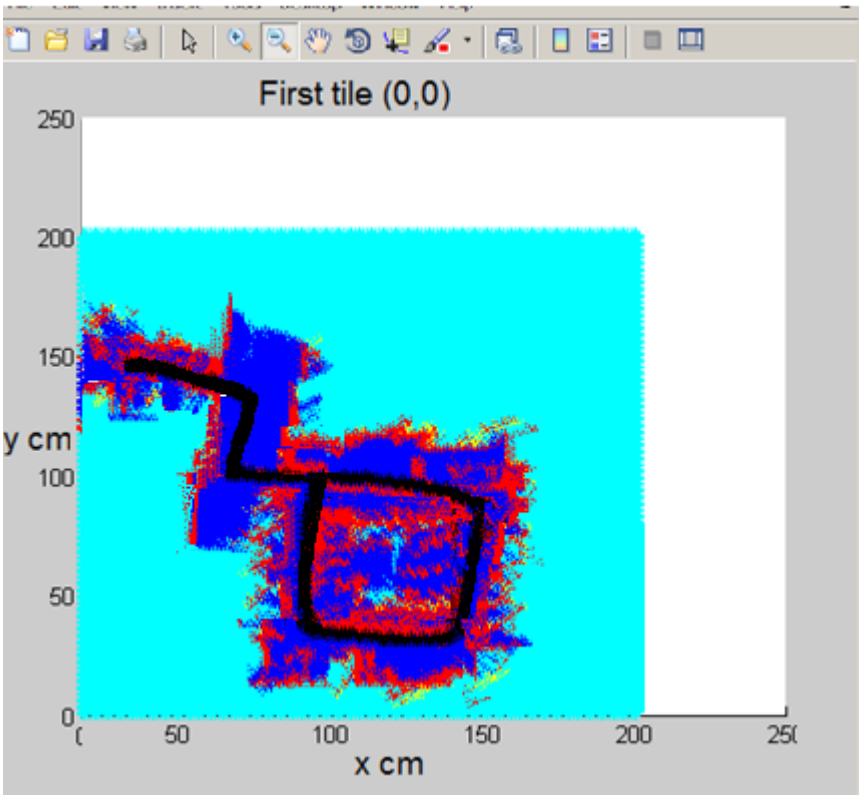


Figure 142: Grid map produced by the robot tile (0,0).

The blue areas indicate grid cells with a probability of occupancy lower than 0.5 while red cells are higher than 0.5 in terms of the probability of occupancy. Further, the cyan cells indicate cells that were not updated yet by the robot i.e. undetermined with a probability of 0.5 exactly since

any new tile is always initiated with 0.5 probability of occupancy. Black cells indicate robot path. The second tile is shown in figure 143 and is a completion of the map.

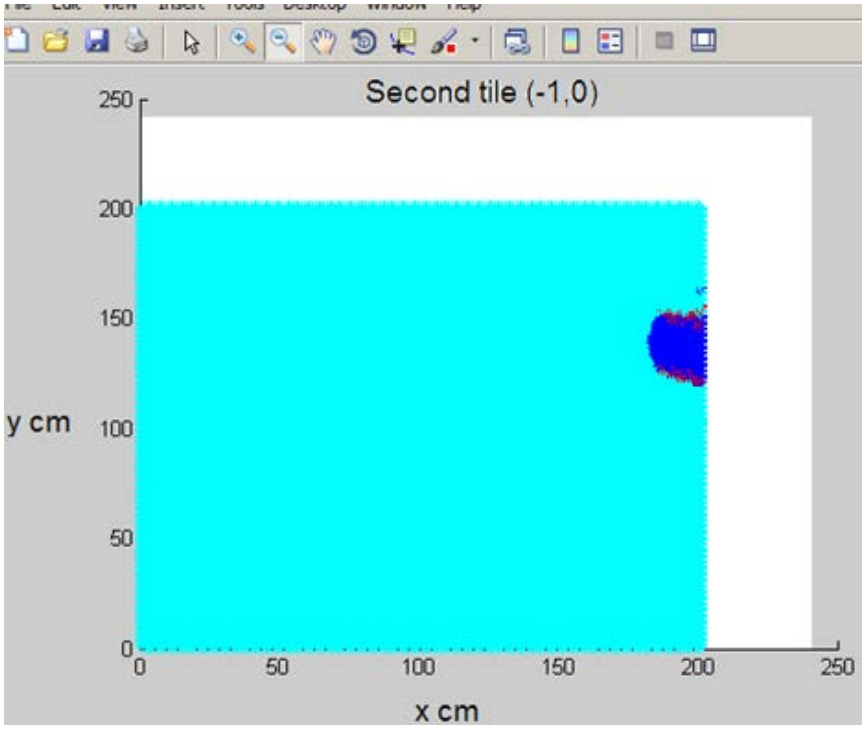


Figure 143: The second tile of the map (-1,0).

Note that the full map then becomes a combination of the two tiles as in figure 144. The map was combined manually as MATLB was unable to plot the big map without crashing.

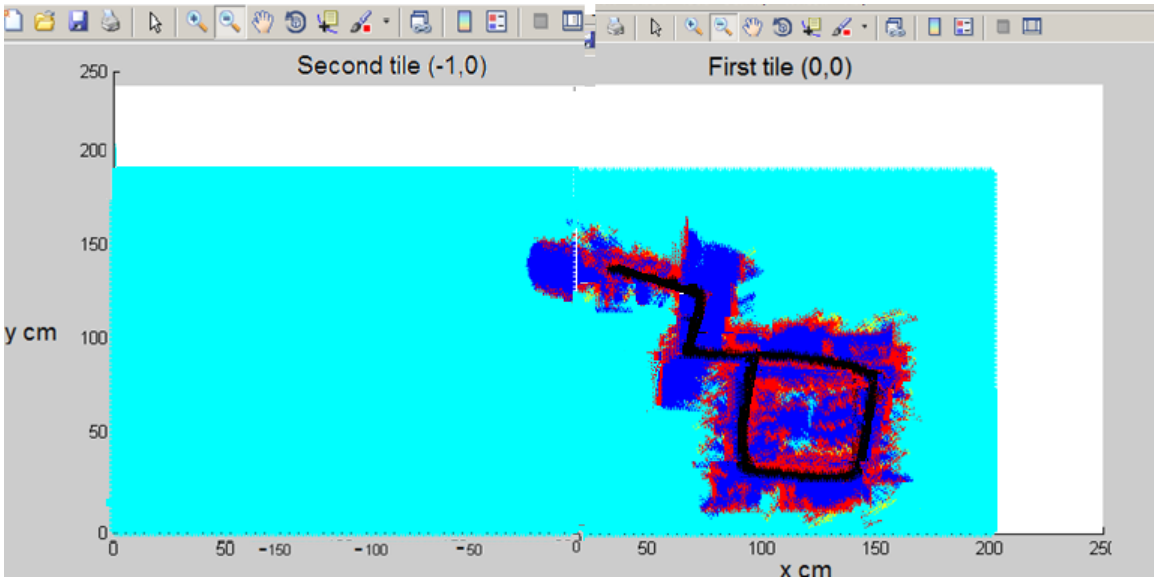


Figure 144: The full map formed by merging the two tiles from figure 142 and 143.

The resultant map is curvy and inaccurate since no UKF or scan matching was used and it was built using odometry only. This was only to proof the concept of local and global metric map fusion. The global metric map acts like a topological unit while the individual tiles act as nodes. In a later chapter, the same path will be followed by the robot to map the same area but using UKF to compare the two results. The outline of the actual lab and corridor environment is drawn in figure 145 (not to scale) for comparison with map produced by the robot.

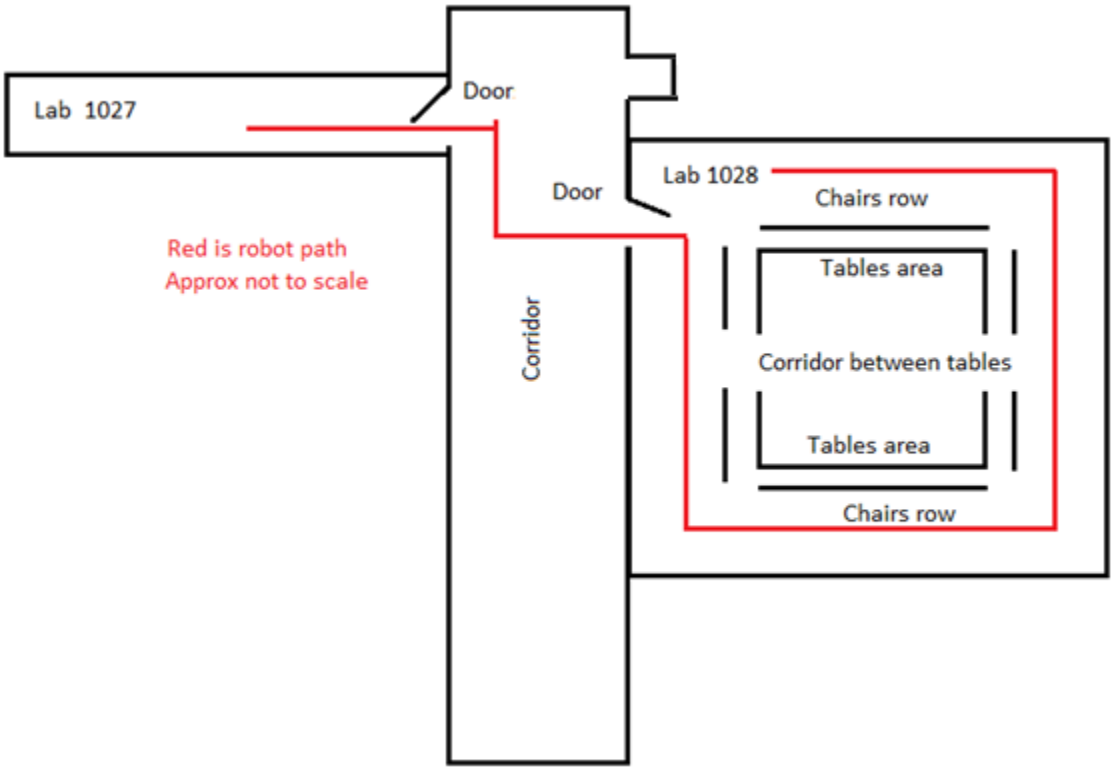


Figure 145: The approximate shape of the environment used.

Notice how the map is curved with the curving being more pronounced as the robot turns 90 degrees. The situation became severe in the 90 degree turn in the corridor which occurred due to a non-systematic odometry error occurring due to the robot bumping in the gap between ground tiles. This is all due to relying on odometry (dead reckoning). The map is fuzzy due to incorporation of sonar data which can be controlled by controlling the degree of fusion.

The resolution of the map and utilization of sonar data can be affected by the degree of fusion that occurs between laser and sonar data. This is achieved by putting more probabilistic weight on the data from one sensor compared to the other. In the implementation, the grid cell size

was chosen as 10 cm by 10 cm since this means the laser sensor model can be simplified to a model where cells containing the obstacles are full while cells on the way to the obstacles (penetrated by laser rays) are unoccupied. The size of grid cells can be made smaller which will necessitate the use of more elaborate laser model and will make the fusion of sonar data with laser less useful due to sensor resolution differences. However, it will lead to a more accurate map using one sensor i.e. laser.

4.6.5 Discussion

The sonar model developed to be used with the grid map was compared to the standard sonar model used. The developed model did much better than the standard model on time basis as it took on average 6 ms to run while the standard model took about 160 ms to run. It must be noted that those times are with a 200 by 200 cells grid map and will be different for different grid sizes and at a maximum processed range of 3000m. The maps show less details compared to the standard sonar model. However, they show that the developed model is more efficient on time basis. The standard model gives a richer map than the developed model but it is less accurate at the boundaries. The developed model on the other hand has less rich map but less erroneous at the boundaries. Further, the developed model is better used with a laser sensor to enhance its results.

4.7 Dynamic objects detection and extraction

4.7.1 Introduction

The aim of this section is to develop a technique that can be used to detect dynamic objects based on local frame to frame matching that helps to exclude the laser readings from dynamic objects that could lead to a wrong grid map building.

The process of dynamic object detection and extraction is made up of two parts. The first part is the dynamic object extraction process which relies on geometric features of expected dynamic objects to extract all possible objects that match the geometric criteria. The second step is the detection process which refines the output of the previous process by tracking the extracted objects and associating them with one another from frame to frame to confirm that they are dynamic. The second part of the algorithm can give rise to some inaccuracies if the dynamic object is slowly moving. This would not have been a problem in a stationary platform but with a

moving robot platform, and noisy sensor readings then slowly moving dynamic objects even if extracted correctly can be hidden in background noise originating from sensor noise.

The approach used here is similar to that used by [20] where experiments were conducted to find the geometric properties of dynamic objects. In an indoor environment, the expected dynamic objects are humans which can be detected as dips in the laser scan. Human legs appear as curves with certain depth, width, and number of scan points. Due to the fixed angular resolution, the number of scan points varies with distance from the laser sensor.

To characterize the properties of human legs, experiments were carried in a lab (indoors environment) with the robot being moved through by a joystick. The environment was populated with moving humans. The laser scans were saved as the robot moved and human legs were identified visually by a human and marked for logging. The visual appearance of human legs in laser scan in polar and Cartesian coordinates is shown in figure 146.

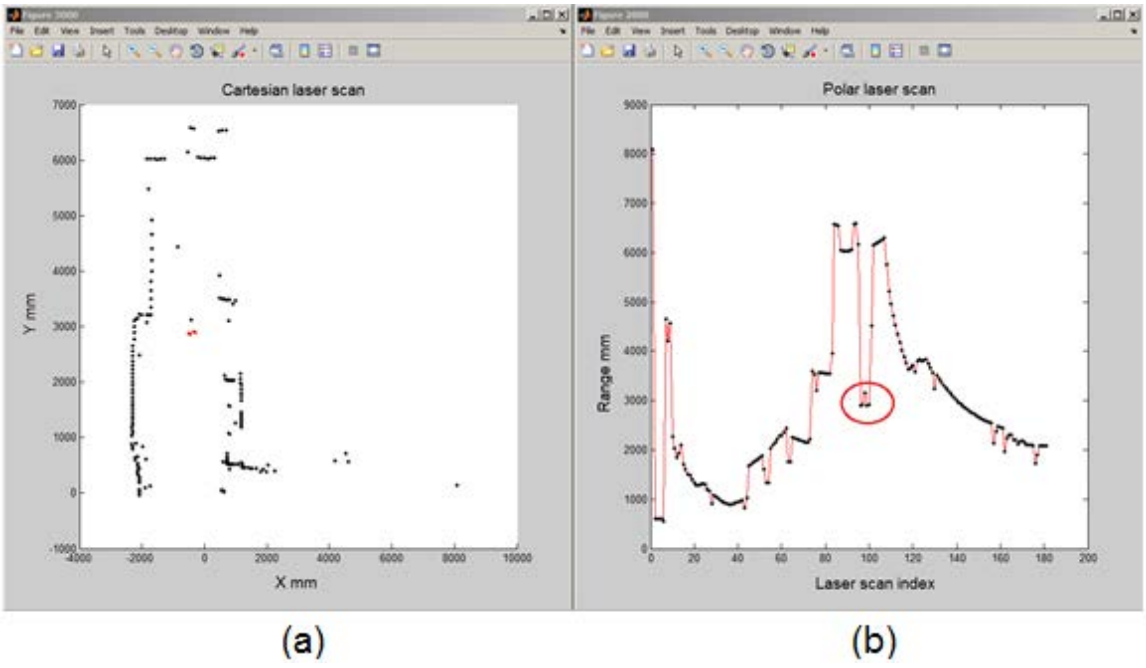


Figure 146: The appearance of human legs in Cartesian coordinates as red dots (a) and polar coordinates circled in (b).

The human leg in Cartesian coordinates is characterized by the properties shown in figure 147.

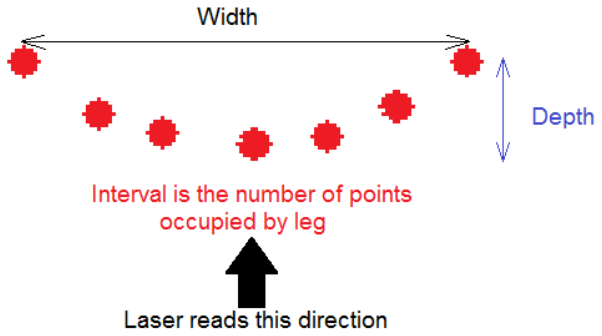


Figure 147: The properties used to differentiate human leg.

Those geometric features of the human leg can be used to better differentiate human legs from chairs. Experiments were conducted to find the properties of legs shown in figure 147 to use them in order to segment and extract possible dynamic objects. The results were plotted in figures 148, 149 and 150. The range (interval) of dynamic objects versus distance from laser sensor is shown in figure 148. As can be seen, there is an inverses relation with distance. However, the relation is non-linear best described by if then conditions.

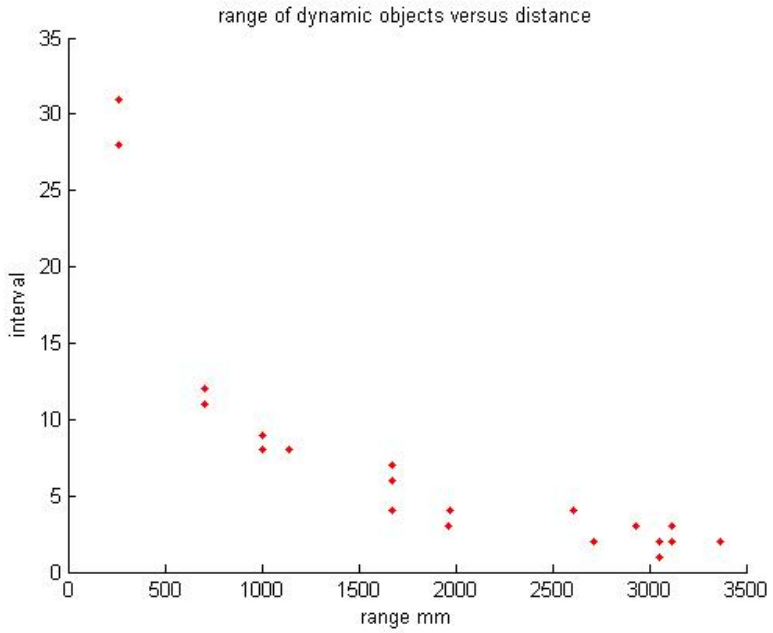


Figure 148: The range interval (laser points subtended) of dynamic objects versus their distance from laser sensor.

The length of dynamic objects versus distance from laser sensor is shown in figure 149.

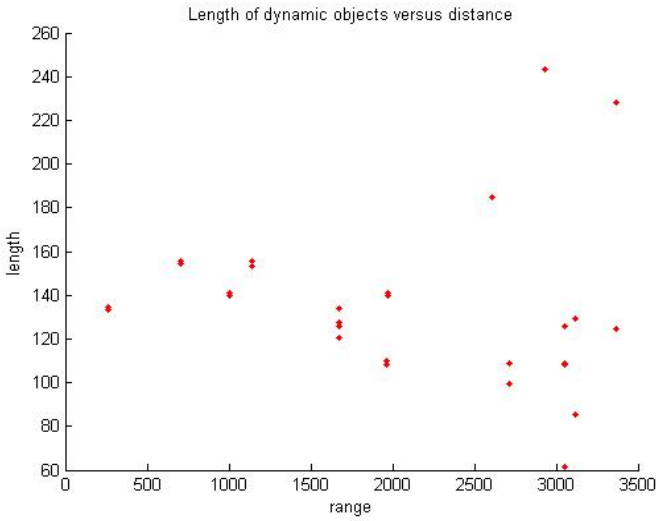


Figure 149: Variation of width of dynamic object with its distance from laser sensor.

No significant correlation was found between range and length as MATLAB function corrcoef was used to find the correlation between the two and the P value for hypothesis. The correlation was -0.08 and the P value was 0.6721 (more likely to be null hypothesis). The negative correlation with range is likely due to the decreased resolution in Cartesian coordinates due to fixed laser polar resolution.

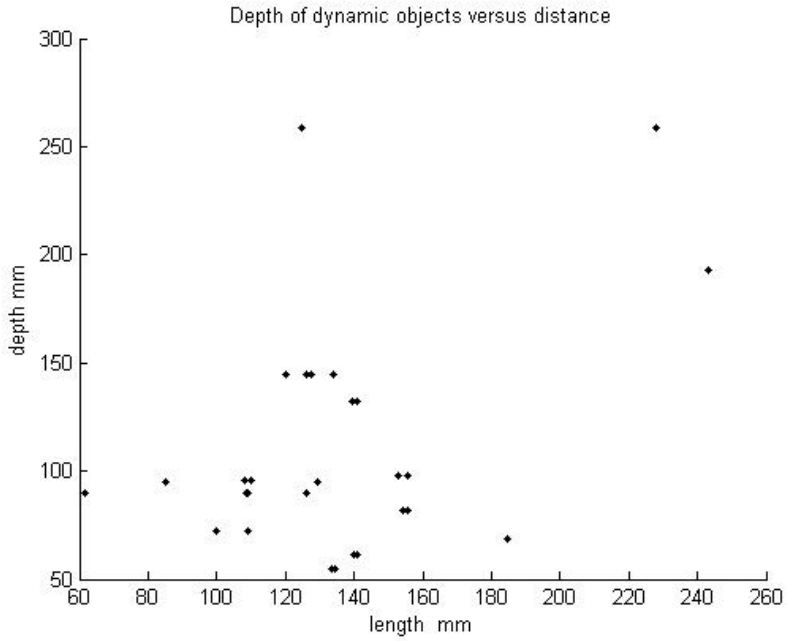


Figure 150: The map of depth versus length (width) of dynamic objects.

This shows that most legs have depth (refer to earlier geometry) of between 50mm and 180mm while the length varies between 80mm and 200mm. The larger values are due to two legs classified as one leg. Those values are similar to what [20] obtained. To help segment dynamic objects, a function was developed to predict the interval of dynamic object based on its distance from laser. The flow chart of this function is shown in figure 151. The find interval function is used to identify dynamic objects based on one of its properties. It will be called find_interv function.

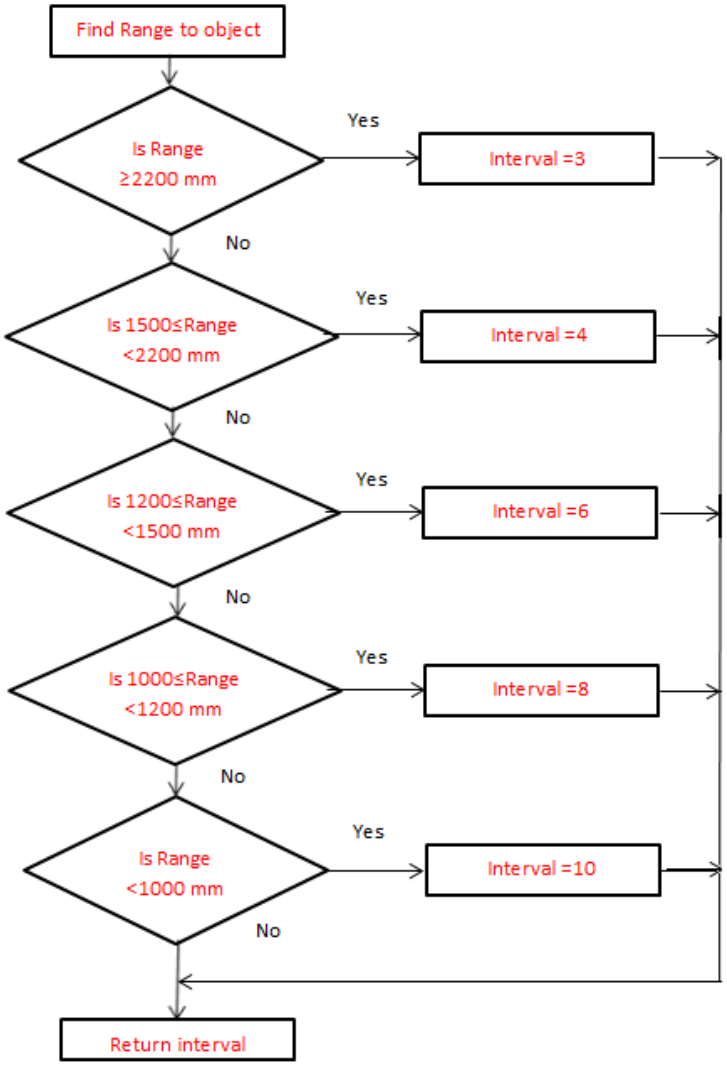


Figure 151: The flow chart for find interval function.

4.7.2. Algorithm for dynamic object detection and extraction

Dynamic objects are first detected based on geometric features then in the next step new dynamic objects are compared to old objects thus an association process is needed which is

what the function is called dyn_det and accepts two laser frames, fr and fr2, of dynamic objects for the current time step and the previous time step. The laser scan at the two time instants, frp and frp2, is acquired. The Two parts of the algorithm are executed according to the number of dynamic objects stored. When the number is zero the first part of the algorithm is executed once. If the number of objects falls again to zero this part will run again. The first step when there are zero objects is to store the current frames for detecting dynamic objects and comparing with later frames. The flow chart is in figure 152. Obj refers to the array that holds properties of current dynamic objects.

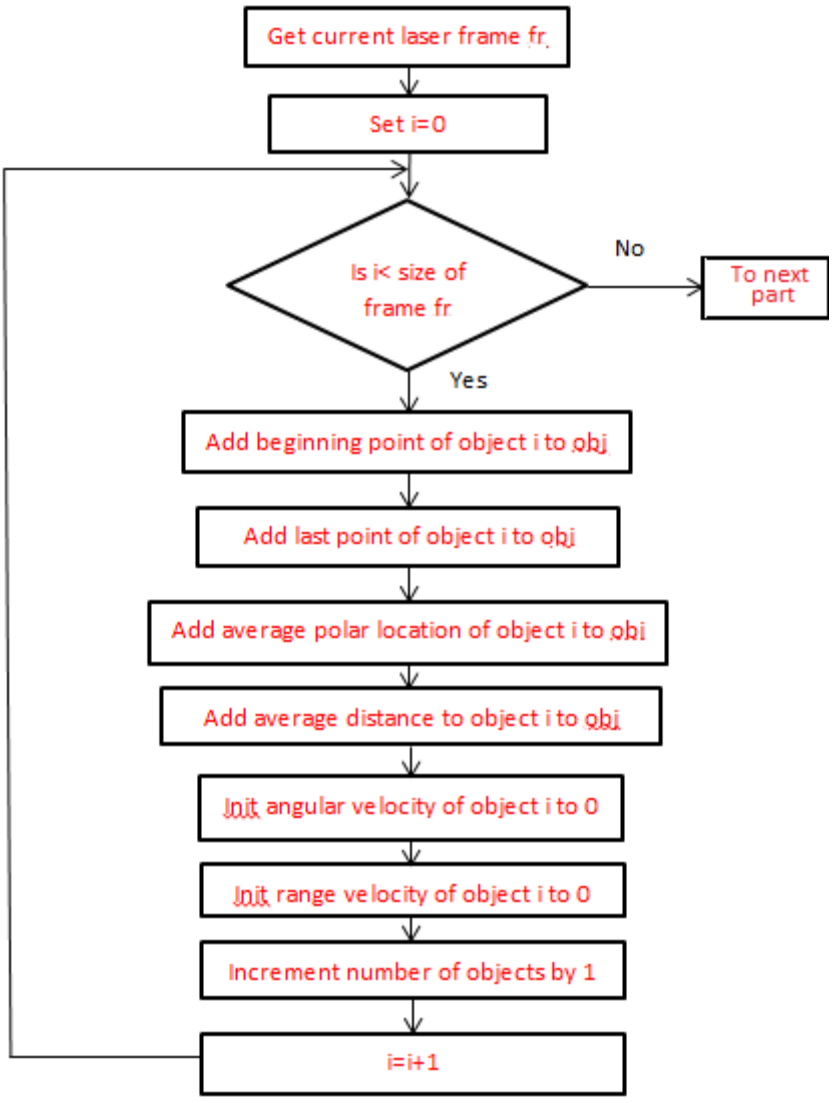
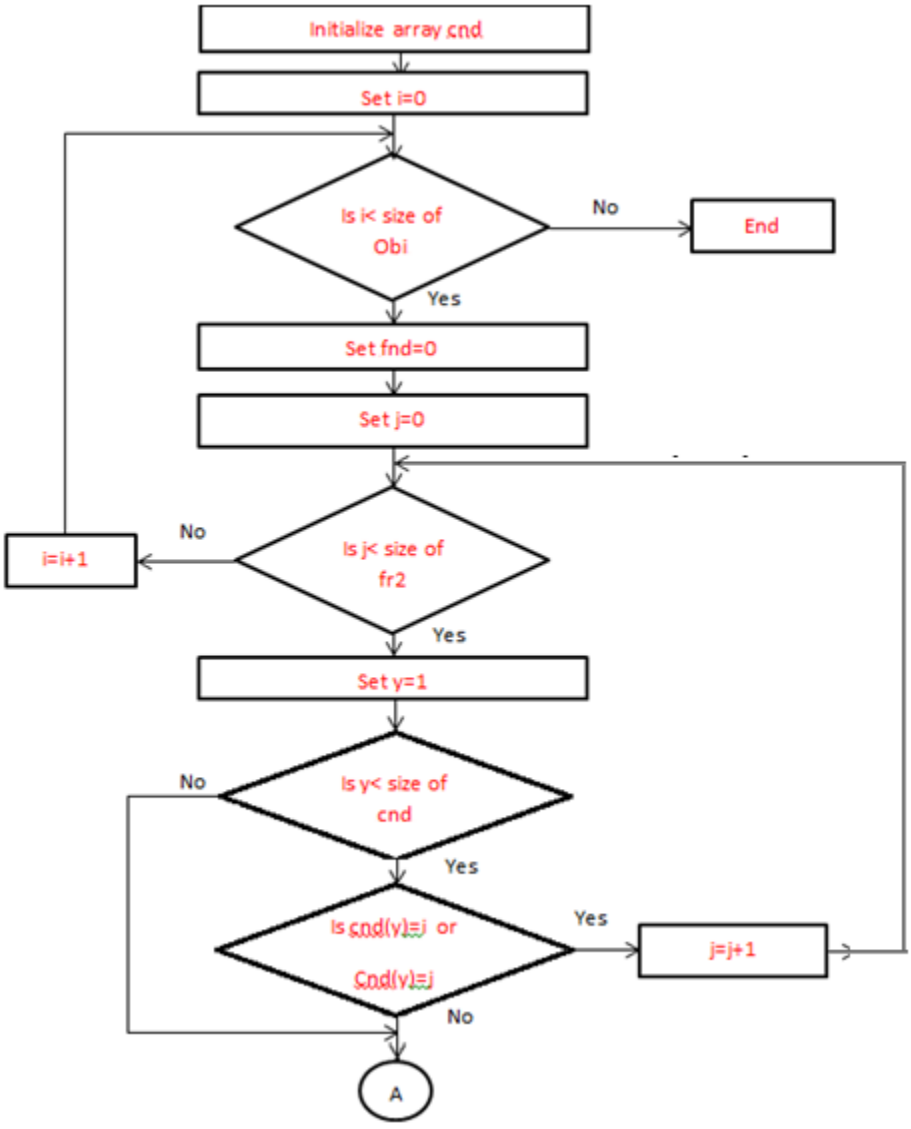


Figure 152: The first part of the dyn_det function.

In figure 152 the beginning points refers to the angle in the laser scan at which the first part of the dynamic object is detected starting from the default scan starting angle. The last point refers to the last point occupied by the dynamic object. The dynamic object is tracked by its polar and range velocities i.e. polar coordinates. However, when starting with no objects the velocities are initialized to zero. The second part of the function (figure 153) is executed when the number of dynamic objects is greater than 1. In this part, dynamic objects in frame two, fr2, and first frame, fr, are compared and the expected dynamic object pose change due to robot motion is calculated and used to associate old objects to objects in the new frame. If no match is found within the new frame for a dynamic object belonging to the previous frame, a null counter is incremented.



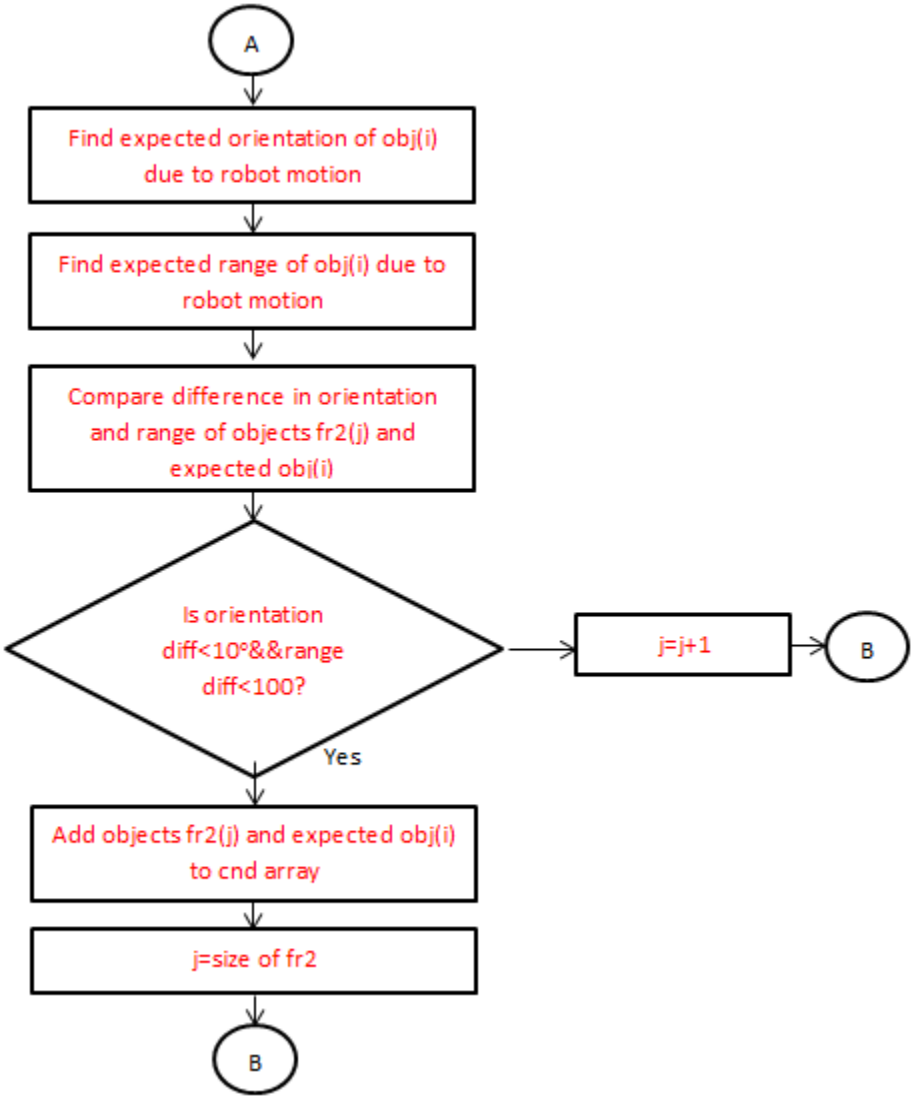


Figure 153: The flow chart for the second part of dyn_det function.

The expected change in object orientation due to robot motion is due to the change in the y distance to the dynamic object as the robot moves (assuming the direction the robot is heading to is the positive y axis). Thus,

$$\boldsymbol{th_n = \operatorname{atan}\left(\frac{y_p - dr}{x_p}\right)} \quad \textbf{(88)}$$

Where th_n is the new orientation due to robot motion that is expected of the dynamic objecty_p, and x_p is the position of the dynamic object, and dr is the distance moved by the robot. The change in object range due to robot motion is given by:

$$r_n = \sqrt{(y_p - dr)^2 + (x_p)^2} \quad (89)$$

Where r_n is the polar range. Following this, unmatched dynamic objects in the old frame are marked by incrementing their null counter and if the null counter exceeds 3 i.e. dynamic object not detected in three consecutive time steps, then the object is eliminated from dynamic object list. New unmatched objects are then added to obj array. The dyn_det function makes use of fr and fr2 arrays which hold the points of segmented dynamic objects. The segmentation of dynamic objects is carried out according to the geometric features of dynamic objects. The laser readings (181 readings) are segmented for dynamic objects expected from human legs according to the model built from experimental data. Any cluster of laser measurements satisfying the geometric features is tracked as a potential human. The dynamic object detection is carried out every 0.1 seconds approximately to extract dynamic objects and remove them from the grid map. The algorithm was tested on a P3-DX robot in a lab environment with humans (typical dynamic objects indoors) moving around. The experiments and results are displayed next.

4.7.3 Experiments

Experiments were carried out in a lab environment as shown in figure 154. The scattered points (circled) are typical points resulting from human legs and chair legs.

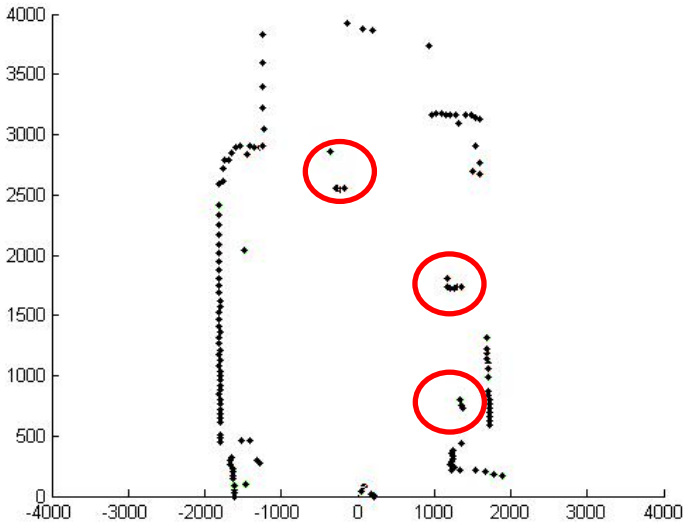


Figure 154: The lab environment with various objects.

However, some of the scattered points are chair legs due to their shape as shown in figure 155.

Figure 155: One chair shown on the left. Notice that the narrow part of the chair leg is what is visible on laser scan.

The robot was driven manually in the environment using the joystick and the robot continuously monitored its pose and used it to calculate dynamic object velocities. Further, the robot motion at certain instances could be very close to that of dynamic objects making it hard to estimate the true speed and classify it as a dynamic object. In section D the Cartesian laser scan will be presented frame by frame and the beginning points of dynamic objects with their endpoints are marked in red. The objects that are too small to be classified as human legs but suspected to be dynamic are in green.

4.7.4. Results

The initial scan is shown in figure 156.

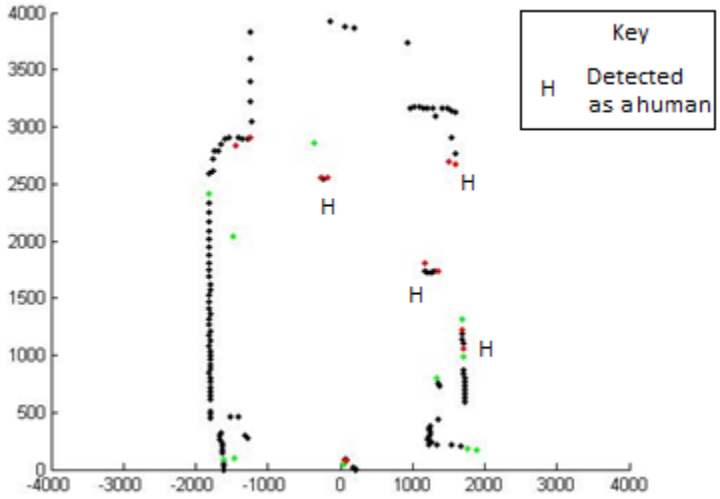


Figure 156: The initial segmentation of the environment.

The green points refer to segments too small to be of human legs but suspected as chair legs. The red points are the beginning points and end points of human legs detected. Note that the scale of the scan is in mm i.e. the size of the environment is of the order of 4m. As the robot moves every 0.1 seconds a new scan is processed and compared with the previous scan in order to measure the speed of dynamic objects (radial component and angular component). The second frame is shown in figure 157 (a) with the polar laser scan (b). Notice that the robot identifies some of the true objects as dynamic objects but identifies segments of wall/tables as dynamic objects. These are ghost objects and will be omitted as the robot discovers their speed does not change in next frame.

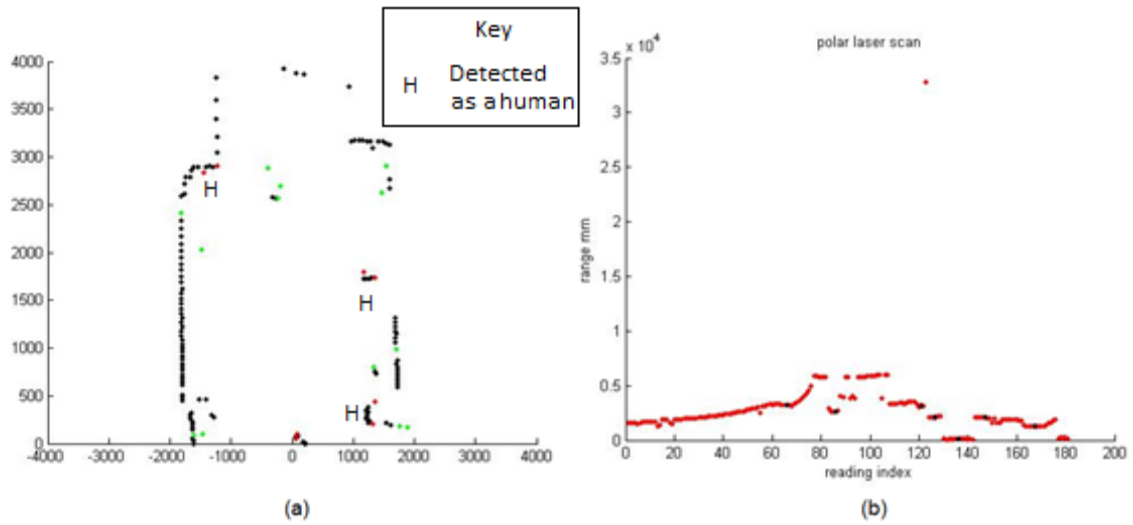


Figure 157: The laser scan at frame two(a) and corresponding polar scan (b).

The average location of the dynamic object on the polar scan is shown with black dots at the average index. The robot calculates the speeds of detected objects as shown in figure 158. The speeds for dynamic velocities are shown in pairs with the top value for radial velocity (mm/s) and the bottom value for angular velocity (deg/s). Notice that dynamic objects detected in the second frame only are initialized with zero velocity while those from frame one have a calculated velocity.

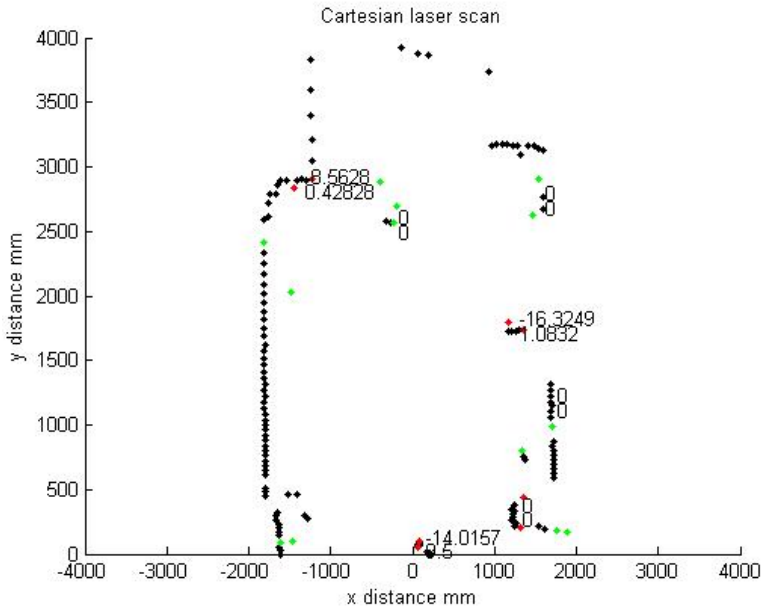


Figure 158: The angular and radial speeds relative to the robot of detected dynamic objects.

In frame 3 in figure 159, the objects detected in frame 2 are assigned speeds as shown.

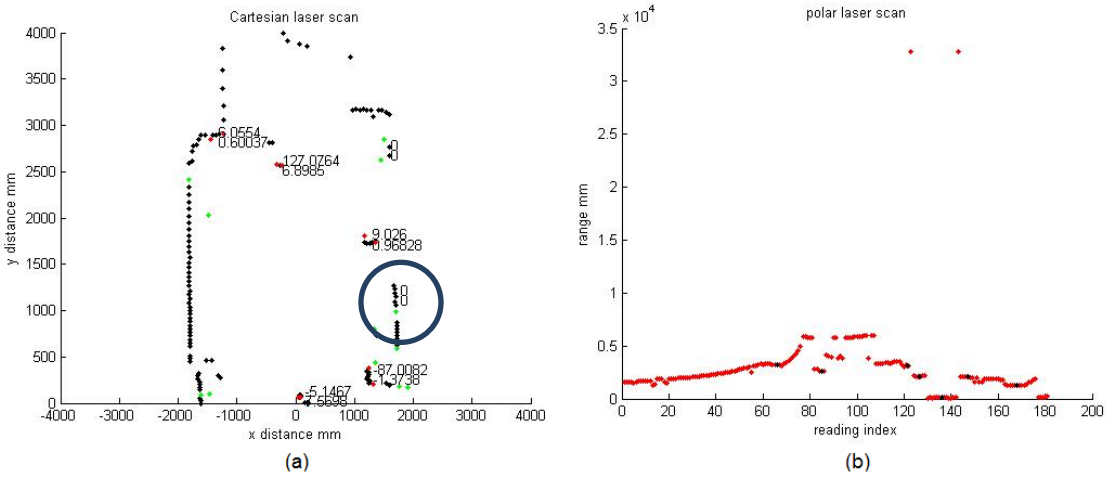


Figure 159: The third frame (a) and the corresponding polar plot (b).

Note that the calculated velocity is the current velocity not the average and is based on the motion from frame to frame thus may not reflect true velocity of object since humans move with gaits and swing their legs at a speed different from average body speed. Notice that the robot mistakenly labels a wall segment (shown in blue circle) as dynamic object and assigns zero speed to it. This is will be dropped from the list of dynamic objects as the robot moves on. This is

shown in figure 160 which shows the fourth frame where the labeled wall segment is no longer in dynamic objects list.

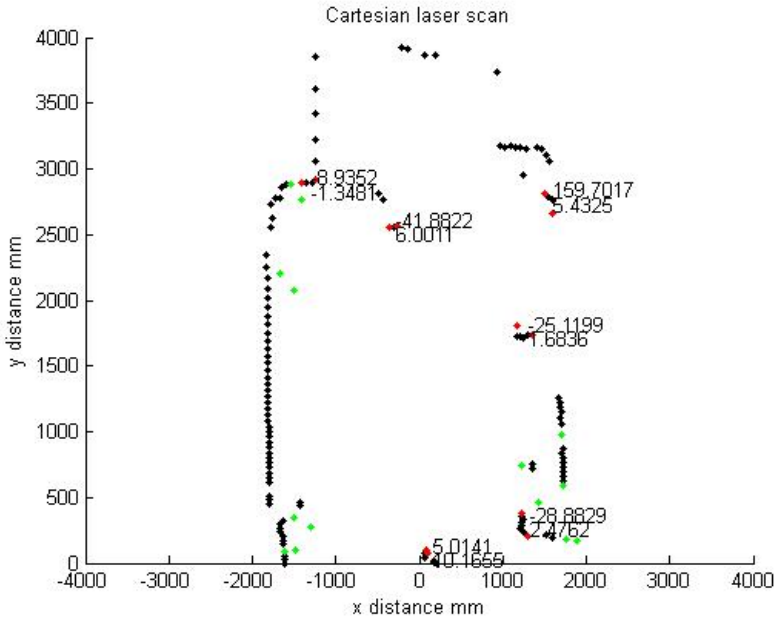


Figure 160: The cartesian frame for frame 4. Notice that the wall segment previously labeled as dynamic no longer exits in object list.

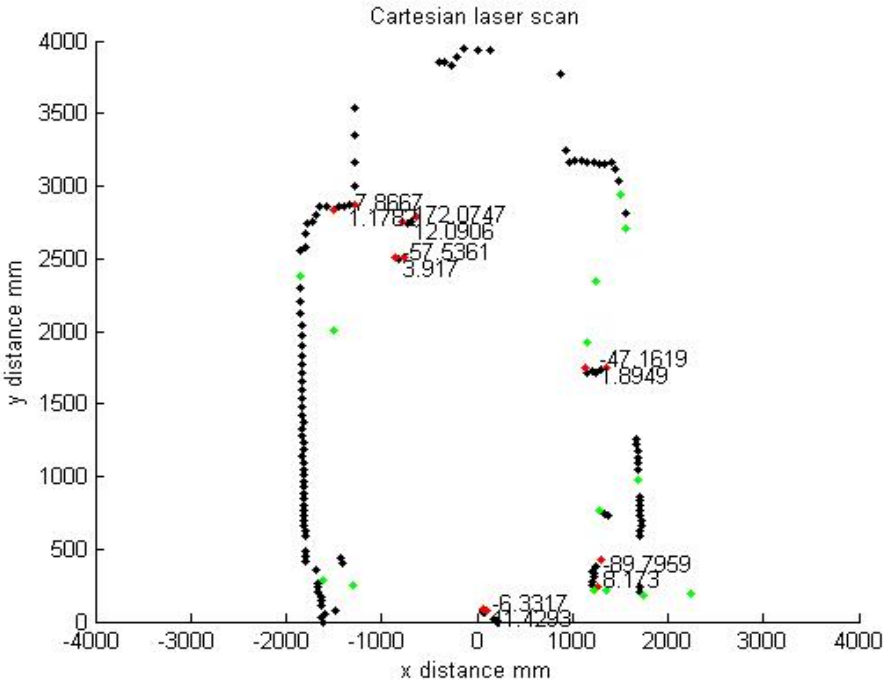


Figure 161:The situation at frame 10.

The situation continues at frame 10. Here the robot has detected the two legs of one human directly in front at about 0 degree heading. Notice that wall segments that were previously classified as dynamic objects no longer exist in list. The same effect repeats itself again this can be seen in the situation from frame 20 to 22 as shown in figure 162. A wall segment (circled) was labeled as dynamic but later excluded.

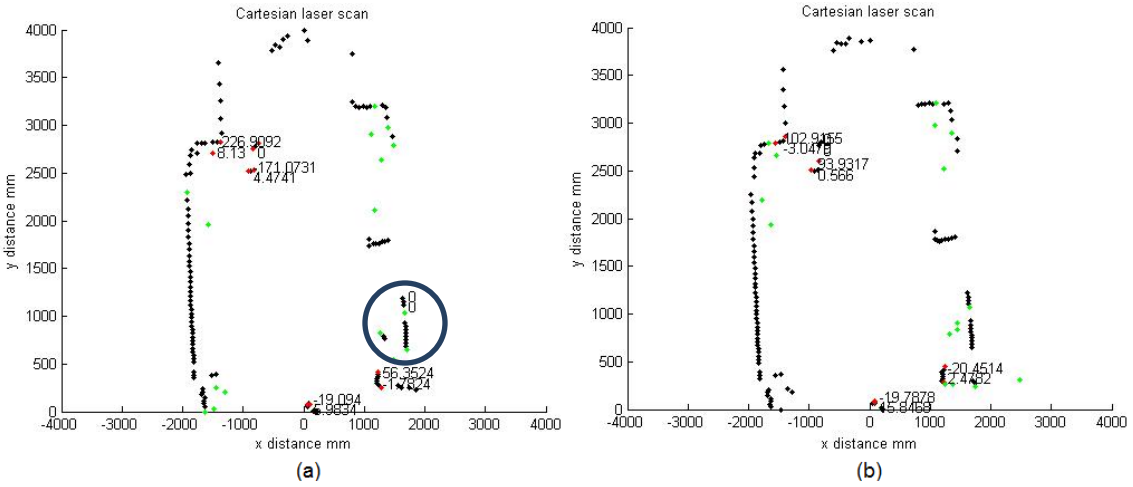


Figure 162: The situation of dynamic objects from frame 20 in (a) to 22 in (b).

Sometimes a wall segment with one off shooting laser point can be classified as dynamic object and a ghost speed is calculated which is due to noise in laser readings coupled with robot motion. However, as the robot moves and views it from different positions it gets less likely it is segmented and included as dynamic in next frames thus eventually falling from list. This evolution is shown in figure 163 where it was detected by mistake as dynamic (a) then its speed calculated for tracking (b) and later excluded. This happened from frame 33 to 35.

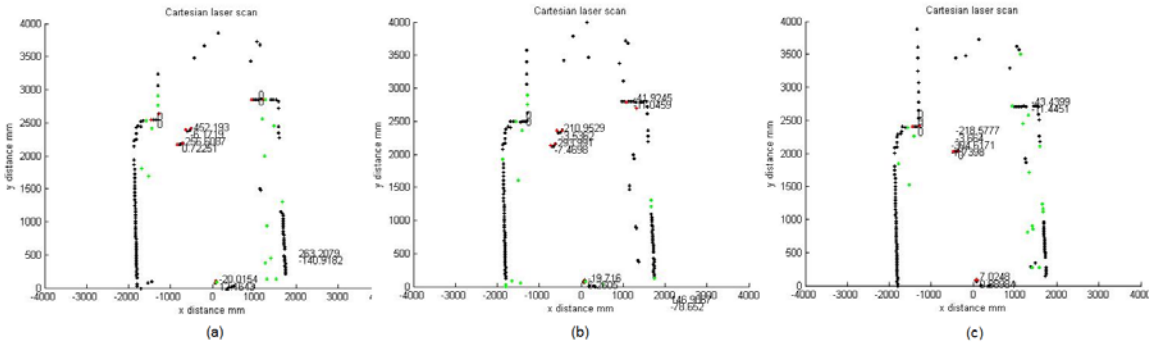


Figure 163: The evolution of ghost dynamic object detection (a) tracking (b) and exclusion (c).

Another observation seen was that a dynamic object correctly detected in one frame may move too much as to be labeled as a new object in next frame and have its previous position tracked as a ghost in next frame. An example of this happened in frames 41 to 42 where the two legs tracked in frame 41 were later identified in frame 42 with one leg labeled as new. Notice that the left leg had zero speed when this happened. The explanation is that this happened while the person was stepping on the left leg and raising the right leg up and fast forward to take a step forward which could explain why this happened.

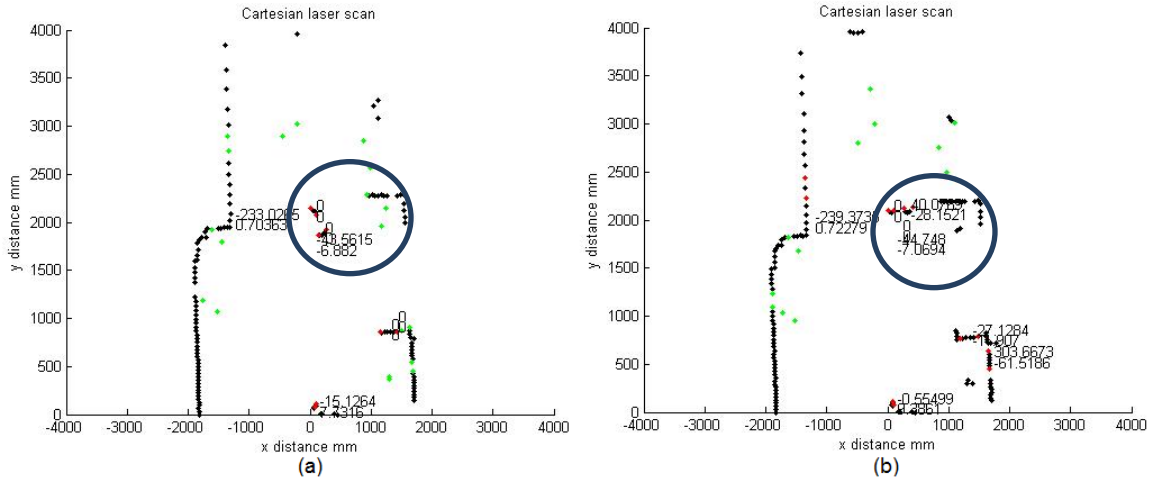


Figure 164: Two legs in frame 41 (a) were identified in next frame (b) with one leg labelled as new and its previous position tracked as a ghost object.

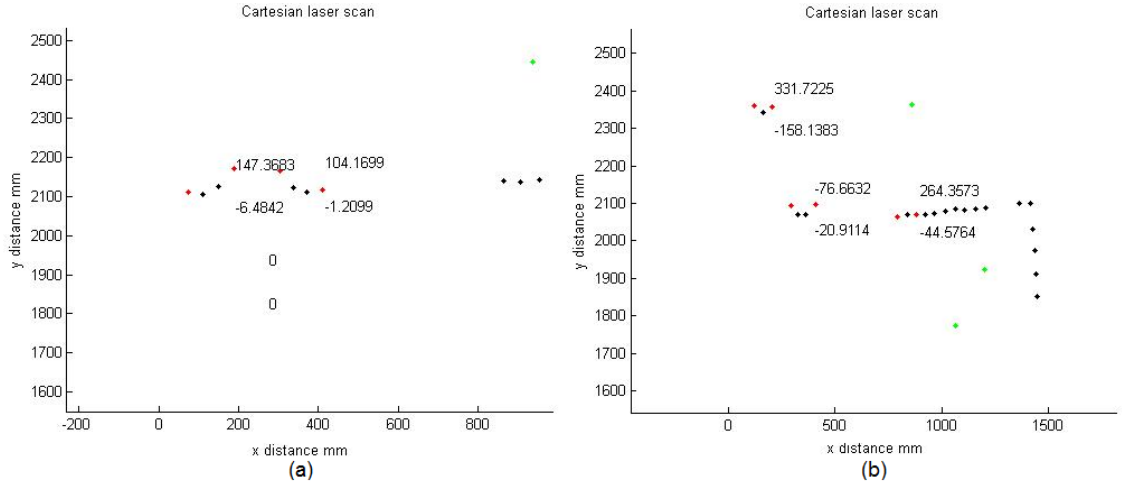


Figure 165: Zoomed in view of the two legs at frame 43(a) and frame 44(b).

A zoomed view is shown of the next two frames which show that the ghost feature was tracked for one frame (43) until it was dropped in frame (44).

4.7.5 Discussion

The dynamic object detection function succeeded in segmenting dynamic objects based on their geometric profile. However, sometimes some wall segments appear as dynamic objects and are added to the list. In those cases the algorithm may succeed in excluding those objects based on two ways. The first way is the geometric approach where the same objected detected from different pose may not satisfy the geometric criteria and thus is excluded from the list. The second way is because those objects are stationary, when tracked with their original speed will not satisfy the expected location and will be excluded such as the case demonstrated in figure 163.

Further, sometimes dynamic object speeds may not reflect the true speed of the dynamic object since the objects under consideration are humans who move with gaits by fixing one leg and moving the other forward to take a step then alternating the motion. Since the laser sensor only detects legs it will not see the true motion of the human (average speed) and may show varying speeds from frame to frame which may occasionally lead to the dynamic object being lost out of tracking. A suggestion is to develop a more involved model that can correlate leg motions according to different gait models and group legs into human groups then based on the gait model predict the human speed. However, this is challenging since gaits and their velocity tend to vary from person to person. Therefore, a better approach could be the use of statistical model that yields an average speed expected from observed gait.

Moreover, if the human leg moves forward at high speed, it may mistakenly be marked as a new dynamic object and its previous position tracked as a ghost for some time. However, it will eventually be dropped from the tracking list of dynamic objects as the null counter counts it unseen for several frames (3).

To overcome those problems, the geometric criteria for dynamic object segmentation could be made stricter by taking more readings to extract tighter tolerances for geometric properties of dynamic objects. However, in this case there is a risk that the data could be environment specific. For the ghost dynamic objects, the threshold of number of times unseen may be decreased but will lead to poorer tracking of dynamic objects due to speed variability.

Another approach is to use average speed for tracking or build a motion model and use Kalman filter for tracking. However, this may not be robust with humans since humans show abrupt

motion change in normal day to day activity thus average speed may not reflect a standing still human and model based tracking methods at best will succeed in constant speed or acceleration situations.

Another approach is to use a global grid map for dynamic objects that is updated locally only for time efficiency. However, this will be slower than the current approach but will certainly be more accurate. It will need better maintenance of robot pose since unlike the local method developed here; it relies on accurate robot pose. Thus the choice is a compromise between different things and this method was developed to be fused with grid mapping for ruling out laser readings originating from dynamic objects. Further analysis and development will be needed to compare global methods to local methods.

4.8 Camera sensor

4.8.1 Introduction

The camera was used in this experimental work to investigate the possibility of it as a sensor that could be fused with the robot laser and sonars. The algorithms for image processing were obtained from OpenCV and the approach was mainly focused on real time processes. Thus only optical flow and image enhancement algorithms were used. More in depth image processing like SIFT will enable more information fusion but at the cost of time so were avoided. This section will start by explaining optical flow since it was the algorithm mainly used.

i. Optical flow from cameras

Optical flow is caused by both camera rotation and translation. It is difficult to fully correlate the effects of the flow to each component as every point in the image has different depth and hence affects the optical flow differently [67]. Since optical flow is caused by the 3D motion of the camera, then a relation between camera 3D motion and the expected 2D image motion field is required. For this consider the geometry in figure 166.

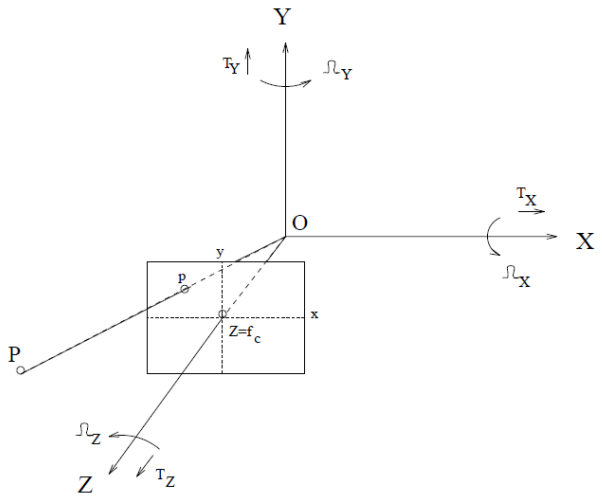


Figure 166: The camera geometry [67].

The global coordinate system XYZ is assumed to have its origin at the camera lens, and the image plane is located parallel to the XY plane at a depth, Z, equivalent to the camera focal length, f_c. The following analysis is obtained from [67]. In figure 166 the point P has real world coordinates (X,Y,Z) which is projected on the image plane as point p with coordinates (x,y). This is expressed mathematically as:

$$\boldsymbol{p} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} = \begin{bmatrix} \frac{\boldsymbol{X}}{\boldsymbol{Z}} \boldsymbol{f_c} \\ \frac{\boldsymbol{Y}}{\boldsymbol{Z}} \boldsymbol{f_c} \end{bmatrix} \quad \textbf{(90)}$$

The camera ego motion has two components, a translational and a rotational component which are given by:

$$\boldsymbol{T} = \left(\boldsymbol{T_x}, \boldsymbol{T_y}, \boldsymbol{T_z}\right)^{\boldsymbol{T}} \qquad \boldsymbol{\omega} = \left(\boldsymbol{\omega_x}, \boldsymbol{\omega_y}, \boldsymbol{\omega_z}\right)^{\boldsymbol{T}} \quad \textbf{(91)}$$

Assuming the camera moves a translation T and a rotation ω, the point P will appear to move to a new location P' which is translated by –T and rotated by – ω thus:

$$\boldsymbol{P'} = \boldsymbol{M_{-\omega}} \boldsymbol{P} - \boldsymbol{T} \quad \textbf{(92)}$$

Where $M_{-\omega}$ is the rotation matrix by – ω. If the small angle approximation is used, the rotation matrix can be simplified to:

$$\boldsymbol{M_\omega} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{-\omega} & \boldsymbol{\omega} \\ \boldsymbol{\omega} & \boldsymbol{0} & \boldsymbol{-\omega} \\ \boldsymbol{-\omega} & \boldsymbol{\omega} & \boldsymbol{0} \end{bmatrix} \quad \textbf{(93)}$$

From this it is possible to get a relation for the expected optical flow in x and y directions on the image plane thus from [67]:

$$\begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix} = \begin{bmatrix} -\boldsymbol{f_c}\left(\frac{\boldsymbol{T_x}}{\boldsymbol{Z}} + \boldsymbol{\omega_y}\right) + \frac{\boldsymbol{xT_z}}{\boldsymbol{Z}} + \boldsymbol{y\omega_z} - \frac{\boldsymbol{x^2\omega_y}}{\boldsymbol{f_c}} + \frac{\boldsymbol{xy\omega_x}}{\boldsymbol{f_c}} \\ -\boldsymbol{f_c}\left(\frac{\boldsymbol{T_y}}{\boldsymbol{Z}} + \boldsymbol{\omega_x}\right) - \boldsymbol{x\omega_z} + \frac{\boldsymbol{yT_z}}{\boldsymbol{Z}} - \frac{\boldsymbol{xy\omega_y}}{\boldsymbol{f_c}} + \frac{\boldsymbol{y^2\omega_x}}{\boldsymbol{f_c}} \end{bmatrix} \quad \textbf{(94)}$$

Equation 94 can be simplified to give:

$$\begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix} = \begin{bmatrix} \frac{\boldsymbol{T_z x - T_x f}}{\boldsymbol{Z}} - \boldsymbol{\omega_y f} + \boldsymbol{\omega_z y} + \frac{\boldsymbol{\omega_x xy}}{\boldsymbol{f}} - \frac{\boldsymbol{\omega_y x^2}}{\boldsymbol{f}} \\ \frac{\boldsymbol{T_z y - T_y f}}{\boldsymbol{Z}} + \boldsymbol{\omega_x f} - \boldsymbol{\omega_z x} - \frac{\boldsymbol{\omega_y xy}}{\boldsymbol{f}} + \frac{\boldsymbol{\omega_y y^2}}{\boldsymbol{f}} \end{bmatrix} \quad \textbf{(95)}$$

From equation 95 it is evident that the magnitude of optical flow depends on the pixel location. Thus for a combination of translations at zero rotation there exists a point (x,y) where the optical flow vector (u,v) is (0,0). This point is called the focus of expansion (FOE).

Further, notice from equation 95that the rotational component is depth independent while the translational component depends on the depth. Therefore, if an assumption of no rotation is made, then it is possible to estimate the range from optical flow [68].

ii. Focus of expansion

In the case of pure translation, the optical flow vectors due to camera motion appear to originate from one point known as the focus of expansion (FOE) [41]. At the focus of expansion there is no optical flow and its position corresponds to the heading of the camera. This is the case only for forward camera translation and no dynamic objects. Figure 167 shows the optical flow for pure translation (a), with a dynamic object (circled) in the bottom right corner (b), and with rotation of the camera (c) [44].

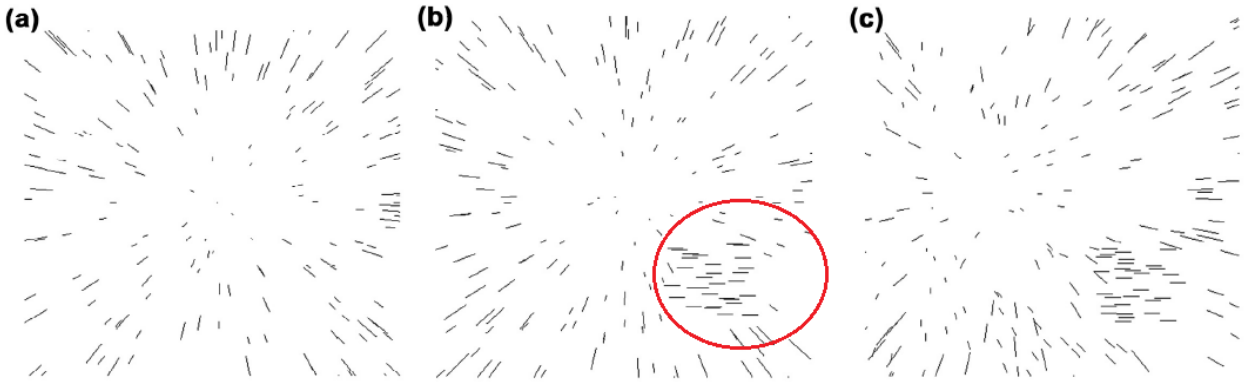


Figure 167: Optical flow for translation only (a), with dynamic object (b) and with rotation of camera in addition to translation (c) [44].

The focus of expansion is important for three dimensional reconstruction of the image, range estimation, time to collision calculation, and obstacle avoidance. Thus it is very important for robotic systems employing vision [45, 46]. There are two main groups of techniques to estimate the FOE location which are local and global methods. Local methods work by comparing corresponding points or lines and work on a local scale in the image making them not robust to noise. Global methods rely on the whole optical flow field to estimate the FOE but are computationally expensive since they usually involve the use of least squares estimate [46].

iii. Neural networks

Neural networks will be used for the estimation of the rotational component of ego motion of the robot. Neural networks are parallel computational units that have the ability to learn from examples to find the model of an unknown system [69]. Commonly used networks are multi-layer feed forward perceptron consisting of input layer, hidden layer that implements a function (usually non-linear), and output layer to scale outputs [69].

Mathematically each neuron, j, in the hidden layer performs the following mathematical operation on its inputs x_i [69]:

$$y_j = f\left(\sum_{i=1}^{n} w_{ji} x_i\right) + b_j \quad (96)$$

Where f is the function performed by the neuron, w_{ji} is the weight connecting neuron j to input I, and x_i is the ith input. Neural networks are commonly trained using back propagation to gradually adjust network weights and biases. MATLAB "train" function was used for this purpose.

4.8.2 Methodology

i. Setup used and experiments

A P3DX robot was used which was fitted with a mono camera as shown in figure 168.

Figure 168: The setup used.

The camera used was a monocamera from Point Grey model number BFLY-U3-03S2C-CS. The camera has 1 megapixel 1/3 inch image format which has the following properties from [70]: Sensor width of 4.8mm, height of 3.6mm and a 6mm diagonal as shown in figure 169.

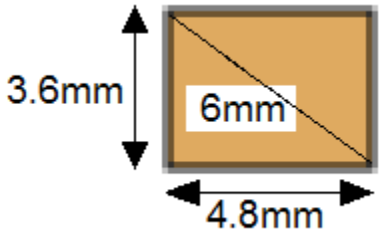


Figure 169: The dimensions of the 1/3'' sensor of the camera [70].

The camera was fitted with DF6HA-1B lens with a focal length of 6mm. Since the differential P3DX robot cannot rotate around the z and y axes and cannot move along the x and y axes the optical flow is related to the motion field by:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \frac{T_z x}{Z} + \frac{\omega_x xy}{f} \\ \frac{T_z y}{Z} + \omega_x f \end{bmatrix} \quad (97)$$

To make use of optical flow, the optical flow due to ego rotation has to be removed first. OpenCV was used to implement the optical flow algorithm. The algorithm used was the sparse Lucas-Kanade method with pyramids implemented by OpenCV.

The first step is trying to remove rotation effect. To do this, experiments were carried out to determine the actual flow due to pure robot rotation and the results were compared to the theoretical case of equation 97. The pixel size is 0.0075mm and the focal length is 6mm so it is possible to predict the flow resulting from certain rotation. An example of an expected optical flow pattern at 3 degree rotation is shown in figure170.

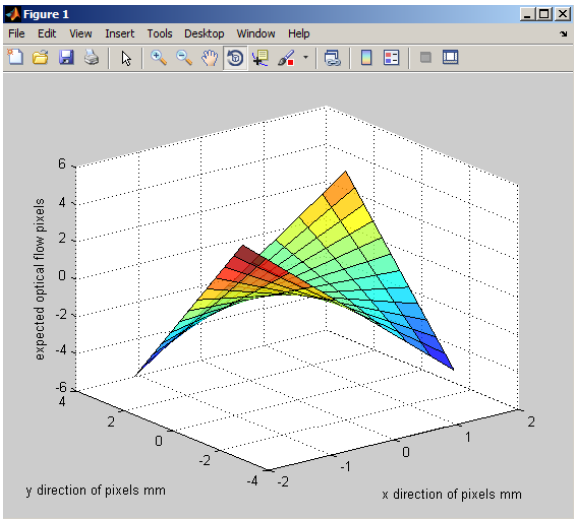


Figure 170: The predicted optical flow at 3 degree rotation.

The flow is saddle shaped and peaks at the edges of the image and is zero at the center. This is in accordance with the theory. Now this is was compared to the actual optical flow obtained from OpenCV as the robot rotated three degrees and captured two consecutive frames for comparison. The result is plotted in figure 171. The results are very noisy but fall within 21 to 25 pixels. This is very different from prediction from theory.

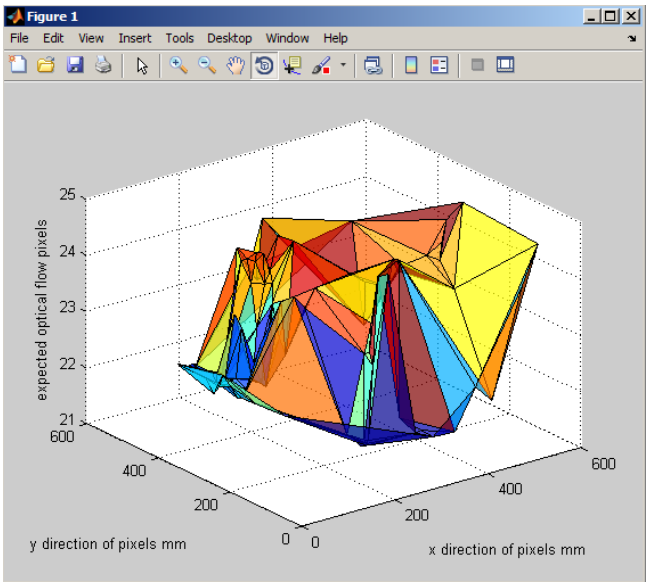


Figure 171: The actual optical flow obtained at 3 degree rotation.

The actual image is shown in figure 172. The circles represent the initial pixel locations. While lines indicate the direction of optical flow and their length corresponds to magnitude.

Figure 172: The actual scene from which the optical flow was taken at 3 degree rotation.

To understand why the optical flow at three degree rotation deviated from the theoretical optical flow, the optical flow was calculated theoretically at negative three degrees and compared to actual negative three degree measurements.

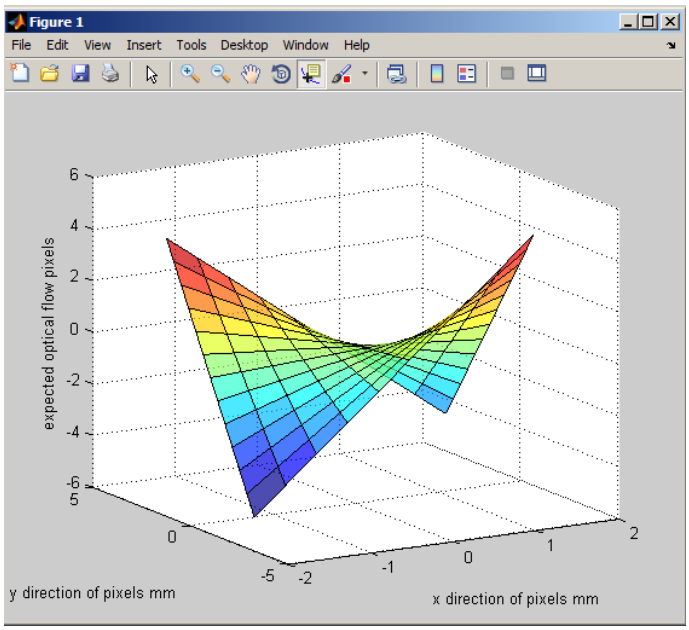


Figure 173: The predicted optical flow at -3 degree rotation.

The flow is saddle shaped, opposite to what is obtained at figure 171 and peaks at the edges of the image and is zero at the center. Now this is was compared to the actual optical flow obtained from OpenCV as the robot rotated negative three degrees and captured two consecutive frames for comparison. The result is plotted in figure 174. The results are very noisy but fall within -37 to -42 pixels. This is very different from prediction from theory.

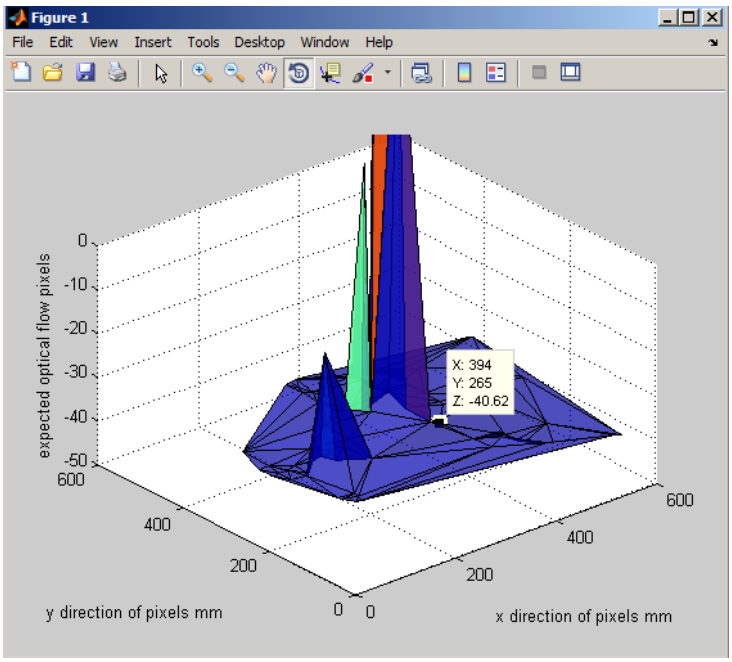


Figure 174: The actual optical flow obtained at -3 degree rotation.

The actual image is shown in figure 175. The circles represent the initial pixel locations. While lines indicate the direction of optical flow and their length corresponds to magnitude.

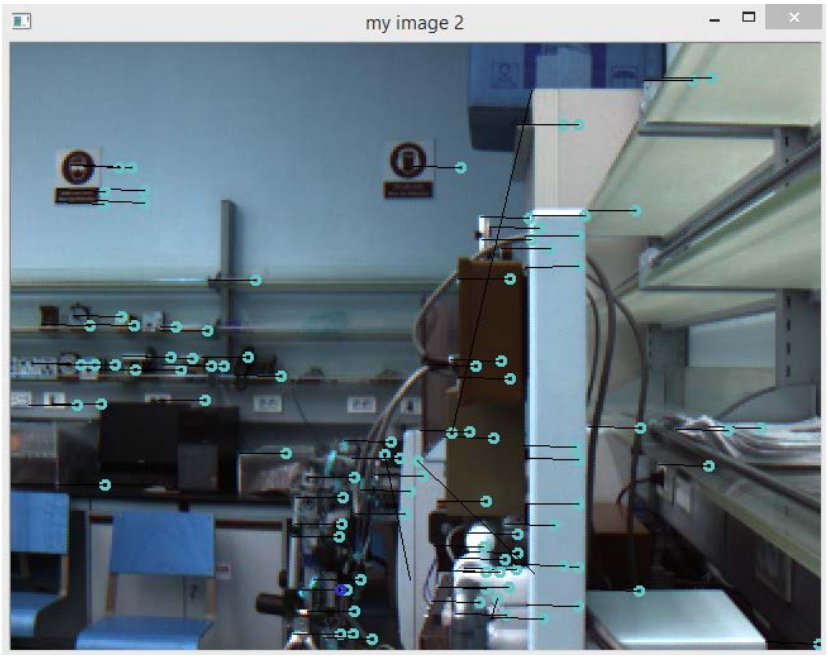


Figure 175: The actual scene from which the optical flow was taken at -3 degree rotation.

Notice how the vectors indicate right to left flow movement reverse to that in figure 172. Following the comparison between prediction and actual data, it was concluded that the actual motions experienced at the camera lens were different from those at the robot center. Referring to figure 168 it can be seen that the lens of the camera is not at the center of the robot rather it is displaced at a certain distance and orientation. Therefore, equation 97 is not directly valid rather the actual motion at the camera is given by:

$$\boldsymbol{P_c} = {}^{\boldsymbol{R}}\boldsymbol{T_c}\boldsymbol{P_R} \quad \textbf{(98)}$$

Where ${}^{R}T_c$ is the transformation matrix, P_R is the robot position, and P_c is the camera position. However, this transformation is difficult to figure out accurately due to many reasons such as the difficulty measuring the distance between camera lens and the robot center accurately, the need to account for motion of PTU unit on which camera is mounted, and the difficulty of measuring camera orientation accurately compared to robot center.

Therefore, the robot rotation induces the effect of both rotation and translation at the camera lens which is responsible for the discrepancy between actual and theoretical results. Although

the relation is not known, it can be taught to a neural network via the use of large set of training data to extract the relation between the optical flow and robot rotation. Furthermore, since the extraction of optical flow is noisy and creates some outliers then the use of neural network will make the algorithm robust to outliers.

ii. Neural network training

Therefore, a neural network was developed and trained to predict the optical flow at any pixel location in the x and y directions due to the robot motion. The neural network used is the non-linear feed forward neural network with tan sigmoid activation function. The network accepts P vector of inputs of R variables and has n neurons connected to inputs by weight matrix of size n x R. Each neuron has a bias and sums its output to a transfer function. Using trial and error, it was found that a neural network of 5 hidden layers employing tan sigmoid transfer function is sufficient to approximate the optical flow from training data without risking over fitting.

The layout of the neural network is shown in figure 176. The neural network accepts three variables at the input:

- Y ordinate of pixel with respect to image center
- X ordinate of pixel with respect to image center
- Rotation angle of the robot

The first two variables are divided by 100 to scale the inputs for easier network training. At the output, the network outputs two results:

- The expected optical flow at input pixel in the y direction
- The expected optical flow at input pixel in the x direction

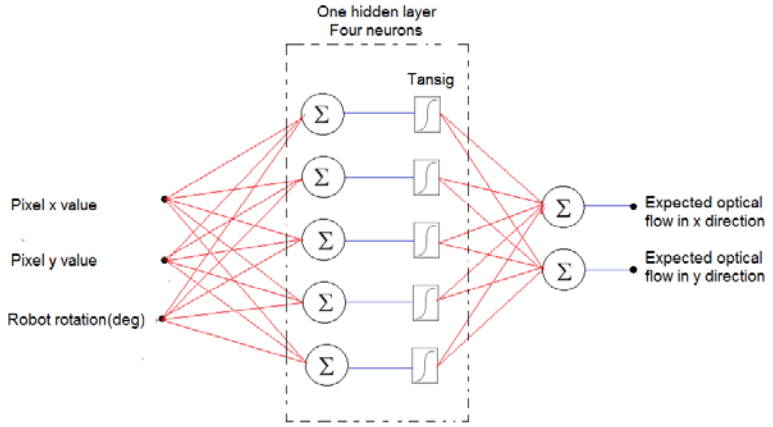


Figure 176: The neural network developed with inputs and outputs (biases not shown for simplicity).

Before training the network, strong outliers had to be removed from the data. Therefore, any optical flow vector in a set that pointed in a direction different from actual was rejected i.e. if flow is supposed to be from left to right any vector right to left is rejected.

Training of the network proceeded using default training function in MATLAB. Training required 142 iterations and yielded a mean square error of 31 pixels squared which corresponds to about an average deviation of 5.5 pixels. As can be seen there is noticeable number of outliers in the plot and this can be attributed to one of two reasons. The first could be due to network not finishing its training or could be due to the data being actual outliers that do not follow the learned relation by then network. The regression plot is shown in figure 177. On top of each plot there is an R value which is the degree of overlap between network output and targets. Training data set and validation sets have high degree overlap as an overall of 1 indicates perfect match while the actual overlap in both cases was greater than 0.9. Further, the high degree of overlap in validation data rules output the possibility of over fitting. This is further confirmed by the fact that the number of hidden neurons is very less than the training set.

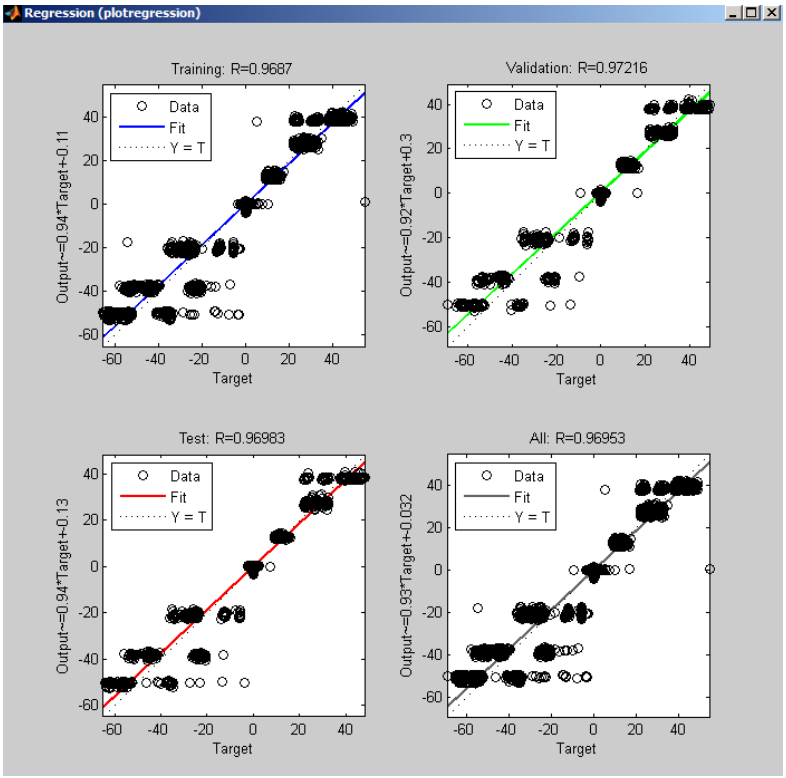


Figure 177: The regression plot from MATLAB for the training of the neural network.

The neural network was tested with validation data different from that used for training. However, the validation data has outliers that will affect the results. The average errors in the x and y directions were worked out as 27 and 1.1 pixels respectively. The reason for lower error in y direction is due to the magnitude of optical flow in y direction being lower than the x direction as the robot movement did not cause significant motion in y direction for the pixels. Further, notice the discrete nature of error distribution in the x direction which is due to discrete training data at different robot rotation angles.

iii. Testing and results

To visualize the performance of the network, the optical flow vectors obtained from the experiments were plotted as MATLAB figure with the starting pixel location labeled by black dot and a vector to final flow position whose magnitude reflects the magnitude of the optical flow. An example is shown in figure 178.

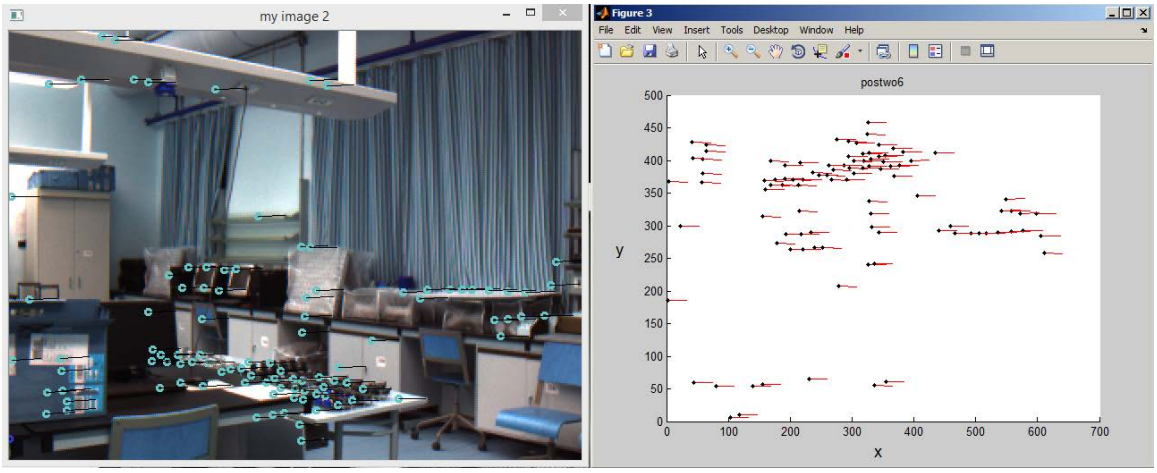


Figure 178: An example of an actual optical flow image (left) plotted as vectors only in MATLAB graph (right).

The network was then programmed to predict the optical flow given initial pixels location and robot rotation. The prediction for figure 178 is shown in figure 179.

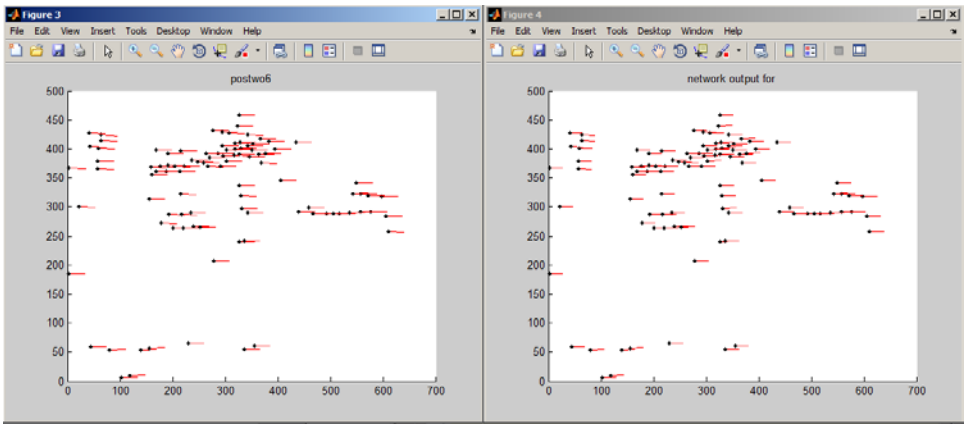


Figure 179: Optical flow prediction from network (right) versus actual flow (left) for situation in figure 293.

As can be seen, the network output is matching the actual experimental optical flow. The network is able to avoid noise in the training data. This can be shown by looking at the optical flow at two degree rotation in the presence of noise as shown in figure 180.

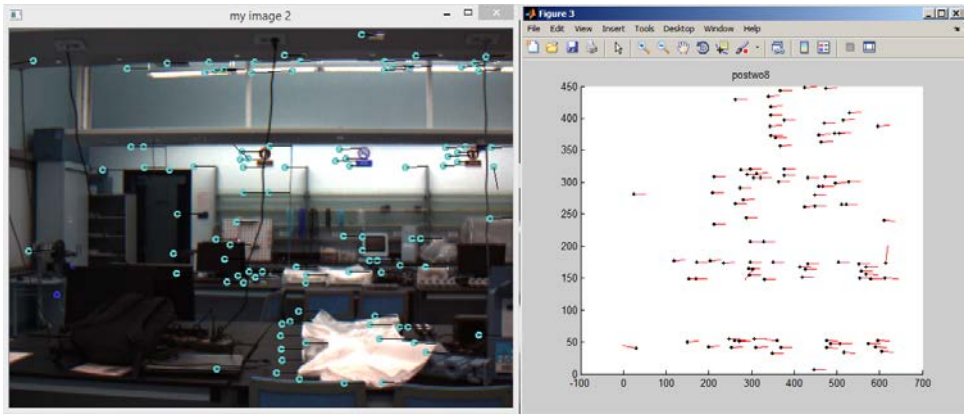


Figure 180: Flow with few noisy vectors left (scene with optical flow) right (optical flow vectors only).

In figure 180, there are some wrong optical flow vectors that deviate from the rest of the flow. Therefore, the network output is expected to be able to detect this noise. The network was run and the estimate of optical flow from the network is shown in figure 181 (left) together with the difference (right) between the network output and actual optical flow.

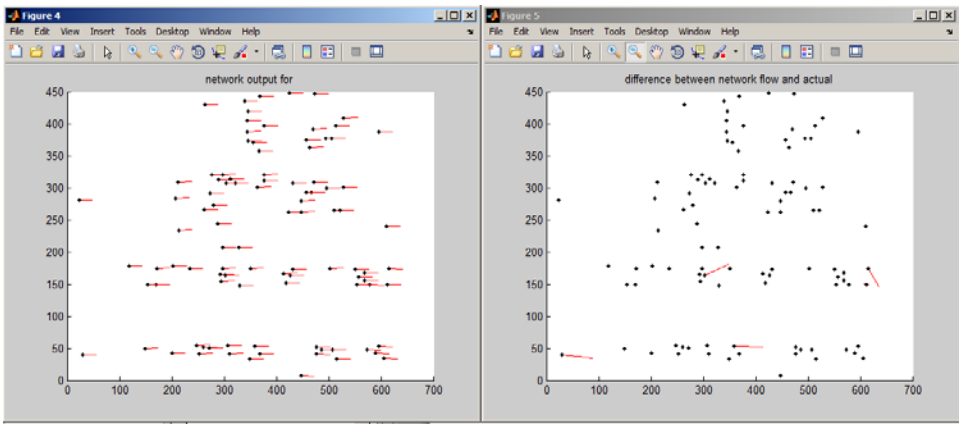


Figure 181: Flow from the neurl network (left) and the difference between neural flow and actual flow.

Notice how the erroneous vectors in figure 180 appear easily in figure 181 (on the right side). Therefore, the neural network can be used to cancel the optical flow component due to ego rotation of the robot despite the noise.

iv. Calculation of the FOE location

For the calculating the FOE location, a counting scheme similar to the work by [47] is used that relies on the divergence property of the flow similar to the works of [46], [45]. The process starts by defining a trigonometric function for calculating the orientation of the optical flow vectors so they lie between zero and 360 degrees. The arctangent function produces angles that are between -90 degrees and 90 degrees. Bearing in mind that the y axis of images in OpenCV is inverted.

This is then used to find the position of the FOE according to the divergence property. In order to ensure the accuracy of the results and prevent noisy readings from biasing the actual FOE location, all vectors that are likely to be due to noise are ruled out. To do this, optical flow algorithm from OpenCV was carried out at no motion to find an average value of the magnitude of noisy optical flow vectors. The average magnitude was worked out to be 0.2 pixels. Therefore, a threshold 10 times this value, i.e. 2 was chosen to ensure that noise will not interfere with FOE calculation.

Following this, the image is divided into 10 columns each column 64 pixel wide and extending all over the image height as shown in figure 182.

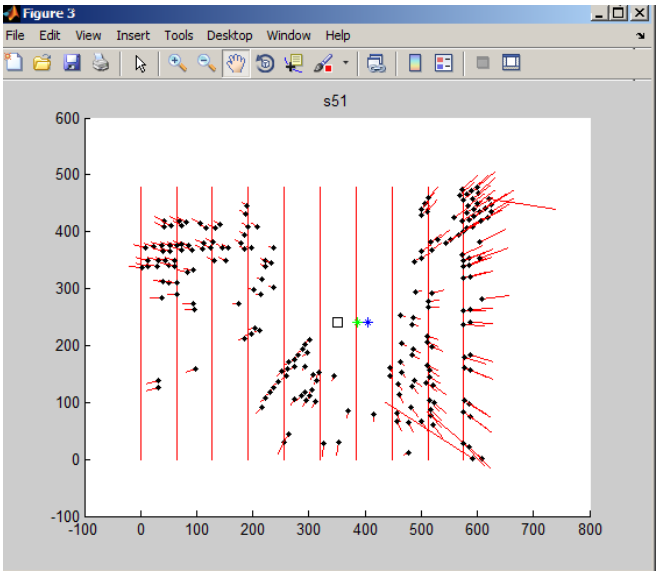


Figure 182: The image is divided into 10 columns to process for the location of the FOE.

Following this, each vector is given a value depending on its orientation. It is given a value of 500 if it points right (i.e. angle less than 90 or above 270 degrees) and a value of -500 if it points left. In the case of exactly vertical vectors a value of 0 is given. The values of vectors in each column are summed to give the overall value in each column stored as a one dimensional 10X1 array. The plot of the sum of vector values for the optical flow in figure 182 is shown in figure 183. The point where the zero crossing happens is the vicinity in which the FOE lies. The location of the FOE is taken to be in the middle of the column in which the zero crossing takes place. The process can be repeated in the y direction to find y position of FOE vertical position but it was found to be in most trials close to the center of the vertical axis.

4.8.3 Results of FOE finding algorithm

The algorithm was tested first without dynamic objects as the robot moved in translational motion in three modes. The first was a straight mode where the robot was parallel to the nearby obstacles, the second was when the robot was inclined towards the obstacle (on the right), and the third was when the robot was inclined leftwards away from the obstacle. In all cases the robot moved only in translational motion. Finally, the experiments were repeated at different displacements of 5 cm and 10 cm. at a velocity of 200 mm/s it will take the robot 0.25 seconds and 0.5 seconds to travel those distances respectively.

A frame was grabbed then the robot moved the specified distance (5m or 10cm) and stopped before grabbing a second frame and working out the optical flow and FOE location. Result of optical flow for straight motion at 5cm is sown in figure 183.

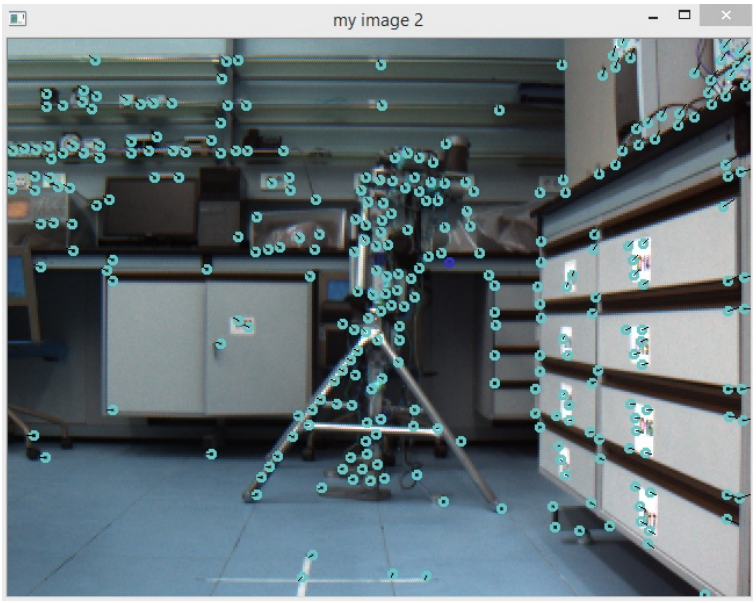


Figure 183: The result of optical flow for straight motion at 50 mm.

The optical flow vectors were mapped to MATLAB and the FOE position was worked out. The mapped flow vectors are shown in figure 184 with the FOE horizontal position marked with a square placed at 240 pixel y position.

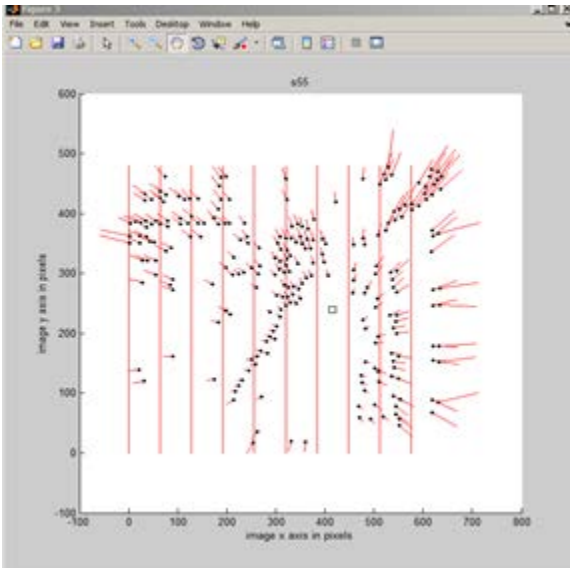


Figure 184: The x position of the FOE (right) for situation in figure 183.

The results from the other experimental trials indicated that the FOE was found with acceptable accuracy although results at higher distances travelled tended to be worse than shorter ones.

4.8.4 Discussions and limitations

From the results in this section it can be concluded that the developed algorithm successfully removes rotation ego motion component of the optical flow to locate the FOE horizontal position which is needed for obstacle avoidance. The neural network developed has successfully removed rotational components from the optical flow enabling calculation of the FOE. The next step is to use the FOE position calculation for guiding the robot to avoid obstacles.

Despite this success, the algorithm cannot work if the OpenCV identified features in the two frames for optical flow are wrong or unstable. When experiments where tried with the robot moving in continuous motion, the results were not satisfactory and many unstable features were being used. The Reason for this is due to the change in features properties as the robot moves and captures a moving frame due to indoor lightning, reflections, and changing perspective as well as the magnitude of distances involved. The distance moved is enough in the lab (despite being relatively small) to change lighting conditions which leads to different pixel values. The use of Histogram to normalize and even out the image helped to enable OpenCV optical flow to correctly identify the same features in the two frames for correct results in some of the trials but was not consistent enough. Therefore, the data from the camera although successful in isolation did not produce desirable result when used with the moving robot (even with no dynamic objects) this problem is being fixed but until now this is still work under progress.

4.9 Results of Overall system use

4.9.1 Driven run

The algorithms outlined and tested individually in chapter 3 were now combined together to plot a map of an indoor environment. Initially, the robot was asked to move on its own and avoid obstacles while mapping. However, this leads to a problem since in the lab chair legs existed that were much smaller than the size of the chair base. The chair leg was detectable by the robot but the wider base was not and this lead to collision problems. Figure 185 explains how the chair legs caused the problem. The robot will collide with base ring before even detecting it is dangerously close to the observable chair leg.

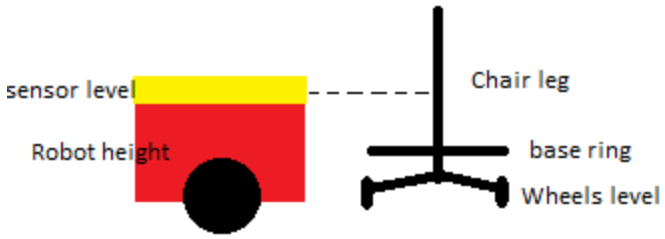


Figure 185: The effect of chair base on robot causing problems with obstacle avoidance.

Therefore, to test the algorithms developed the robot was first manually driven using a joystick with all algorithms functional except for the optical flow (unstable see section 3.8 c) and the obstacle avoidance to see their effect. The robot was driven over the same path as in figure 145 since it was marked with tape crosses in the ground to compare it to raw data in figures 142-143. The environment is shown again for convenience, and the resultant robot map is shown in figure 186.

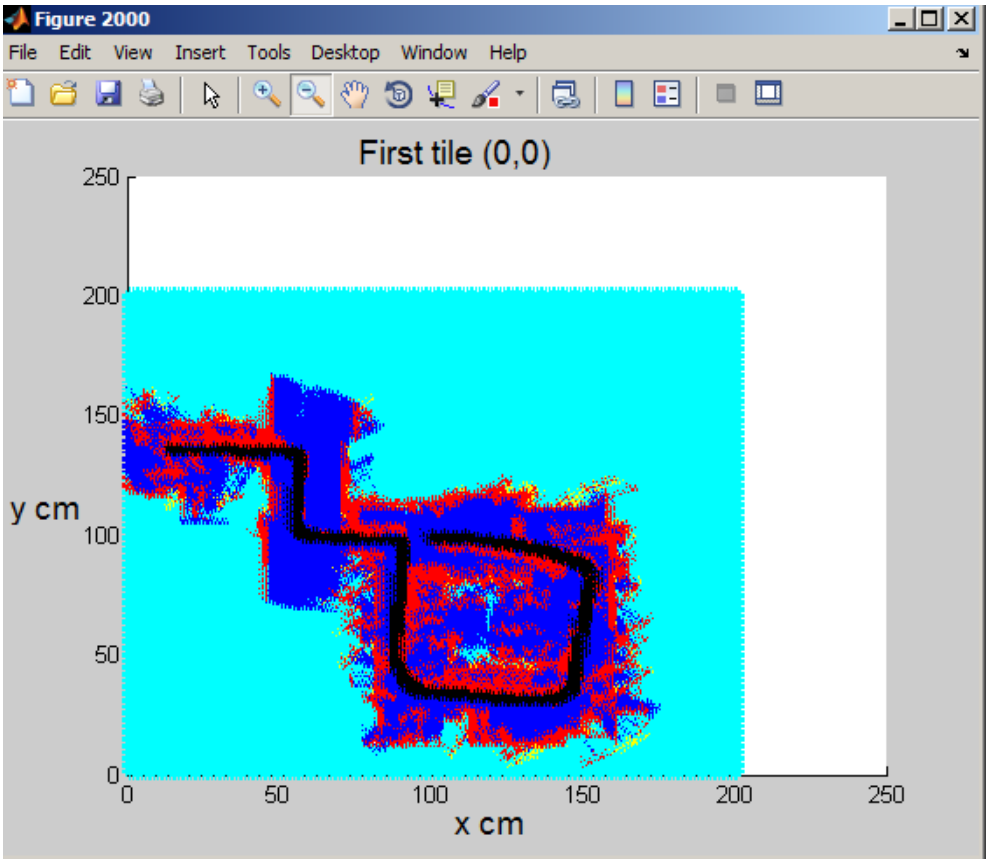


Figure 186: The map produced by the robot for figure 145.

The robot followed the exact path marked by the cross since it was manually driven. Notice how the map in figure 186 resembles the actual environment much better than the raw one from figure 142. The odometry error was not much of a problem as scan matching helped correct robot pose and dynamic objects were being removed.

4.9.2. Discussion

The overall hybrid system performed mapping well when compared with the raw robot without any system. The robot was able to build a map using sensor fusion in an indoors environment with active human beings who could possibly lead to wrong mapping but dynamic objects were being extracted to help with the consistency of the map.

Despite being successful in simulations when avoiding obstacles, the robot did not perform well in the lab environment and this was not due to a limitation in the algorithm itself but was due to hardware limitation as pointed out since the robot was unable to see the chairs at the correct distances due to their shapes. This could be overcome by enlarging the robot boundaries artificially so that it avoids the obstacle earlier on but this will limit the robot field of motion. Another solution would be to develop an algorithm that detects chair legs and enlarges them according to a chair geometrical model. One other approach is to use large cardboards at chair wheels to make them easy for the robot sensors to detect but this involves modifying the environment.

Finally the map produced by the robot can be further processed to remove the sparse area based on continuity of objects in the environment. Due to the nature of sensor model used, the map is sparse and some gaps exist in the solid objects. This is easy for human to identify but will be beneficial for the robot to eliminate too. This can be part of a future investigation since it was not seen to happen a lot. Additionally, in the corridor there is a blocked region immediately in front of the robot path which is thought to happen due to wrong identification of false dynamic objects which are not removed immediately.

Chapter 5
Conclusion and Future work

5.1 Conclusion

The investigation on the use of hybrid approach to combine various sensor and algorithms in a mobile robot SLAM application was followed by this thesis. The algorithms and sensors were tested one by one then combined together in one system. The Thesis concluded several points based on the discussion sections for each of the algorithms. First that the use of sensor fusion in grid map is possible enabling the employment of laser and sonar sensors for better map accuracy compared to a simpler sonar map. Moreover, the use of the modified sensor model that limits the updated cells has proven to reduce the time needed as verified by the simulations and comparisons done. The validation of this is reflected by the practical value that the use of this sensor model adds to the robot as it enables real time operation while the robot maps.

Regarding the use of UKF to combine the results of feature based scan matching with odometry update, it was proven and verified by the experiments carried out before the use of the UKF update and after its use which showed that the resultant robot pose and hence map are more accurate than the raw case. The validation of this point is that in practice the smaller the error in robot pose and localization the smaller the error in the resulting map.

Further, the detection and tracking of dynamic objects based on geometrical features has been verified using experiments only since it is not possible to simulate humans in the simulator. The experiments verified that humans can be tracked and hence excluded from the resulting map to reduce any error that is likely to happen. The validation for this is that it leads to less error prone maps as dynamic objects are less likely to affect the map.

Another important conclusion was regarding the use of TBF corners from sonar sensor where it was verified using simulations and experiments that corners from the sonar sensor are far less accurate than the laser sensor and hence they were only included in the update if the laser fails to return any features. Further, the use of feature based scan matching is affected by the number of features in the environment as environments like corridors with little features tend to cause wrong updates. From a validation point of view this means that the use of sonar corners alone will affect the map accuracy.

It was concluded that work with images and image based sensors such as camera when limited to low level corner features for optical flow may not yield consistent results that help with robot mapping and further work is needed. This is due to the problems with intensity changes in the image as the robot moves. This was verified in experiments involving optical flow. However, the design and use of neural networks to cancel the rotational ego motion component proved successful and could have been implemented in the overall system if the optical flow algorithm from OpenCV optical flow produced stable output. The validation of this point is that more work is needed with image based sensors to reach better results.

Another point that was drawn from this research is that systems with many components functioning with one another will tend to be complicated and individual modular blocks may need tuning to be accommodated in the overall system which increases difficulty in implementation. However, the advantage is better performance than simpler systems. This was verified by testing the system itself as it was built block by block and the by the final outcome which was better than a raw system and its map represented the environment in a better way.

5.2 Future work

The investigation in this thesis has led to many conclusions and discovery of many limitations from which further future work could be planned for more investigation. The Robot calibration procedure in chapter 4.1 was made using the standard method not accounting for the floor type. It is said in the robot manual [51] that floor type will affect the systematic and random odometry errors differently with carpets being worse than other types. Therefore, a future investigation could be carried out to see the effect of floor type on calibration results and how the adjustment of calibration according to environment affects the resultant map.

The obstacle avoidance algorithm was based on a reactive subsumption principle for extreme speed and is different from methods like VPH and VFH in literature so a future work could be to compare the performance time wise of those two algorithms. Further, the obstacle avoidance algorithm could be coupled with a path planning algorithm that divides the robot operation between exploration and operation where the path planning algorithm in discovery phase helps the robot discover the undiscovered areas by dictating target locations while in operation phase commands are given by the user. However, path planning is beyond the scope of this thesis so no attempt was made to test it and this makes a good point for future work.

The sonar detection algorithm results were fused with the laser data on a feature level by fusing corners when the laser fails to detect any corners only. This is due to the very poor resolution and accuracy of sonar TBF data compared to laser corner extraction as was pointed out in chapter 4.3. However, the sonar data could be beneficial even with low resolution if it is fused correctly using a special UKF. This was not tried and could be a point for further future research where the sonar TBF data could be fused directly with laser corners by using a weighting function that produces an output dependent on the predicted performance of the sonar. However, this necessitates the development of sonar scan matching which is discussed in the next paragraph.

The scan matching algorithm could be modified in future work to include the sonar scan matching where corners extracted from sonar sensors could be used to produce an independent estimate of scan matching from that of laser then the two are latter combined. More investigations into the possibility of using point based and feature based scan matching simultaneously to enhance one another where feature matching is used to find a rough estimate then point to point matching (based on ICP) could be used to refine the estimate. This approach could potentially be faster than pure ICP and comparable to feature based matching but with higher robustness and the ability to operate in unstructured environments or low feature corridors. Future work for the UKF Algorithm could include possibility of using variable variances and setting these variances online while the robot operates based on the flow of data and previous estimates. This is will enhance the operation of the UKF since if for example one source of data fails, then UKF could adjust its belief about the correctness of data from that source on the fly as it happens. Then when the source operates normally with correct reading the trust in its accuracy (represented by the covariance of its quantity) is increased again.

Grid maps in particular are known for being computationally expensive. In the grid map algorithm implemented in this chapter, the grid map was modified to include a local map and a global map with a transition occurring between different tiles as the robot moves. Further, each individual tile was processed by limited sonar and laser ranges to minimize time. Therefore, future investigations could focus on finding ways to improve the computational efficiency. One such method is the use of variable size grid map. This has the feature that the size of each individual grid is varied according to the obstacles in the environment thus free areas are represented and updated with one big grid cell instead of many smaller time consuming ones.

However, investigation is needed to ensure that the time saved and computation decrease outweighs the processing and time needed to bisect and segment the tile into varying size grid cells according to the environment.

The dynamic object detection has revealed the difficulty faced by robotic platforms which use 2D sensors for motion detection since they are the most accurate with the consequence that limited features can be used to identify objects as they appear on a 2D laser scan. In particular, features of human leg size were used to identify humans. However, other objects like chairs and tables could move and are harder to identify and track. Future work can investigate the possibility of dealing with that situation. Further, in future work a dynamic objects map could be maintained for moving humans to not only exclude them from metric map but to ensure that even if they are seen after an obstruction they are correctly re-identified. This will need the use of multiple KF to track each individual object with the difficulty of choosing an appropriate dynamic model for those filters and this provides scope for significant future work.

The camera sensor provides large amount of information but it is computationally expensive to use. One way to investigate possible use and benefit is to use parallel computing where an array of parallel processors are carried by the robot and used for processing image data to provide faster rates of update. Further, alternative methods to OpenCV optical flow could be investigated to yield a more stable algorithm to overcome the problem as was outlined in section 4.8.

References

[1]S. Thrun and J. Leonard, "Simultaneous Localization and Mapping", *Springer Handbook of Robotics*, pp. 871-889, 2008.

[2]H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I*", IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99-110, 2006.

[3]R. Smith and P. Cheesman, “On the representation of spatial uncertainty,” Int. J. Robot. Res., vol. 5, no. 4, pp. 56–68, 1987.

[4]M. Csorba, “Simultaneous Localisation and Map Building,” Ph.D. dissertation, Univ. Oxford, 1997.

[5]C. Stachniss, "Robotic Mapping and Exploration", *Springer Tracts in Advanced Robotics*, 2009.

[6]T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (SLAM): part II", *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 108-117, 2006.

[7]W. Chang, C. Wu, W. Luo and H. Ling, "Mobile robot navigation and control with monocular surveillance cameras", *2013 CACS International Automatic Control Conference (CACS)*, 2013.

[8]S. Wen, K. Othman, A. Rad, Y. Zhang and Y. Zhao, "Indoor SLAM Using Laser and Camera with Closed-Loop Controller for NAO Humanoid Robot", *Abstract and Applied Analysis*, vol. 2014, pp. 1-8, 2014.

[9]F. Fang, X. Ma, X. Dai and K. Qian, "A new multisensor fusion SLAM approach for mobile robots", *J. Control Theory Appl*., vol. 7, no. 4, pp. 389-394, 2009.

[10]N. Dhiman, D. Deodhare and D. Khemani, "A review of path planning and mapping technologies for autonomous mobile robot systems", *Proceedings of the 5th ACM COMPUTE Conference on Intelligent & scalable system technologies* - COMPUTE '12, 2012.

[11]G. Dissanayake, S. Huang, Z. Wang and R. Ranasinghe, "A review of recent developments in Simultaneous Localization and Mapping", 2011 *6th International Conference on Industrial and Information Systems*, 2011.

[12]M. Milford, R. Schulz, D. Prasser, G. Wyeth and J. Wiles, "Learning spatial concepts from RatSLAM representations", *Robotics and Autonomous Systems*, vol. 55, no. 5, pp. 403-410, 2007.

[13]N. Sünderhauf and P. Protzel, "Beyond RatSLAM: Improvements to a biologically inspired SLAM system", 2010 IEEE 15th *Conference on Emerging Technologies & Factory Automation (ETFA 2010)*, 2010.

[14]M. Milford, G. Wyeth and D. Prasser, "RatSLAM: a hippocampal model for simultaneous localization and mapping", *IEEE International Conference on Robotics and Automation, 2004. Proceedings.ICRA '04.2004*, 2004.

[15]Milford, M. J. and Wyeth, G. F. "Hippocampal models for simultaneous localisation and mapping on an autonomous robot",*Australasian Conference on Robotics and Automation*, 2003.*Proceedings*

[16]A. Davison, I. Reid, N. Molton and O. Stasse, "MonoSLAM: Real-Time Single Camera SLAM", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052-1067, 2007.

[17]J. Meyer and D. Filliat, "Map-based navigation in mobile robots:", *Cognitive Systems Research*, vol. 4, no. 4, pp. 283-317, 2003.

[18] Hyun ChulRoh, Chang Hun Sung, Min Tae Kang and Myung Jin Chung, "Fast SLAM using polar scan matching and particle weight based occupancy grid map for mobile robot", *2011 8th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 2011.

[19]R. Murphy, *Introduction to AI robotics*. Cambridge, Mass.: MIT Press, 2000.

[20]H. Kim, W. Chung and Y. Yoo, "Detection and tracking of human legs for a mobile service robot", *2010 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 2010.

[21]K. Konolige, E. Marder-Eppstein and B. Marthi, "Navigation in hybrid metric-topological maps", *2011 IEEE International Conference on Robotics and Automation*, 2011.

[22]S. Xue, Y. Zhang, J. Cheng and Q. Zhao, "Topological map building for mobile robots based on thining algorithm", *2014 11th International Computer Conference on Wavelet Actiev Media Technology and Information Processing(ICCWAMTIP)*, 2014.

[23]B. Kuipers, J. Modayil, P. Beeson, M. MacMahon and F. Savelli, "Local metrical and global topological maps in the hybrid spatial semantic hierarchy", *IEEE International Conference on Robotics and Automation, 2004. Proceedings.ICRA '04.2004*, 2004.

[24] Chieh-Chih Wang, C. Thorpe and S. Thrun, "Online simultaneous localization and mapping with detection and tracking of moving objects: theory and results from a ground vehicle in crowded urban areas", *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*.

[25] Chieh-Chih Wang and C. Thorpe, "Simultaneous localization and mapping with detection and tracking of moving objects", *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*.

[26] Fantian Kong, Youping Chen, JingmingXie, Gang Zhang and Zude Zhou, "Mobile Robot Localization Based on Extended Kalman Filter", *2006 6th World Congress on Intelligent Control and Automation*, 2006.

[27]H. Khoury and V. Kamat, "Evaluation of position tracking technologies for user localization in indoor construction environments", *Automation in Construction*, vol. 18, no. 4, pp. 444-457, 2009.

[28]K. Kouzoubov and D. Austin, "Hybrid topological/metric approach to SLAM", *IEEE International Conference on Robotics and Automation, 2004. Proceedings.ICRA '04.2004*, 2004.

[29]T. Bailey and E. Nebot, "Localisation in large-scale environments", *Robotics and Autonomous Systems*, vol. 37, no. 4, pp. 261-281, 2001.

[30]I. Loevsky and I. Shimshoni, "Reliable and efficient landmark-based localization for mobile robots", *Robotics and Autonomous Systems*, vol. 58, no. 5, pp. 520-528, 2010.

[31]C. Urdiales, A. Bandera, R. Ron and F. Sandoval, "Real time position estimation for mobile robots by means of sonar sensors", *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*.

[32]F. Fraundorfer, C. Engels and D. Nister, "Topological mapping, localization and navigation using image collections", *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007.

[33]H. Badino, D. Huber and T. Kanade, "Real-time topometric localization", *2012 IEEE International Conference on Robotics and Automation*, 2012.

[34]Y. Lee, T. Kwon and J. Song, "SLAM of a Mobile Robot using Thinning-based Topological Information", *International Journal of Control, Automation, and Systems*, vol. 5, no. 5, pp. 577-583, 2007.

[35]S. Oh, M. Hahn and J. Kim, "Simultaneous Localization and Mapping for Mobile Robots in Dynamic Environments", *2013 International Conference on Information Science and Applications (ICISA)*, 2013.

[36]A. Diosi, G. Taylor and L. Kleeman, "Interactive SLAM using Laser and Advanced Sonar", *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*.

[37] Jinwoo Choi, SunghwanAhn and Wan Kyun Chung, "Robust sonar feature detection for the SLAM of mobile robot", *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.

[38]O. Wijk and H. Christensen, "Triangulation-based fusion of sonar data with application in robot pose tracking", *IEEE Trans. Robot. Automat.*, vol. 16, no. 6, pp. 740-752, 2000.

[39]J. Choi, M. Choi and W. Chung, "Topological localization with kidnap recovery using sonar grid map matching in a home environment", *Robotics and Computer-Integrated Manufacturing*, vol. 28, no. 3, pp. 366-374, 2012.

[40]J. Martínez, J. González, J. Morales, A. Mandow and A. García-Cerezo, "Mobile robot motion estimation by 2D scan matching with genetic and iterative closest point algorithms", *Journal of Field Robotics*, vol. 23, no. 1, pp. 21-34, 2006.

[41]J. Kuiaski, H. Neto and A. Lazzaretti, "Focus of Expansion Estimation for Motion Segmentation from a Single Camera", *Workshop de VisaoComputacional*, 2011.

[42]C. Braillon, C. Pradalier, J. Crowley and C. Laugier, "Real-time moving obstacle detection using optical flow models", *2006 IEEE Intelligent Vehicles Symposium*.

[43]G. Stein, O. Mano and A. Shashua, "A robust method for computing vehicle ego-motion", *Proceedings of the IEEE Intelligent Vehicles Symposium 2000 (Cat. No.00TH8511)*.

[44]R. Constance and M. Holloway, "Detecting moving objects in an optic flow field using direction- and speed-tuned operators. - PubMed - NCBI", *Ncbi.nlm.nih.gov*, 2016.[Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/24607912. [Accessed: 20- Apr- 2016].

[45]M. Narayana, A. Hanson and E. Learned-Miller, "Coherent Motion Segmentation in Moving Camera Videos Using Optical Flow Orientations", *Cv-foundation.org*, 2013. [Online]. Available: http://www.cvfoundation.org/openaccess/content_iccv_2013/html/Narayana_Coherent_Motion_Segmentation_2013_ICCV_paper.html. [Accessed: 20- Apr- 2016].

[46]D. Sazbon, H. Rotstein and E. Rivlin, "Finding the focus of expansion and estimating range using optical flow images and a matched filter", *Machine Vision and Applications*, vol. 15, no. 4, pp. 229-236, 2004.

[47]K. Souhila and A. Karim, "Optical Flow based Robot Obstacle Avoidance", *Int J Adv Robotic Sy*, p. 1, 2007.

[48] He Feng, Fang Yongchun, Wang Yutao and Ban Tao, "Practical feature-based simultaneous localization and mapping using sonar data", *2008 27th Chinese Control Conference*, 2008.

[49]B. Johann and L. Feng, "UMBmark: A method for measuring, comparing, and correcting dead-reckoning errors in mobile robots", 1994.

[50]Korodi A., Dragomir, "CorrectingOdometry Errors for Mobile Robots Using Image Processing", *Proceedings of the IAENGInternational Conference on Control andAutomation*, vol. 2, pp. 1040-1045, 2010.

[51] "ARCOS pioneer robots user Manual", Adept Mobile Robots.

[52]"Math Forum - Ask Dr. Math", *Mathforum.org*, 2016. [Online]. Available: http://mathforum.org/library/drmath/view/51836.html. [Accessed: 10- Oct- 2015].

[53]D. Kohanbash, *the field of view and range for Lidars*. 2014. Available at:http://robotsforroboticists.com/sick-tim551-lidar-review/

[54]A. Diosi and L. Kleeman, "Laser scan matching in polar coordinates with application to SLAM", *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.

[55]V. Nguyen, A. Martinelli, N. Tomatis and R. Siegwart, "A comparison of line extraction algorithms using 2D laser rangefinder for indoor mobile robotics", *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.

[56]p. bloomfield, "weighted least squares", NC state University, 2014.

[57]G. Borges and M. Aldon, "A split-and-merge segmentation algorithm for line extraction in 2D range images", *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*.

[58]V. Hlava, Czech Technical University, 2004. Available at: http://cmp.felk.cvut.cz/~hlavac/Public/TeachingLectures/RANSAC.pdf

[59]R. Faragher, "Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation [Lecture Notes]", *IEEE Signal Process. Mag.*, vol. 29, no. 5, pp. 128-132, 2012.

[60]E. Wan and R. Van Der Merwe, "The unscented Kalman filter for nonlinear estimation", *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*.

[61]B. Åkesson, J. Jørgensen, N. Poulsen and S. Jørgensen, "A tool for kalman filter tuning",*Computer Aided Chemical Engineering*, pp. 859-864, 2007.

[62]Y. Jia, *Discrete-time Kalman Filter and the Particle Filter*, 1st ed. Iowa State University, 2015.

[63]S. Julier, "The scaled unscented transformation", *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, 2002.

[64]E. Chatzi and A. Smyth, "The unscented Kalman filter and particle filter methods for nonlinear structural system identification with non-collocated heterogeneous sensing", *Structural Control and Health Monitoring*, vol. 16, no. 1, pp. 99-123, 2009.

[65]R. Capua and A. Bottaro, "Implementation of the Unscented Kalman Filter and a Simple Augmentation System for GNSS SDR Receivers", *Proceedings of the 25th International Technical Meeting of The Satellite Division of the Institute of Navigation*, pp. 2398 - 2407, 2012.

[66]J. Ko and D. Fox, "GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models", *Autonomous Robots*, vol. 27, no. 1, pp. 75-90, 2009.

[67]M. Irani, B. Rousso and S. Peleg, "Recovery of ego-motion using region alignment",*IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 3, pp. 268-272, 1997.

[68]S. Perkins, "Real time optical flow based range sensing on mobile robots", Masters, University of Edinburgh, 1993.

[69]M. Al Shamisi, A. H. and H. N. Hejase, "Using MATLAB to Develop Artificial Neural Network Models for Predicting Global Solar Radiation in Al Ain City – UAE", *Engineering Education and Research Using MATLAB*, 2011.

[70]"Image Size Chart", *Siliconimaging.com*, 2016. [Online]. Available: http://www.siliconimaging.com/Lens Image formats.htm. [Accessed: 20- Apr- 2016].

[71]*MATLAB*.Mathworks,V. 7.6.0, 2008.

[72]A. Davison, Y. Gonzalez cid and N. kita, "REAL-TIME 3D SLAM WITH WIDE-ANGLE VISION", *Proc. IFAC Symposium on Intelligent Autonomous Vehicles, Lisbon*, 2004.

[73]J. Leonard and P. Newman, "Consistent, convergent, and constant-time SLAM",*Proceedings of the 18th international joint conference on Artificial intelligence*, pp. 1143-1150, 2003.

[74]A.I. Eliazar and R. Parr, "DP-SLAM 2.0," in Proc. IEEE Int. Conf. Robotics Automation, 2004, pp. 1314–1320.

[75]F. Dellaert and M. Kaess, "Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing", *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1181-1203, 2006.

[76]M. Bosse, "Simultaneous Localization and Map Building in Large-Scale Cyclic Environments Using the Atlas Framework", *The International Journal of Robotics Research*, vol. 23, no. 12, pp. 1113-1139, 2004.

[77] Jonghyuk Kim and S. Sukkarieh, "Autonomous airborne navigation in unknown terrain environments", *IEEE Trans. Aerosp. Electron.Syst.*, vol. 40, no. 3, pp. 1031-1045, 2004.

[78]M. Walter, R. Eustice and J. Leonard, "A Provably Consistent Method for Imposing Sparsity in Feature-Based SLAM Information Filters", *Springer Tracts in Advanced Robotics*, pp. 214-234.

[79]B. Morse, Lecture 15: Segmentation (Edge Based, Hough Transform). University of Edinburgh, 2000.

[80]"Robot Architectures", Cs.brown.edu, 2016. [Online]. Available: http://cs.brown.edu/~tld/courses/cs148/02/architectures.html. [Accessed: 20- Jan- 2016].

[81]C. Constable, Total Least Squares and Robust Methods, 1st ed. San Diego: UC San Diego, 2016. Available at http://igppweb.ucsd.edu/~cathy/Classes/SIO223A/sio223a.chap8.pdf

[82]M. K. Habib, "Real Time Mapping and Dynamic Navigation for Mobile Robots", Int J Adv Robotic Sy, p. 1, 2007.

Appendices

Appendix A

Code for Obstacle avoidance

```
//before calling function
double oder=robot.getOdometerDistance();//Get odometer distance

 struct ret //structure to return the processing from subsumption
layers
        {
            int key;
            double f;//forward vel control
            double rot;//rot control
        };


 ret l3;ret l4;ret l5;ret l0;//structure to return from layers 3, 4, 5,
and 0.
while(1)//main function
{
 double ng=0;
     my.getAdjusted(&x,&y);
 double kyr=0;
     int a=0;int b=0;//for output from layers 1 and 2
// if(robot.getOdometerDistance()-oder>0)
     if(my.getButton(1)==1)
     {
         robot.setVel(3*y);
         robot.setRotVel(-0.02*x);
     }
     else
 {//oder=robot.getOdometerDistance();
    //from avoid subsumption
double distance=laser->currentReadingPolar(-90,90)-
robot.getRobotRadius();//(robot->getSonarReading(0)->getRange()+robot-
>getSonarReading(1)->getRange()+robot->getSonarReading(2)-
>getRange()+robot->getSonarReading(3)->getRange()+robot-
>getSonarReading(4)->getRange()+robot->getSonarReading(5)-
>getRange()+robot->getSonarReading(6)->getRange()+robot-
>getSonarReading(7)->getRange())/8;
double kk=pow(distance/5000,0.5)/pow(5,0.5);//l2 weight
double kk2=0.5*(1+tanh((3*0.001*distance)-9));//l5 weight

//get response of layers
 a=layer1(&robot,&sonar);
b=layer2(&robot);
```

```
l3=layer3(&robot);
l4=layer4(&robot);
l5=layer5(&robot);cout<<"\r"<<l4.rot<<" "<<l5.rot<<"\n";
l0=layer0(2000,000,&robot);
//resolver
if(b==1&&a==0)
{kyr=1;
    robot.setVel(-
50);robot.setRotVel(20+(20*abs(robot.getVel()/100)));ng=robot.getPose()
.getTh();
    oder=robot.getOdometerDistance();cout<<"\n"<<"1";
}
if(b==0&&a==1)
{kyr=1;robot.setRotVel(20);
    robot.setVel(-50);ng=robot.getPose().getTh();
    oder=robot.getOdometerDistance();cout<<"\n"<<"2";
}
if(b==1&&a==1)
{kyr=1;
    robot.setVel(-50);
    robot.setRotVel(20);ng=robot.getPose().getTh();
    oder=robot.getOdometerDistance();cout<<"\n"<<"3";
}
if(b==0&&a==0&&kyr==0)
{my.getAdjusted(&x,&y);
    robot.setVel(0.5*l3.f);//robot.setVel(3*y);
//robot.setRotVel(l4.rot+(l5.rot*kk2));//robot.setRotVel(-0.02*x);
ng=robot.getPose().getTh();
    oder=robot.getOdometerDistance();
    robot.setRotVel((((1-
kk)*l4.rot)+(l5.rot*kk2)+(l0.rot*1))/(1+kk2+1));
}
if(kyr==1&&(abs(ng-
robot.getPose().getTh()>35)||robot.getOdometerDistance()-oder>150))
{
    kyr=0;
}
    avid<<3*y<<" "<<-0.02*x<<" "<<sonar.currentReadingPolar(-50,50)<<"
"<<sonar.currentReadingPolar(-10,10)<<" "<<a<<" "<<b<<";"<<"\n";
 }

 //layer1(&robot,&sonar);
 //layer2(&robot);
if(l0.key==1)//if target reached
{
    robot.setVel(0);
    break;
}
}
```

```
//////////////////////////////////////////////////////////////
Function definition
struct ret layer0(int x, int y,ArRobot *robot) //this is go to target
    {
        double rob_ang=0;//this is used to modify robot angle to cover
range from 0 to 360 instead of -180 to 180
        double key=0;//this is used to declare when robot has reached
desired target
        double vel_control;//variable to hold desired forward velocity
        double to_target;//distance to target
        double control;//control signal
        struct ret group;//structure that will be returned by function

        if(robot->getTh()<0)
{
rob_ang=robot->getTh()+360;
}
else
{
rob_ang=robot->getTh();
}//modify range to 0-360


        to_target=abs(170+robot->getX()-x)+abs(robot->getY()-y);
        control=3*(-rob_ang+(atan2(y-robot->getY(),x-robot->getX()-
0)*57));//compute desired control angle *57 is to convert to degrees
        vel_control=(500*(to_target/(to_target+500)));//setvelocity to
slow as we approach target

        if (abs(control)<5&&x<robot->getX()){control=0.1*180;}
        if (abs(control)<5&&x>robot->getX()){control=0.1*-180;}

        if(abs(robot->getX()-x)+abs(robot->getY()-y)<200)
        {
            key=1;
        }else{key=0;}
        group.key=key;
        group.f=vel_control;
        group.rot=1*control*robot->getVel()/400; //gain is 3/400

        return group;
}
int layer1(ArRobot *robot,ArSonarDevice *sonar)// this is the avoid
front collision layer
{
double range=sonar->currentReadingPolar(-50.0,50.0)-300;
 int lr=0;
```

```
//  if(sonar->currentReadingPolar(-50.0,50.0)-robot-
>getRobotDiagonal()<200)//read sonar in front (subtract robot diagonal)
and if <200 act
if(range<200)
{
    //robot->setVel(-50);robot->setRotVel(50);//move backwards and
rotate
    lr=1;
//ArUtil::sleep(200);//sleep to give time to robt to react

}
return lr;
}
int layer2(ArRobot *robot)// this is the corner avoiance layer
{int conrr=0;
    if(Corn_detect(robot)==1&&corner2==0)//if corner is detected and
counter is reset
    {
        timer1.setToNow();corner1=1;corner2=1;//set timer and counter
    }

    if(Corn_detect(robot)==1&&timer1.mSecSince()>30&&corner1>0)
    {
        corner1++;timer1.setToNow();//keep conting every 30 ms
    }
    else if(Corn_detect(robot)==0&&timer1.mSecSince()>30&&corner1>0)
    {
        ;timer1.setToNow();//if no corner detect decrement counter
    }
    if(corner1>3)//if corner is confirmed
    {cout<<"\n"<<"new ";
        corner1=0;corner2=0;conrr=1;//robot-
>setDeltaHeading(90);ArUtil::sleep(2000);sleeps=1;sleept=2000;
    }//change heading and sleep to give time to robot to react
    return conrr;
}
struct ret layer3(ArRobot *robot)//control of front speed function
    {
        struct ret group;//initialise structure to be returned
        double front=0;//initialise front to 0
        for(int i=0;i<8;i++)
        {
            front=front+(sin((robot->getSonarReading(i)-
>getSensorTh()+90)*PI/180)*pow((double)robot->getSonarReading(i)-
>getRange(),1));//work out how crowded the region is
        }

        group.key=0;//key is not needed hetre so set to 0
        group.f=front*0.034;//set forward component
```

```
        group.rot=0;//layer does not affect rotational component
        return group;
        cout<<front*0.034<<"/n";
    }
struct ret layer4 (ArRobot *robot)//layer for side avoidance
{
    struct ret group;
    double priorL=0;//set prior left to zero
    double priorR=0;//set prior right to zero

    double side_ang=0;


    for(int i=0;i<4;i++)//for al sonar sensors on the left side
    {
        double S=0;
        S=pow(5000.0,0.1)-(abs(cos((robot->getSonarReading(i)-
>getSensorTh()+90)*PI/180))*pow((double)robot->getSonarReading(i)-
>getRange(),0.1));//work out the value of S_left
        if(S<0)//just in case user wants to consider ranges less than
5000
        {
            priorL=priorL;
        }
        else
        {
            priorL=priorL+S;//add the contributions of each sonar to
the priorL
        }
    }

    for(int i=4;i<8;i++)//for right side sonar sensors
    {
        double S=0;
        S=pow(5000.0,0.1)-(abs(cos((robot->getSonarReading(i)-
>getSensorTh()+90)*PI/180))*pow((double)robot->getSonarReading(i)-
>getRange(),0.1));
        if(S<0)
        {
            priorR=priorR;
        }
        else
        {
            priorR=priorR+S;//add the contributions of each sonar to
the priorR
        }
    }
```

```
    priorL=priorL/(priorL+priorR);//normalise
    priorR=1-priorL;//find prior R
    side_ang=(priorR-priorL)*1*(robot->getVel()+50);//control signal

    group.key=0;
    group.f=0;
    group.rot=side_ang;

    return group;
}
struct ret layer5 (ArRobot *robot)//layer for far range avoidance

{
    struct ret group;
    double fL=robot->getSonarReading(3)->getRange();//read sonar 3
(front left)
    double fR=robot->getSonarReading(4)->getRange();//read sonar 4
(front right)

    if(fL+200<fR)//if obstacle to left rotate right
    {
        group.rot=(-0.035*robot->getVel());
    }
    else if(fL>fR+200)//if obstacle to right rotate left
    {
        group.rot=(0.035*robot->getVel());
    }
    else {group.rot=0;}//must set it to zero to prevent huge numbers
from appearing

    group.key=0;
    group.f=0;
    return group;

}
```

Appendix B

TBF Code

Appendix B.1: Get sonar readings

```
void getSonar(ArRobot *robot)//function to get sona readings and
arrange in shifting table

{

	ofstream son_table;
	son_table.open("son_corn.txt", ios_base::app);

	//check sonar column counter to see if needed to shift table
	if (sonar_column_counter==10)

	{
		//start shifting table
		sonar_column_counter=9;

		for(int i=0;i<8;i++) //i is rows which is equal to number
of sonars on robot
		{
			for(int j=0;j<9;j++)//j is the number columns in the
shifting table
			{
				sonar_range[i][j]= sonar_range[i][j+1];//shift
columns leftwise
				sonar_y[i][j]= sonar_y[i][j+1];//shift columns
leftwise
				sonar_x[i][j]= sonar_x[i][j+1];//shift columns
leftwise
				gamma[i][j]=gamma[i][j+1];//shift columns
leftwise
				y_t[i][j]=y_t[i][j+1];//shift columns leftwise
				x_t[i][j]= x_t[i][j+1];//shift columns leftwise
			}
		}
	}


	//fill the new column (latest in the table) with recent readings
	for (int i=0;i<8;i++)
	{
		//loop over all sonars to fill table
		double sensor_diag=sqrt(pow(robot->getSonarReading(i)-
>getSensorY(),2)+pow(robot->getSonarReading(i)->getSensorX(),2));//this
gets the sensor diagonal (distance from robot centre)
```

```
            sonar_range[i][sonar_column_counter]= robot-
>getSonarReading(i)->getRange();//fill first row of table by reading of
first sonar.
            gamma[i][sonar_column_counter]=(robot->getSonarReading(i)-
>getThTaken()*PI/180)+(robot->getSonarReading(i)-
>getSensorTh()*PI/180);//get current sonar orientation=robot
orientation+sonar orientation on robot
            sonar_y[i][sonar_column_counter]= robot-
>getSonarReading(i)-
>getYTaken()+(sensor_diag*sin(gamma[i][sonar_column_counter]));//store
current y position of sonar (y of robot + y of sonar on device)
            sonar_x[i][sonar_column_counter]= robot-
>getSonarReading(i)-
>getXTaken()+(sensor_diag*cos(gamma[i][sonar_column_counter]));//store
current x position of sonar (x of robot + x of sonar on device)


son_table<<sonar_x[i][sonar_column_counter]<<"
"<<sonar_y[i][sonar_column_counter]<<"
"<<gamma[i][sonar_column_counter]<<"
"<<sonar_range[i][sonar_column_counter]<<"  ";

            //get the "expected" obstacle (fron sonar beam midpoint)
for each sensor

      y_t[i][sonar_column_counter]=sonar_y[i][sonar_column_counter]+(so
nar_range[i][sonar_column_counter]*sin(gamma[i][sonar_column_counter]))
;//get y position of point

      x_t[i][sonar_column_counter]=sonar_x[i][sonar_column_counter]+(so
nar_range[i][sonar_column_counter]*cos(gamma[i][sonar_column_counter]))
;//get x position of point

      }

      sonar_column_counter++;// increment counter
son_table<<";"<<"\n";

}
```

Appendix B.2: max dist function code

In TBF code, a function was developed to work out the maximum deviations between successful triangulation. The function was called max_dst. The function compares every corner in the array to other corners and finds the distance between the corners. It stores the maximum distance. If the maximum distance is greater than 0.1m then it returns this value and terminates the algorithm.

The code is given below.

```
function max=max_dst(arry)%function for finding the maximum distance
between elements of an array (x,y)

if(size(arry,1)<2)% if the array is less than two elements in size
max=0; %set max to zero
end



if(size(arry,1)>1)%if more than 1 element in array
max=0;%initialise max
   i=1;%loop index
   j=1;%loop index
while (i<size(arry,1)+1)
while (j<size(arry,1)+1)
dst=sqrt(((arry(i,1)-arry(j,1))^2)+((arry(i,2)-arry(j,2))^2)); %find
distance

if(dst>max)%if dist is greater than max then set max to distance
max=dst;
end

if(max>0.1)%if max is greater than 10 cm then quit loop to save time
             i=20;
             j=20;
end
         j=j+1;%increment counter
end
       i=i+1;%increment counter
end

end
end
```

Appendix B.3: Circcirc function code from matlab

A code to plot two circles then find and plot the intersection points for comparison with theoretical solution.

```
figure(1)
th = 0:pi/50:2*pi;%complete circle
xunit = 100 * cos(th) + 100;%circle x
yunit = 100 * sin(th) + 100;%circle y
```

```
plot(xunit, yunit,'r');%plot first circle of radius 100 and center
(100,100)
hold on
%second circle
th = 0:pi/50:2*pi;
xunit = 300 * cos(th) +  00;
yunit = 300 * sin(th) +  00;
plot(xunit, yunit,'b');
figure(1)
th = 0:pi/50:2*pi;
xunit = 400 * cos(th) + 100;
yunit = 400 * sin(th) + 100;
plot(xunit, yunit,'r');%plot second circle of radius 400 and center
(0,0)
hold on
%th = 0:pi/50:2*pi;
%xunit = 300 * cos(th) +  00;
%yunit = 300 * sin(th) +  00;
%plot(xunit, yunit,'b');
%title('Intersection of two cicles')
xlabel('x axis')
ylabel('y axis')
axis([-700 700 -700 700])
axis([-700 700 -700 700])
plot(0,0,'*b')%plot centers of circles
plot(100,100,'*r')
[xout,yout] = CIRCCIRC(0,0,300,100,100,400)%find intersection point
plot(xout(1),yout(1),'*k',xout(2),yout(2),'*k')%plot it
```

Appendix B.4: TBF algorithm code

```
nc=1;%hypothesis counter
figure(2)
for l=2:43%to cycle through last sonar (last column in sliding window
table)

forjj=1:size(fg,2)/4 %for each sonar in the last column of sliding
window table
nc=1;%set counter to 1
for j=1:size(fg,2)/4%for each sonar in table
for i=1:size(fg,1)-45+l+0%for each column except last one

intersect=180*(fg(l+1,3+(4*(jj-1)))-fg(i,3+(4*(j-1))))/3.142;% the
intersection angle between two candidtate sonars for TBF
fgx1=fg(l+1,1+(4*(jj-1)))+(cos(fg(l+1,3+(4*(jj-1))))*fg(l+1,4+(4*(jj-
1))));%x ordinate of first sonar reading
fgy1=fg(l+1,2+(4*(jj-1)))+(sin(fg(l+1,3+(4*(jj-1))))*fg(l+1,4+(4*(jj-
1))));%y ordinate of first sonar reading
```

```
fgx2=fg(i,1+(4*(j-1)))+(cos(fg(i,3+(4*(j-1))))*fg(i,4+(4*(j-1))));%x
ordinate of second sonar reading
fgy2=fg(i,2+(4*(j-1)))+(sin(fg(i,3+(4*(j-1))))*fg(i,4+(4*(j-1))));%y
ordinate of second sonar reading
angrng=(atan2(fgy2-fg(l+1,2+(4*(jj-1))),fgx2-fg(l+1,1+(4*(jj-
1)))))*180/3.142;%to check whether within sonar range
dist=sqrt(((fgx1-fgx2 )^2)+((fgy1-fgy2)^2));%distance between readings


if(abs(angrng-(fg(l+1,3+(4*(jj-1)))*180/3.142))<20&&dist<300)%if within
sonar cone

%set parameters for TBF algorithm radii of circles are the range of
%sonars, and centres of circles are postions of sonars
r1=fg(i,4+(4*(j-1)));
r2=fg(l+1,4+(4*(jj-1)));

x1=fg(i,1+(4*(j-1)));
x2=fg(l+1,1+(4*(jj-1)));

y1=fg(i,2+(4*(j-1)));
y2=fg(l+1,2+(4*(jj-1)));


dx=x1-x2;
dy=y1-y2;


ds2=(dx^2)+(dy^2);
%dr2=((r1^2)-(r2^2)-ds2)/2;
k=(ds2+(r1^2)-(r2^2))/(2*sqrt(ds2));

%result of first solution use angle criterion to filter wrong solution
xtt=x1-(dx*k/sqrt(ds2))+((-dy/sqrt(ds2))*sqrt((r1^2)-(k^2)));
ytt=y1-(dy*k/sqrt(ds2))-((-dx/sqrt(ds2))*sqrt((r1^2)-(k^2)));

angrng= (atan2(ytt-fg(l+1,2+(4*(jj-1))),xtt-fg(l+1,1+(4*(jj-
1)))))*180/3.142;
faill=0;%flag for failing triangulation

if(abs(angrng-(fg(l+1,3+(4*(jj-1)))*180/3.142))>20)%if first solution
out of sonar cone work out second solution
xtt=x1-(dx*k/sqrt(ds2))-((-dy/sqrt(ds2))*sqrt((r1^2)-(k^2)));
ytt=y1-(dy*k/sqrt(ds2))+((-dx/sqrt(ds2))*sqrt((r1^2)-(k^2)));

angrng= (atan2(ytt-fg(l+1,2+(4*(jj-1))),xtt-fg(l+1,1+(4*(jj-
1)))))*180/3.142;
```

```
if(k^2>r1^2)% if not possible to triangulate set fial flag
faill=1;
end

if(abs(angrng-(fg(l+1,3+(4*(jj-1)))*180/3.142))>20)%if no solution
within cone then set fail flag
faill=1;
end

end


if(faill==0)% if not fail find distance between reading of sonar nd
triangulation
sep=sqrt(((xtt-fgx1)^2)+((ytt-fgy1)^2));


if (sep<300/nc&&abs(intersect)>40&&abs(r1-r2)>500)%if matching criteria
increment counter of hypothesis
nc=nc+1;


end
end
end
end

if(nc>1)
figure(2)
holdall
plot(xtt,ytt,'*')
end


end

end
%pause
end
```

Appendix B.5 Plotting code

```
%sonar readings accepted ffrom array FG in the format [sonar x
position,
```

```
%sonar y position, sonar orientation, sonar range] where each row
%represents a time step
figure(2)
holdon%to plot on same handle
for i=1:size(fg,1)% for each time step i.e. column in table
for j=1:size(fg,2)/4%for each sonar
plot(fg(i,1+(4*(j-1))),fg(i,2+(4*(j-1))),'.r')%plot the position of
sonar sensor
fgx=fg(i,1+(4*(j-1)))+(cos(fg(i,3+(4*(j-1))))*fg(i,4+(4*(j-1))));%x
ordinate of sonar reading
fgy=fg(i,2+(4*(j-1)))+(sin(fg(i,3+(4*(j-1))))*fg(i,4+(4*(j-1))));%y
ordinate of sonar reading
plot(fgx,fgy,'.k')%plot sonar reading
if(fg(i,4+(4*(j-1)))>4990)%if likely to be specula reflection
plot(fg(i,1+(4*(j-1))),fg(i,2+(4*(j-1))),'.b')%plot specualr sonar in
red
plot(fgx,fgy,'.c')%and reading in cyan
end
end
pause
end
```

Appendix C

Line features related code

Appendix C.1: Line Segmentation algorithm

```
vector<double> segmentLaser(void)
{ofstream seg;seg.open("segger.txt",ios_base::app);
seg<<"\n"<<"new call to segment laser function "<<"\n";
    double p1[181][1];//to hold the log oflaser readings
      double dr1[181][1];//to hold first derivative
      double dr2[181][1];//to hold second derivative
      double max_dr1=0;//to hold maximumof dr1

      seg<<"log readings"<<"\n";
        for (int i=0; i<181; i++)
        {
        p1[i][0]=log(laser_range[i][0]);//take ln of reading
        seg<<p1[i][0]<<"  ";
        }
 seg<<"\n";seg<<"first d"<<"\n";
        dr1[0][0]=(-p1[2][0]+(4*p1[1][0])-(3*p1[0][0]))/2;//first
derivative of first element
     seg<<dr1[0][0]<<"  ";
        max_dr1=abs(dr1[0][0]);//set first reading to max
        for (int i=1; i<180; i++)//first derivative of the rest of
elements
        {
            dr1[i][0]=(p1[i+1][0]-p1[i-1][0])/2;
            max_dr1=max(max_dr1,abs(dr1[i][0]));//find maximum of the
two
            seg<<dr1[i][0]<<"  ";
        }
        dr1[180][0]=(p1[180-2][0]-(4*p1[180-
1][0])+(3*p1[180][0]))/2;//derivative of the last element
         seg<<dr1[180][0]<<"\n";

        max_dr1=57.2/max(max_dr1,abs(dr1[180][0]));

seg<<"atan"<<"\n";
        for (int i=0; i<181; i++)
        {
        p1[i][0]=atan(max_dr1*dr1[i][0]);//find atan(mdr*dr)
        seg<<p1[i][0]<<"  ";
        }
        seg<<"\n";seg<<"derv 2 "<<"\n";
```

```
        dr2[0][0]=(-p1[2][0]+(4*p1[1][0])-(3*p1[0][0]))/2;//second
derivative of first element
         seg<<dr2[0][0]<<"  ";

        for (int i=1; i<180; i++)//second derivative of the rest of
elements
        {
            dr2[i][0]=(p1[i+1][0]-p1[i-1][0])/2;
             seg<<dr2[i][0]<<" ";
        }
        dr2[180][0]=(p1[180-2][0]-(4*p1[180-
1][0])+(3*p1[180][0]))/2;//derivative of the last element
          seg<<dr2[180][0]<<"\n";



vector<double> shod;//array to hold peak points
int tr=0; //key to switch between if conditions
seg<<"shod array now i  "<<0<<"\n";
for (int i=0;i<181;i++)
{
   if(abs(dr2[i][0])>0.1)//if value is peak
   {
      shod.push_back(dr2[i][0]);//set value in array

   }
   else
   {
       shod.push_back(0);//else put zero

   }   seg<<i<<"  "<<shod[i]<<" ";
}

for (int i=1;i<179;i++)//for all array
{seg<<i<<"  "<<shod[i]<<"  "<<shod[i+1]<<"\n";
    if(shod[i]>0&&shod[i+1]<0)//if one point and the other are opposite
polarity
    {
    tr=1;//set key
    }

    if(shod[i]<0&&shod[i+1]>0)// if the other way around set key
    {
        tr=1;
    }

   if(tr==1&&i<179)//added second condition //if key is set i.e. zero
crossing
```

```
   {
      shod[i-1]=0;//shod(i-1)=0;
      shod[i]=10;//shod(i)=10;%was 10
      shod[i+1]=0;//shod(i+1)=0;%set to 10 for diff 10;
      //shod[i+2]=0;//shod(i+2)=0;
      tr=0;
   }
}

for (int h=0;h<shod.size();h++)
{
seg<<shod[h]<<"  ";

}seg<<"\n";
int i=0;//%loop index
int mm=0;//%holder of previous index
int st=0;//holder of index copy
while(i<179)
{seg<<i<<"\n";
    int m=0;//%set m to zero
    if(abs(shod[i])>0)
    {
        m=1;//if first point is found set m to 1
        mm=i;//hold current index in mm
        st=i;//hold another copy in st

        while(shod[i]!=0&&shod[i]!=10&&i<179)//while point is
disvontinouity and not an edge
        {
            if(abs(shod[i+1])>abs(shod[mm])) //this is to find mono
peak
            {
                mm=i+1;
            }
                i=i+1;

        }

        for (int k=st;k<i;k++)//for range of peaks
        {
           shod[k]=0;
        }

        shod[mm]=10;


}
    i=i+1;
```

```
}


for (int h=0;h<shod.size();h++)
{
seg<<shod[h]<<"  ";

}seg<<"\n";

////////////////////////////////////////////////////
int rt=0;//%this is the root value
int knk=2;//%this is the corner (knee) value
int tp=0;//%this is the top VAL
int prcss=0;//%this is a semaphore to start processing data
int ind1=1;//%this indicates that knee value has been obtained
int ind2=0;//%this indicates whether top value has been obtained
int ll=0;//%this is to indicate if the point satissfies a line
 i=0;//%this is the loop index
 seg<<"second while "<<"\n";
while(i<181)
{
   if(shod[i]==10&&ind1==1) //%while first line not obtianed yet
   {
    knk=i;//%get knee
    ind2=1;//%prepare for next line
    ind1=0; i=i+1;//%increment counter
   }

   if(shod[i]==10&&ind2==1)//%once first line is obtained, find second
   {
      tp=i;
      ind2=0;// %consider setting later
      prcss=1;//%start processing the line segments
   }

   if(prcss==1)
   {



double a1=atand(((laser_range[knk-1][0]*sind(knk-1))-
(laser_range[rt+1][0]*sind(rt+1)))/((laser_range[knk-1][0]*cosd(knk-
1))-(laser_range[rt+1][0]*cosd(rt+1))));//%this is the gradient of
first lline
double b1=atand(((laser_range[tp-1][0]*sind(tp-1))-
(laser_range[knk+1][0]*sind(knk+1)))/((laser_range[tp-1][0]*cosd(tp-
1))-(laser_range[knk+1][0]*cosd(knk+1))));//%this is the gradient of
second lline
```

```
if(a1<0){a1=a1+180;}
if(b1<0){b1=b1+180;}
 // i added the following modification
if(knk-1==rt+1&&knk+1!=tp-1)
{
a1=b1;
}
if(knk-1!=rt+1&&knk+1==tp-1)
{
b1=a1;
}
if(knk-1==rt+1&&knk+1==tp-1)
{
a1=0;b1=0;
}
//end modification
double a=sqrt(pow(((laser_range[knk][0]*sind(knk))-
(laser_range[rt+1][0]*sind(rt+1))),2)+pow((atand(laser_range[knk][0]*co
sd(knk))-atand(laser_range[rt+1][0]*cosd(rt+1))),2));//%this is a
distance
double b=sqrt(pow(((laser_range[tp][0]*sind(tp))-
(laser_range[knk+1][0]*sind(knk+1))),2)+pow((atand(laser_range[tp][0]*c
osd(tp))-atand(laser_range[knk+1][0]*cosd(knk+1))),2));//%this is b
distance
double c=sqrt(pow(((laser_range[tp][0]*sind(tp))-
(laser_range[rt+1][0]*sind(rt+1))),2)+pow((atand(laser_range[tp][0]*cos
d(tp))-atand(laser_range[rt+1][0]*cosd(rt+1))),2));//%this is c
distance
double aa=sqrt(pow(((laser_range[knk-1][0]*sind(knk-1))-
(laser_range[knk+1][0]*sind(knk+1))),2)+pow(((laser_range[knk-
1][0]*cosd(knk-1))-(laser_range[knk+1][0]*cosd(knk+1))),2));//%this is
aa distance

//DEBUG
seg<<"\n"<<" points  "<<laser_range[knk-1][0]*sind(knk-1)<<"
"<<laser_range[rt+1][0]*sind(rt+1)<<laser_range[knk-1][0]*cosd(knk-
1)<<" "<<laser_range[rt+1][0]*cosd(rt+1)<<"\n";
 seg<<"\n"<<" rt knk tp a1 b1 aa "<<rt<<" "<<knk<<" "<<tp<<" "<<a1<<"
"<<b1<<" "<<aa<<"\n";
if(abs(a1-b1)<=10)//%if gradient difference is less than 10 deg
{
if(aa<100)//(abs(aa+a+b)/c)<=1.05)//%distance criterion less than
1.05//DEBUG added =
{
    ll=1;//%set to same line
    seg<<"entered ll=1 due to gradient "<<"\n";
}
//}
else if(aa>100)//(abs(aa+a+b)/c)>1.049)
```

```
{
    ll=0;//%set to different line
}
}

if(abs(a1-b1)>15)//%if gradient greater than 10
{
//if(aa>100)//(abs(aa+a+b)/c)>1.049)
{
     ll=0;
}
//}
//else if(aa<100)//(abs(aa+a+b)/c)<1.05)
{
  //  ll=1;seg<<"entered ll=1 not due to gradient "<<"\n";
}
}



   if(ll==1)//%if two segemtns are acctualy false segements (due to
noise)
   {
       shod[knk]=0;
    knk=tp;
    ind2=1;
    ind1=0;//was added now
    prcss=0;
   }



    //%if not line%%%%%%%
    if(ll==0)
    {
        rt=knk;
        knk=tp;
        ind2=1;
        ind1=0;
        prcss=0;
    }

    prcss=0;ll=0;
   }
  i=i+1;
}
shod[0]=10;shod[180]=10;seg<<"shod is "<<"\n";
for (int h=0;h<shod.size();h++)
{
seg<<shod[h]<<"  ";
```

```
}
return shod;
}
```

Appendix C.2: Line extraction algorithms

```
vector<double> LineFit(vector<double> shod)
{ofstream linefitl;linefitl.open("fitter.txt",ios_base::app);
linefitl<<"new call to function"<<"\n"<<"\n";
    unsigned int i=0;//%array index for while loop
vector<double> mss;//%array for gradients
vector<double> bound;//array for points vector<double> css;//%array fpr
y intercept
linefitl<<"i  "<<i<<"\n";
int wz=0;
i=1;
 int n=0;//%points counter
   double xm;
 double ym ;
 double ys=0;//%y
 double xs=0;//%x
 double x=0;double y=0;
 double m1=0;//temp gradient
  double m2=0;//temp y intercept
 double c0=0;//temp y intercept
  double c1=0;//temp y intercept
   double c2=0;//temp y intercept
  double e1=0;
  double e2=0;
 int flg=0;//%flag to act as semaphore
while (i<shod.size())
{
linefitl<<"entered"<<"   i = "<<i<<"\n";
  n=0;//%points counter

  ys=0;//%y
  xs=0;//%x
  x=0;y=0;
  m1=0;//temp gradient
  m2=0;//temp y intercept
  c0=0;//temp y intercept
  c1=0;//temp y intercept
  c2=0;//temp y intercept
  e1=0;
  e2=0;
  flg=0;//%flag to act as semaphore
```

```
while(shod[i]!=10&&i<shod.size())//%while point is within line
boundaries
{linefitl<<"entered while "<<i<<"\n";
    flg=1;//%set flag
    xs=xs+(laser_range[i][0]*cosd(i));//sum of xs
    ys=ys+(laser_range[i][0]*sind(i));// sum of ys
    i=i+1;
}
linefitl<<"n wz i "<<n<<"  "<<wz<<" "<<i<<"\n";
 n=i-wz-1;
   xm=xs/(i-wz-1);
   ym=ys/(i-wz-1);

int nn=sqrt(pow((laser_range[wz+1][0]*cosd(wz+1))-(laser_range[i-
1][0]*cosd(i-1)),2)+pow((laser_range[wz+1][0]*sind(wz+1))-
(laser_range[i-1][0]*sind(i-1)),2));



if(flg==1&&nn>100&&n>2)//4 %if line detected and more than 4 points
{
linefitl<<"entered for "<<"\n";linefitl<<"xm ym "<<xm<<" "<<ym<<"\n";

for (int ft=wz+1;ft<i;ft++)
{
   x=(laser_range[ft][0]*cosd(ft));
   y=(laser_range[ft][0]*sind(ft));
   c2=c2+((x-xm)*(y-ym));
   c1=c1+pow((x-xm),2)-pow((y-ym),2);
   linefitl<<" n"<<ft<<"\n";
}



c0=-1*c2;linefitl<<" c0 c1 c2 "<<c0<<" "<<c1<<" "<<c2;
m1=((-c1/c2)+sqrt(pow(c1/c2,2)+4))*0.5;
m2=((-c1/c2)-sqrt(pow(c1/c2,2)+4))*0.5;

c1=ym-(m1*xm);
c2=ym-(m2*xm);
linefitl<<"line "<<c1<<" "<<c2<<" "<<m1<< " "<<m2<<"\n";
//find error
for (int ft=wz+1;ft<i;ft++)
 {
    x=(laser_range[ft][0]*cosd(ft));
    y=(laser_range[ft][0]*sind(ft));
    e1=e1+((pow(y-c1-(m1*x),2))/(1+(m1*m1)));
    e2=e2+((pow(y-c2-(m2*x),2))/(1+(m2*m2)));
```

```
}
    e1=sqrt(e1/(i-wz-1));
    e2=sqrt(e2/(i-wz-1));
linefitl<<"error "<<e1<<" "<<e2<<"\n";
    if (e1<e2&&e1<150)
    {
    mss.push_back(m1);
    mss.push_back(c1);mss.push_back(wz+1);mss.push_back(i);
    }

    if(e2<e1&&e2<150)
    {
    mss.push_back(m2);
    mss.push_back(c2);mss.push_back(wz+1);mss.push_back(i);
    }

    if(e1==e2&&e2<150)
    {
    mss.push_back(m1);
    mss.push_back(c1);mss.push_back(wz+1);mss.push_back(i);
    }

wz=i;
flg=0;
}
else
{wz=i;}

i=i+1;//%increment loop counter
}

//process before returning
linefitl<<"\n"<<" now processing "<<"\n";
for (int q=0;q<(mss.size()/4)-1;q++)
{
 int ptr=mss[(4*(q))+3]-1;
 int ptr0=mss[(4*(q))+2];
 int ptr2=mss[(4*(q+1))+2];
 int ptr3=mss[(4*(q+1))+3]-1;
 double grd1= atand(mss[(4*(q))+0]);
 double grd2= atand(mss[(4*(q+1))+0]);
 double xx=(laser_range[ptr][0]*cosd(ptr));
 double yy=(laser_range[ptr][0]*sind(ptr));
 double xx2=(laser_range[ptr2][0]*cosd(ptr2));
 double yy2=(laser_range[ptr2][0]*sind(ptr2));
 double sep=sqrt(pow(xx-xx2,2)+pow(yy-yy2,2));
if(grd1<0){grd1=grd1+180;}
if(grd2<0){grd2=grd2+180;}
```

```
 if(sep<100&&abs(grd1-grd2)<5)
 {
 c1=0;c2=0;
  xs=0;ys=0;e1=0; e2=0;

  for(int jk=ptr0;jk<ptr3+1;jk++)
  {
    xs=xs+(laser_range[jk][0]*cosd(jk));//sum of xs
    ys=ys+(laser_range[jk][0]*sind(jk));// sum of ys
   }


n=ptr2-ptr;
xm=xs/n;
ym=ys/n;
 for (int ft=ptr0;ft<ptr3+1;ft++)
{
   x=(laser_range[ft][0]*cosd(ft));
   y=(laser_range[ft][0]*sind(ft));
   c2=c2+((x-xm)*(y-ym));
   c1=c1+pow((x-xm),2)-pow((y-ym),2);

}



c0=-1*c2;
m1=((-c1/c2)+sqrt(pow(c1/c2,2)+4))*0.5;
m2=((-c1/c2)-sqrt(pow(c1/c2,2)+4))*0.5;

c1=ym-(m1*xm);
c2=xm-(m2*xm);

//find error
for (int ft=ptr0;ft<ptr3+1;ft++)
 {
    x=(laser_range[ft][0]*cosd(ft));
    y=(laser_range[ft][0]*sind(ft));
    e1=e1+((pow(y-c1-(m1*x),2))/(1+(m1*m1)));
    e2=e2+((pow(y-c2-(m2*x),2))/(1+(m2*m2)));
 }
    e1=sqrt(e1/(n));
    e2=sqrt(e2/(n));
 linefitl<<" df "<<e1<<" "<<e2<<" n "<<n<<" ptr ptr2 "<<ptr<<"
"<<ptr2<<"\n";
    if (e1<e2&&e1<150)
    {
    bound.push_back(m1);
```

```
    bound.push_back(c1);
    bound.push_back(e1);
    bound.push_back(ptr0);
    bound.push_back(ptr3);
    mss[(4*(q+1))+0]=m1;mss[(4*(q+1))+1]=c1;
    mss[(4*(q+1))+2]=ptr0;//mss[(4*(q+1))+3]=ptr2;
    linefitl<<m1<<" "<<c1<<" MERGED "<<"\n";
    }

    if(e2<e1&&e2<150)
    {
    bound.push_back(m2);
    bound.push_back(c2);
    bound.push_back(e2);
    bound.push_back(ptr0);
    bound.push_back(ptr3);
    linefitl<<m2<<" "<<c2<<" MERGED "<<"\n";
    mss[(4*(q+1))+0]=m1;mss[(4*(q+1))+1]=c1;
    mss[(4*(q+1))+2]=ptr0;//mss[(4*(q+1))+3]=ptr2;
    }

    if(e1==e2&&e2<150)
    {
    bound.push_back(m1);
    bound.push_back(c1);
    bound.push_back(e1);
    bound.push_back(ptr0);
    bound.push_back(ptr3);
    linefitl<<m1<<" "<<c1<<" MERGED "<<"\n";
    mss[(4*(q+1))+0]=m1;mss[(4*(q+1))+1]=c1;
    mss[(4*(q+1))+2]=ptr0;//mss[(4*(q+1))+3]=ptr2;
    }
}
    else
    {
    bound.push_back(mss[(4*(q))+0]);
    bound.push_back(mss[(4*(q))+1]);
    e1=0;m1=mss[(4*(q))+0];c1=mss[(4*(q))+1];n=0;
    for (int ft=ptr0;ft<ptr+1;ft++)
    {
        n=n+1;
    x=(laser_range[ft][0]*cosd(ft));
    y=(laser_range[ft][0]*sind(ft));
    e1=e1+((pow(y-c1-(m1*x),2))/(1+(m1*m1)));
    }
    e1=sqrt(e1/(n));
    bound.push_back(e1);
    bound.push_back(ptr0);
    bound.push_back(ptr);
```

```
  //  bound.push_back(mss[(4*(q+1))+0]);
  //  bound.push_back(mss[(4*(q+1))+1]);
    linefitl<<mss[(4*(q))+0]<<"
"<<mss[(4*(q))+1]<<"\n";//<<mss[(4*(q+1))+0]<<"
"<<mss[(4*(q+1))+1]<<"\n";


 }
 if(q==((mss.size()/4)-2))
    {bound.push_back(mss[(4*(q+1))+0]);
     bound.push_back(mss[(4*(q+1))+1]);
     e1=0;m1=mss[(4*(q+1))+0];c1=mss[(4*(q+1))+1];n=0;
    for (int ft=mss[(4*(q+1))+2];ft<mss[(4*(q+1))+3];ft++)
    {
        n=n+1;
    x=(laser_range[ft][0]*cosd(ft));
    y=(laser_range[ft][0]*sind(ft));
    e1=e1+((pow(y-c1-(m1*x),2))/(1+(m1*m1)));
    }
    e1=sqrt(e1/(n));

    bound.push_back(e1);
    bound.push_back(mss[(4*(q+1))+2]);
    bound.push_back(mss[(4*(q+1))+3]-1);
     linefitl<<mss[(4*(q+1))+0]<<" "<<mss[(4*(q+1))+1]<<"\n";
    }

}
linefitl<<"done "<<bound.size();

return bound;
```

Appendix D

Scan matching algorithm

```
vector<double> scn_mtch(double dx,double dy,double dth, ArLaser *laser)
{



    //if no prior scan obtained
    if(acquired==0)//acquired global variable indicates that no
previous scan existed i.e. number of acquired scans is zero
    {

diff[0]=dx;diff[1]=dy;diff[2]=dth;diff[3]=01;diff[4]=01;diff[5]=1;//if
no previous scan exists, return the current odometry variables with the
errors specified
        getLaser1(laser);//read laser data
        shod_G=segmentLaser(); //segement laser data to get the points
where corners exist
        outpt<<"THe laser readings  "<<laser_range[0][1]<<"
"<<laser_range[180][1]<<"start"<<"\n";
        for (int of=0;of<181;of++)//set the previous scan as the
current scan (for next cycle)
        {
            prev_scan[of][0]=laser_range[of][0];//set cuurent range as
prev range
            outpt<<prev_scan[of][0]<<"  ";
            prev_scan[of][1]=laser_range[of][1];//set current angles of
ranges as previous
        }
        outpt<<"shod_G"<<"\n";
        for (int of=0;of<181;of++)
        {
            outpt<<shod_G[of]<<"   ";
        }
    }
    outpt<<"\n";
    //if priorscan obtained
    if(acquired==1)
    {
        outpt<<"new scan";
        getLaser1(laser);//get the new laser data
        vector<double> shod=segmentLaser();//segment the new data and
store in array
        ///for plotting
        for (int of=0;of<181;of++)
        {
            outpt<<laser_range[of][0]<<"   ";
        }
```

```
        outpt<<"\n";
        outpt<<"shod_new"<<"\n";
        for (int of=0;of<181;of++)
        {
            outpt<<shod[of]<<"   ";
        }
        outpt<<"\n";

        unsigned int i=0;//%array index
        vector< vector<double> > rel1;//%relations array for scan 1
        vector<double> rel1a;//row for intermediate operation
        vector< vector<double> > rel2;//%relations array for scan 2
        vector<double> mylines;//array to get lines from new scan
        vector<double> mylines2;//array to get lines from old scan

        mylines2=LineFit(shod_G);
        mylines=LineFit(shod);

        for (int yi=0;yi<mylines2.size()/5;yi++)//for old/previous scan
        {
            int strt;int fnl;double mr;double ee;double cc;double
sep;double x;double y;
            strt=mylines2[(5*(yi))+3];//record starting point
            fnl=mylines2[(5*(yi))+4];//record final line point
            mr=mylines2[(5*(yi))+0];//record gradient of line
            cc=mylines2[(5*(yi))+1];//record intercept (y) of lines
            ee=mylines2[(5*(yi))+2];//record error in line
            x=(prev_scan[strt][0]*cosd(prev_scan[strt][1]))-
(prev_scan[fnl][0]*cosd(prev_scan[fnl][1]));//difference in x
            y=(prev_scan[strt][0]*sind(prev_scan[strt][1]))-
(prev_scan[fnl][0]*sind(prev_scan[fnl][1]));//difference in y
            sep=sqrt(pow(x,2)+pow(y,2));//line length
            rel1a.clear();

rel1a.push_back(strt);rel1a.push_back(fnl);rel1a.push_back(sep);rel1a.p
ush_back(mr);rel1a.push_back(ee);rel1a.push_back(cc);//%record the
starting and final index of the line as well as the distance  added
intercept y at end
            rel2.push_back(rel1a);outpt<<"\n"<<"rel2 old scan row  str
fnl   sep  mr   ee cc "<<strt<<" "<<fnl<<" "<<sep<<" "<<atand(mr)<<"
"<<ee<<" "<<cc<<"\n";
        }

        for (int yi=0;yi<mylines.size()/5;yi++)//for new/recent scan
        {
            int strt;int fnl;double mr;double ee;double cc;double
sep;double x;double y;
            strt=mylines[(5*(yi))+3];//record starting point
            fnl=mylines[(5*(yi))+4];//record final line point
```

```
            mr=mylines[(5*(yi))+0];//record gradient of line
            cc=mylines[(5*(yi))+1];//record intercept (y) of lines
            ee=mylines[(5*(yi))+2];//record error in line
            x=(laser_range[strt][0]*cosd(laser_range[strt][1]))-
(laser_range[fnl][0]*cosd(laser_range[fnl][1]));//difference in x
            y=(laser_range[strt][0]*sind(laser_range[strt][1]))-
(laser_range[fnl][0]*sind(laser_range[fnl][1]));//difference in y
            sep=sqrt(pow(x,2)+pow(y,2));//line length
            rel1a.clear();

rel1a.push_back(strt);rel1a.push_back(fnl);rel1a.push_back(sep);rel1a.p
ush_back(mr);rel1a.push_back(ee);rel1a.push_back(cc);//%record the
starting and final index of the line as well as the distance  added
intercept y at end
            rel1.push_back(rel1a);outpt<<"\n"<<"rel1 new scan row  str
fnl   sep  mr   ee cc "<<strt<<" "<<fnl<<" "<<sep<<" "<<atand(mr)<<"
"<<ee<<" "<<cc<<"\n";

        }
        outpt<<"\n"<<"pairs  ";

////////////////////////////////////////////////////////////////////////
//////////////////////////////////////find pairs
        vector< vector<double> > pairs;//%empty array for pairing
points

        for (unsigned int i=0;i<rel1.size();i++)//%for all elements of
relation array
        {
            unsigned int j=0;//%initialise index for array
            while (j<rel2.size())
            {
                double sse=max(rel1[i][4],rel2[j][4]);//find max error
                int len=rel1[i][1]-rel1[i][0];//interval
                int len2=rel2[j][1]-rel2[j][0];//interval of second
line
                double clsf=abs(((rel1[i][2]/len)/(rel2[j][2]/len2))-
1);//length of line divided by interval
                double
dmagi=abs((rel1[i][5]*cosd(abs(atand(rel1[i][3]))))-
(rel2[j][5]*cosd(abs(atand(rel2[j][3])))));//difference in distance to
line pairs
                outpt<<"i "<<i<<" j  "<<j<<" clsf "<<clsf<<"size j
"<<rel2.size()<<" "<<dmagi<<" "<<dx<<"\n";
                if(clsf<=0.1&&dmagi<abs(4*dy)&&abs(rel1[i][1]-
rel2[j][1])<25)//sse<200)//if possible match found
                {
                    outpt<<" match found";
                    vector<double> pairsa;
```

```
pairsa.push_back(rel1[i][0]);pairsa.push_back(rel2[j][0]);pairsa.push_b
ack(rel1[i][3]);pairsa.push_back(rel2[j][3]);pairsa.push_back(rel1[i][5
]);pairsa.push_back(rel2[j][5]);pairsa.push_back(rel1[i][4]);pairsa.pus
h_back(rel2[j][4]);//%group matched pairs  str, mr, intercept error
                    outpt<<"\n"<<" line pair "<<pairs.size()<<"
{str(1,2) mr(1,2) intercept(1,2)}: "<<rel1[i][0]<<"  "<<rel2[j][0]<<"
"<<rel1[i][3]<<"  "<<rel2[j][3]<<"  "<<rel1[i][5]<<"  "<<rel2[j][5];
                    pairs.push_back(pairsa);
                    rel2.erase(rel2.begin()+j);//%remove matched pair
                    j=180;//%terminate while loop
                }
                outpt<<"\n";
                j=j+1 ; //%increment counter if didnot find match
            }
        }

////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////finding theta/////
        ////find theta
        ofstream ash;ash.open("debugger.txt",ios_base::app);
        vector<double> x1;
        vector<double> y1;
        vector<double> x2;
        vector<double> y2;

        for (int h=0;h<181;h++)
        {
            x1.push_back(laser_range[h][0]*cosd(laser_range[h][1]));
            y1.push_back(laser_range[h][0]*sind(laser_range[h][1]));
            x2.push_back(prev_scan[h][0]*cosd(prev_scan[h][1]));
            y2.push_back(prev_scan[h][0]*sind(prev_scan[h][1]));
        }
        outpt<<"finding th  "<<"\n";
        double detr=0;
        double rot=dth;//% for rotation values
        double sumRot=0;//
        double wgt;//% for weight
        double sumWgt=0;//sum of weihts
        if(pairs.size()>0)
        {
            //DEBUG added
            double certn=0;
            //added fr least squares method
            vector<double> wghts;vector<double> UKFCO;ash<<"\n"<<"new
rotation"<<"\n";
            for (unsigned int ii=0;ii<pairs.size();ii++)//%for all
pairs
            {
```

```
if(atand(pairs[ii][2])<0&&!(atand(pairs[ii][3])<0)){pairs[ii][2]=atand(
pairs[ii][2])+180;}//to avoid wrong quadrant

if(atand(pairs[ii][3])<0&&!(atand(pairs[ii][2])<0)){pairs[ii][3]=atand(
pairs[ii][3])+180;}
                rot= atand(pairs[ii][2])-
atand(pairs[ii][3]);outpt<<"\n";UKFCO.push_back(-1*rot);outpt<<" line
"<<rot<<" details 1 2 "<<atand(pairs[ii][2])<<"
"<<atand(pairs[ii][3]);
                wgt=exp(-1*pow(-rot-dth,2)/9);//consider narrowing the
gaussian
                detr=max(detr,wgt);
                certn=certn+wgt;
                sumWgt=sumWgt+(wgt*wgt);
                sumRot=sumRot+(wgt*rot*wgt);outpt<<"\n"<<"rot
"<<rot<<"  wgt   "<<wgt;wghts.push_back(wgt);
             }
             double meanth=0;
             double covth=0;
             for (unsigned int
ty=0;ty<UKFCO.size();ty++){meanth=meanth+UKFCO[ty];}meanth=meanth/UKFCO
.size();outpt<<"\n"<<" avg th "<<meanth<<" covth ";
             for (unsigned int
ty=0;ty<UKFCO.size();ty++){covth=covth+pow(meanth-UKFCO[ty],2);}
outpt<<covth;
             certn=certn/pairs.size();
             rot=-1*sumRot/(sumWgt);//robot rotation
             //if(detr<0.1){rot=dth;certn=0.1;}
             diff[2]=rot;outpt<<" actual "<<rot<<" dth from robot
"<<dth<<" detr "<<detr;
             diff[5]=9;//covth;//0.5/certn;

////////////////////////////////////////////////////////////////////
//////////////////////find y intercept//////////////////////////////
          /*  vector<double> cexp;//expected y intercept
             double c1=0;
             double c2=0;
             double c0=0;
             double m1;
             double m2;
             double mbar=0;
             double dcbar=0;
             double e1=0;
             double e2=0;

             for (unsigned int ii=0;ii<pairs.size();ii++)//%for all
pairs
             {
```

```
                double resul;

resul=pairs[ii][5]*(cosd(rot)+(sind(rot)*pairs[ii][2]));
                cexp.push_back(resul);
                /*if(wghts[ii]>0.7)
                {

                mbar=mbar+pairs[ii][2];
                dcbar=dcbar+pairs[ii][4]-resul;
                outpt<<"\n"<<"cexp  ctrue  "<<resul<<" "<<pairs[ii][4];
                }
            }
            mbar=mbar/pairs.size();
            dcbar=dcbar/pairs.size();
            outpt<<"\n"<<"mbar  dcbar "<<mbar<<" "<<dcbar<<"\n";
            for (unsigned int ii=0;ii<pairs.size();ii++)
            {
                double mym=pairs[ii][2];
                double mydc=pairs[ii][4]-cexp[ii];
                if(wghts[ii]>0.7)
                {
                c2=c2+((mym-mbar)*(mydc-dcbar));
                c1=c1+pow((mym-mbar),2)-pow((mydc-dcbar),2);
                outpt<<"\n"<<"mym mydc "<<mym<<" "<<mydc;
                }
            }
            //m1 is dx we need c is dy we need
            c0=-1*c2;
            outpt<<"\n"<<" c0 c1 c2 "<<c0<<" "<<c1<<" "<<c2<<"\n";
            m1=((-c1/c2)+sqrt(pow(c1/c2,2)+4))*0.5;
            m2=((-c1/c2)-sqrt(pow(c1/c2,2)+4))*0.5;

            c1=dcbar-(m1*mbar);
            c2=dcbar-(m2*mbar);
            for (unsigned int ii=0;ii<pairs.size();ii++)
            {
                double mym=pairs[ii][2];
                double mydc=pairs[ii][4]-cexp[ii];
                e1=e1+((pow(mydc-c1-(m1*mym),2))/(1+(m1*m1)));
                e2=e2+((pow(mydc-c2-(m2*mym),2))/(1+(m2*m2)));
            }
            diff[0]=m2;
            diff[1]=c2;
            outpt<<"ms and ces and ees"<<m1<<" "<<m2<<" "<<c1<<"
"<<c2<<" "<<e1<<" "<<e2<<"\n";
            if (e1<e2)
            {
                diff[0]=m1;
                diff[1]=c1;
```

```
            }
            diff[3]=3600/2;
            diff[4]=3600/2;

            double wghtx=exp(-pow(-diff[0]-dx,2)/(3600));
            double wghty=exp(-pow(-diff[1]-dy,2)/(3600));

            if(pairs.size()<2)
            {
                diff[0]=dx;diff[1]=dy;
            }
            if(wghtx<0.5)
            {
                diff[0]=dx;
            }
            if(wghty<0.5)
            {
                diff[0]=dy;
            }
            e1=0;e2=0;*/
        }
        if(pairs.size()<2||detr<0.9)
            {
                diff[2]=dth;diff[5]=100;rot=-dth;
            }
//m1 is dx we need c is dy we need.  i used dx which is dy for robot
thus dy is actually robot dx cheers
            vector<double> xe(181,0);//x expected
            vector<double> ye(181,0);//y expected
            double sumWgx=0;
            double sumWgy=0;
            double expdx=0;
            double sumx=0;
            double expdy=0;
            double sumy=0;
            for(int in=0;in<181;in++)
            {
                xe[in]=(x2[in]*cosd(rot))-(y2[in]*sind(rot));//%find
expected x from rotation of coordinates
                ye[in]=(x2[in]*sind(rot))+(y2[in]*cosd(rot));//%find
expected y from rotation of coordinates
                expdx=x1[in]-xe[in];
                wgt=exp(-pow(-expdx-dx,2)/(2*400));

                sumWgx=sumWgx+(wgt*wgt);
                sumx=sumx+(wgt*expdx*wgt);
                expdy=y1[in]-ye[in];
                wgt=exp(-pow(-expdy-dy,2)/(2*400));
```

```
                sumWgy=sumWgy+(wgt*wgt);
                sumy=sumy+(wgt*expdy*wgt);
            }
            expdx=-sumx/sumWgx;//%find  expected dx %/(size(xrnd,2));
            expdy=-sumy/sumWgy;//%find  expected dy %/(size(yrnd,2));
            diff[0]=expdx;
            diff[1]=expdy;
            diff[3]=400;//certn;
            diff[4]=900;//certny;
            outpt<<"\n"<<"dx dy "<<expdx<<" "<<expdy<<"\n";
        shod_G=shod;
        for (int of=0;of<181;of++)
        {
            prev_scan[of][0]=laser_range[of][0];
            prev_scan[of][1]=laser_range[of][1];
        }
}
  if(acquired==0)
  {
    acquired=1;
  }
  return diff;
}//scan matching based on lines
```

Appendix E

UKF related code

Appendix E.1: UKF in main loop

```
while(1)
{
    robot.setVel2(100,200);
    pos=robot.getPose();
    double modTh=pos.getTh();//This is the modified th to take into
account +/- 180
    double shrt=0;//key to shorten extended range
    if(pos.getTh()<0){modTh=pos.getTh()+360;}//keep within 0-360

    if(abs(modTh-breaker[2][0])>300&&modTh>180)//differet quadrants top
vs bottom
    {
        shrt=-1;
    }

    if(abs(modTh-breaker[2][0])>300&&breaker[2][0]>180)//differet
quadrants top vs bottom
    {
        shrt=1;
    }



    if(abs(modTh-breaker[2][0]+(shrt*360))>5||abs(pos.getX()-
breaker[0][0])>100||abs(pos.getY()-breaker[1][0])>100)
    {
        double dth=modTh-breaker[2][0]+(shrt*360);
        double dx=pos.getX()-breaker[0][0];
        double dy=pos.getY()-breaker[1][0];
        breaker[2][0]=modTh;
        breaker[0][0]=pos.getX();
        breaker[1][0]=pos.getY();
        vector<double> lns;
        lns=scn_mtch(dx,dy,dth,laser);

globl_scn[0]=globl_scn[0]+lns[0];globl_scn[1]=globl_scn[1]+lns[1];globl
_scn[2]=globl_scn[2]+lns[2];

    ofstream ako;ako.open("logs2.txt",ios_base::app);
//  ako<<"\n"<<" new run has : "<<"dth  "<<dth<<" tho "<<globl_ps[2]<<"
ztho  "<<globl_scn[2]<<" odo "<<breaker[2][0]<<" zdth "<<lns[2]<<"
sigth "<<sqrt(globl_ps[5])<<" sigzth "<<sqrt(lns[5])<<"\n";
```

```
    ako<<"\n"<<modTh<<" "<<dth<<"  "<<globl_ps[2]<<"
"<<breaker[2][0]<<" "<<lns[2]<<" "<<dx<<"  "<<globl_ps[0]<<"
"<<breaker[0][0]<<" "<<lns[0]<<" "<<dy<<"  "<<globl_ps[1]<<"
"<<breaker[1][0]<<" "<<lns[1]<<" ;";

globl_ps=UKF1(dx,dy,dth,globl_ps[0],globl_ps[1],globl_ps[2],lns[0],lns[
1],lns[2],sqrt(globl_ps[3]),sqrt(globl_ps[4]),(globl_ps[5]));



    msb<<"\n"<<" dx dy dz                  "<<lns[0]<<"  "<<lns[1]<<"
"<<lns[2];
    msb<<"\n"<<"pose est  dx dy dz      "<<globl_ps[0]<<"
"<<globl_ps[1]<<"   "<<globl_ps[2]<<";";
    msb<<"\n"<<"observe   dx dy dz      "<<globl_scn[0]<<"
"<<globl_scn[1]<<"   "<<globl_scn[2]<<";";
    msb<<"\n"<<"odometer dx  dy dz     "<<pos.getX()<<"
"<<pos.getY()<<"   "<<pos.getTh()<<";"<<" ffft "<<5/lns[5]<<"
"<<lns[5];
    cout<<"\n"<<"odometer dx     "<<pos.getX()<<" dy "<<pos.getY()<<"
dth "<<pos.getTh()<<";";
    cout<<"\n"<<"ukf dx "<<globl_ps[0]<<" ukf dy "<<globl_ps[1]<<" ukf
dth "<<globl_ps[2]<<"\n";

    }
}
```

Appendix E.2: UKF function

```
vector<double> UKF1(double dx,double dy,double dth,double xo, double
yo,double tho,double zxo,double zyo,double ztho, double sigx, double
sigy, double sigth)
{
    //inputs dx/dy/dth xo/yo/tho from previous run zxo,zyo,ztho are
deltas from scan matching sigx sigy sigth are from previous run
     //assume dx is for dv.dt


//update=UKF1(dx,dy,dth,update[0],update[1],update[2],zx,zy,zth,update[
3],update[4],update[5]);

     //returnable
     vector<double> update(6,0);//xo yo tho sigx sigy sigth
     ofstream offf;offf.open("debug2.txt",ios_base::app);
     offf<<"\n"<<"This is new  "<<"\n";
     offf<<" inputs "<<dx<<" "<<dy<<" "<<dth<<" "<<xo<<" "<<yo<<"
"<<tho<<" "<<zxo<<" "<<zyo<<" "<<ztho<<" "<<sigx<<" "<<sigy<<"
"<<sigth<<"\n";
     offf<<"For DX "<<"\n";
```

```
      //for dx
      double scx=0;//square root of covariance of cx
      double Qx=400;//measurement noise covariance
      double Rx=(30)+abs((1*dx*sind(tho)));//process noise covariance
      offf<<"Rx is "<<Rx<<"\n";
      //sigma points
      double sx1;
      double sx2;
      double mx=0;//mean of sigma points

      //weights for sigma points
      double k=2;
      double alpha=0.25;
      double lamda=(pow(alpha,2)*(1+k))-1;
      double wm=lamda/(1+lamda);//weight for mean of first point
      double ws1=(lamda/(1+lamda))+(1-(alpha*alpha)+2);//weight for
variance of first point
      double wc=1/(2*(1+lamda));//weight for rest of points
      offf<<" Weights wm "<<wm<<" ws1 "<<ws1<<" wc "<<wc<<"\n";
      double pcx=0;

      //create sigma points
      sx1=xo+sqrt((1+lamda)*sigx*sigx);
      sx2=xo-sqrt((1+lamda)*sigx*sigx);
      offf<<"Sigma points "<<xo<<" "<<sx1<<" "<<sx2<<"\n";
      //propagate sigma point
      xo=xo+dx;
      sx1=sx1+dx;
      sx2=sx2+dx;
      mx=mx+(wm*xo)+(wc*sx1)+(wc*sx2);
      offf<<"Propagations "<<xo<<" "<<sx1<<" "<<sx2<<"with mean
"<<mx<<"\n";
      //predicted covariance
      pcx=pcx+(ws1*(xo-mx)*(xo-mx));
      pcx=pcx+(wc*(sx1-mx)*(sx1-mx));
      pcx=pcx+(wc*(sx2-mx)*(sx2-mx));
      pcx=pcx+Rx;
      offf<<" Covariance predicted "<<pcx<<"\n";
      //expected measurement is the same
      double Sx=pcx-Rx+Qx;
      double CCx=pcx-Rx;//cross covariance

      double KG=CCx/Sx;//kalman gain
      //state update
      mx=mx+(KG*(xo-dx+zxo-mx));
      double cx=(pcx)-(KG*KG*Sx);
      offf<<"Covs sx "<<Sx<<" ccx "<<CCx<<" KG "<<KG<<"\n";
      update[0]=mx;
```

```
     update[3]=cx;
     offf<<"final  "<<mx<<" "<<cx<<"\n";

////////////////////////////////////////////////////////////////
     //for dy
     double scy=0;//square root of covariance of cx
     double Qy=400;//measurement noise covariance
     double Ry=(30)+abs((1*dy*cosd(tho)));//process noise covariance
     offf<<"\n"<<"For Dy "<<"\n";
     //sigma points
     //use same sigma points
     mx=0;//mean of sigma points
     offf<<"Ry is "<<Ry<<"\n";
     //weights for sigma points
     //use same weights
     pcx=0;

     //create sigma points
     sx1=yo+sqrt((1+lamda)*sigy*sigy);
     sx2=yo-sqrt((1+lamda)*sigy*sigy);
     offf<<"Sigma points "<<yo<<" "<<sx1<<" "<<sx2<<"\n";
     //propagate sigma point
     yo=yo+dy;
     sx1=sx1+dy;
     sx2=sx2+dy;
     mx=mx+(wm*yo)+(wc*sx1)+(wc*sx2);
     offf<<"Propagations "<<yo<<" "<<sx1<<" "<<sx2<<"with mean
"<<mx<<"\n";
     //predicted covariance
     pcx=pcx+(ws1*(yo-mx)*(yo-mx));
     pcx=pcx+(wc*(sx1-mx)*(sx1-mx));
     pcx=pcx+(wc*(sx2-mx)*(sx2-mx));
     pcx=pcx+Ry;
     offf<<" Covariance predicted "<<pcx<<"\n";
     //expected measurement is the same
     Sx=pcx-Ry+Qy;
     CCx=pcx-Ry;//cross covariance

     KG=CCx/Sx;//kalman gain  y
     //state update
     mx=mx+(KG*(yo-dy+zyo-mx));
     cx=(pcx)-(KG*KG*Sx);
     offf<<"Covs sx "<<Sx<<" ccx "<<CCx<<" KG "<<KG<<"\n";
     update[1]=mx;
     update[4]=cx;
     offf<<"final"<<mx<<" "<<cx<<"\n";

////////////////////////////////////////////////////////////////
     //for dth
```

```
      if(tho<0){tho=tho+360;}
      offf<<" For Dth "<<"\n";
      double scth=0;//square root of covariance of cx
      double Qth=9;//measurement noise covariance
      double Rth=pow(2+(0.05/(2*0.33)),1);//process noise covariance
      offf<<"Rth is "<<Rth<<"\n";
      //sigma points
      //use same sigma points
      mx=0;//mean of sigma points

      //weights for sigma points
      //use same weights
      pcx=0;

      //create sigma points

      sx1=tho+sqrt((1+lamda)*sigth*sigth);
      sx2=tho-sqrt((1+lamda)*sigth*sigth);
      offf<<"Sigma points "<<tho<<" "<<sx1<<" "<<sx2<<"\n";
      //propagate sigma point
      tho=tho+dth;
      sx1=sx1+dth;
      sx2=sx2+dth;
      mx=mx+(wm*tho)+(wc*sx1)+(wc*sx2);
      offf<<"Propagations "<<tho<<" "<<sx1<<" "<<sx2<<"with mean
"<<mx<<"\n";
      //predicted covariance
      pcx=pcx+(ws1*(tho-mx)*(tho-mx));
      pcx=pcx+(wc*(sx1-mx)*(sx1-mx));
      pcx=pcx+(wc*(sx2-mx)*(sx2-mx));
      pcx=pcx+Rth;
      offf<<" Covariance predicted "<<pcx<<"\n";
      //expected measurement is the same
      Sx=pcx-Rth+Qth;
      CCx=pcx-Rth;//cross covariance

      KG=CCx/Sx;//kalman gain  y
      //state update
      mx=mx+(KG*(tho-dth+ztho-mx));
      cx=(pcx)-(KG*KG*Sx);
      if(mx<0){mx=mx+360;}
      update[2]=mx;
      update[5]=cx;
      offf<<"Covs sx "<<Sx<<" ccx "<<CCx<<" KG "<<KG<<"\n";
      offf<<"final"<<mx<<" "<<cx<<"\n";
      offf<<"summary "<<update[0]<<" "<<update[1]<<" "<<update[2]<<"
"<<update[3]<<" "<<update[4]<<" "<<update[5]<<"\n";
      return update;
}
```

Appendix F

Get laser function

Function to get laser sensor data for grid map, obstacle avoidance, and dynamic objects detection.

```
void getLaser1(ArLaser *Laser)
{
    //optional to print out to user
    ofstream las1;
    las1.open("reading las.txt", ios_base::app);

    const std::list<ArSensorReading *> *readingsList;//pointer to array
with laser readings
    std::list<ArSensorReading *>::const_iterator it;//iterator to loop
through readings
    int i = -1; //Loop counter for readings
    readingsList = Laser->getRawReadings();//get laser readings and
save in array

    for (it = readingsList->begin(); it != readingsList->end(); it++)
    {
        i++;
        laser_range[i][0]=(*it)->getRange();//save laser range
        laser_range[i][1]=(*it)->getSensorTh();//save laser angle


        //Output distance and angle
    las1 << "Laser reading " << i << " = " << (*it)->getRange()<< "
Angle " << i << " = " << (*it)->getSensorTh() << "\n";
    las1 << (*it)->getRange()<< "  ";
    }

    laser_pose[0][0]=(*readingsList->begin())->getXTaken();
    laser_pose[1][0]=(*readingsList->begin())->getYTaken();
    laser_pose[2][0]=(*readingsList->begin())->getThTaken();
    las1<<laser_pose[0][0]<<"  "<<laser_pose[1][0]<<"
"<<laser_pose[2][0]<<" ;";las1<<"\n";
//las1<<laser_pose[1][1]<<";         "<<laser_pose[2][1]<<";
"<<laser_pose[3][1]<<";\n";
    las1.close();


}//get laser readings
```

Appendix G

Grid map related code

Appendix G.1: Acquiring sonar data for grid map

```
vector<vector<double>> sonarRanges(ArRobot *robot)//function for grid
map
{
    vector<vector<double>> readings;//ofstream outp;//outp.open("sonar
grid.txt",ios_base::app);

            for (int i=0;i<8;i++)
            {
                vector<double> rows;
                double rng= robot->getSonarReading(i)-
>getRange();//fill first row of table by reading of first sonar.
                double gamma=(robot->getPose().getTh()*PI/180)+(robot-
>getSonarReading(i)->getSensorTh()*PI/180);//get current sonar
orientation=robot orientation+sonar orientation on robot

                double mag=sqrt((robot->getSonarReading(i)-
>getSensorX()*robot->getSonarReading(i)->getSensorX())+(robot-
>getSonarReading(i)->getSensorY()*robot->getSonarReading(i)-
>getSensorY()));//distance of sonar from robot centre
                double x=robot->getPose().getX()+(mag*cos(gamma));//to
be used for building map origin of sonar scan
                double y=robot->getPose().getY()+(mag*sin(gamma));

rows.push_back(x);rows.push_back(y);rows.push_back(gamma);rows.push_bac
k(rng);
                readings.push_back(rows);
            }

            //plotting
            //for (int i=0;i<readings[0].size();i++)
            //{
            //  outp<<readings[0][i]<<"  "<<readings[1][i]<<"
"<<readings[2][i]<<"  "<<readings[3][i]<<"  "<<readings[4][i]<<"
"<<readings[5][i]<<"  "<<readings[6][i]<<"  "<<readings[7][i]<<" ;
"<<"\n";

        //  }
          //  outp.close();

return reading
```

Appendix G.2: Basic Grid Map Laser only

```
void mapp3(ArRobot *robot,ArLaser *laser)
 {
    getLaser1(laser);
    double id=0;
    for (int i=0;i<181;i++)
    {
        double lxc;
        double lyc;
        double olxc;
        double olyc;
        double
lx=laser_pose[0][0]+(laser_range[i][0]+(100*sind(i)))*cosd(laser_range[
i][1]+laser_pose[2][0]);//point x
        double
ly=laser_pose[1][0]+(laser_range[i][0]+(100*sind(i)))*sind(laser_range[
i][1]+laser_pose[2][0]);//point y
        int k=1;
        if(laser_range[i][0]>5000)
        {

lx=laser_pose[0][0]+(5000+(100*sind(i)))*cosd(laser_range[i][1]+laser_p
ose[2][0]);

ly=laser_pose[1][0]+(5000+(100*sind(i)))*sind(laser_range[i][1]+laser_p
ose[2][0]);
            k=0;
        }
        lxc=100+ceil((lx-50)/100);// laser xc
        lyc=100+ceil((ly-50)/100);//laser yc
        if(!(k==0))
        {
            mapg2[lyc][lxc]=1;
        }

        olxc=100+ceil((laser_pose[0][0]-50)/100);//origin x of laser
grid cell
        olyc=100+ceil((laser_pose[1][0]-50)/100);//origin y of laser
grid cell
        mapg2[olyc][olxc]=10;
        double m=(lyc-olyc)/(lxc-olxc);//gradient of line
        double c=lyc-(m*lxc);//intercept
        for (int
j=min(olxc,lxc)+1;j<max(lxc,olxc);j++)//j=olxc+1;j<lxc;j++ modified
        {
            int cell=ceil((m*j)+c);
            double prob=0.3;
```

```
mapg2[cell][j]=(prob*mapg2[cell][j])/((prob*mapg2[cell][j])+((1-
prob)*(1-mapg2[cell][j])));//bayesian update
        }
}


 }
```

Appendix G.3: Basic grid map sonar only

```
void mapp1(ArRobot *robot)
 {
//mod
 for (int i=0;i<8;i++)//for each sonar
 {
     double sensor_diag=sqrt(pow(robot->getSonarReading(i)-
>getSensorY(),2)+pow(robot->getSonarReading(i)->getSensorX(),2));//this
gets the sensor diagonal (distance from robot centre)
     //find robot position
     double sonc[1][2];//sonar centre position x,y
     double robc[1][2];//robot centre position x,y
     sonc[0][1]= robot->getSonarReading(i)-
>getYTaken()+(sensor_diag*sin(gamma[i][sonar_column_counter]));//store
current y position of sonar (y of robot + y of sonar on device)
     sonc[0][0]= robot->getSonarReading(i)-
>getXTaken()+(sensor_diag*cos(gamma[i][sonar_column_counter]));//store
current x position of sonar (x of robot + x of sonar on device)
     robc[0][1]=0.5*( robot->getSonarReading(0)->getYTaken()+robot-
>getSonarReading(7)->getYTaken());//store current y position of robot
     robc[0][0]= robot->getSonarReading(0)->getXTaken();//store current
x position of robot

     double robl[2][2];//limits array for robot position
     robl[0][0]=robc[0][0]-177;//min x
     robl[0][1]=robc[0][0]+177;//max x
     robl[1][0]=robc[0][1]-177;//min y
     robl[1][1]=robc[0][1]+177;//max y
//mod
     double robcr[2][2];//to get coordinates in of cells in grid map
     robcr[0][0]=100+ceil((robl[0][0]-50)/100);//min x cell
     robcr[0][1]=100+ceil((robl[0][1]-50)/100);//max x cell
     robcr[1][0]=100+ceil((robl[1][0]-50)/100);//min y cell
     robcr[1][1]=100+ceil((robl[1][1]-50)/100);//max y cell

     for (int l=robcr[1][0];l<robcr[1][1]+1;l++)//fill cells with
current robot poition i.e. occupied by robot
     {
```

```
          for (int m=robcr[0][0];m<robcr[0][1]+1;m++)
          {
              mapg[l][m]=10;//cells occupied by robot
          }
      }

      double ang=(robot->getSonarReading(i)-
>getThTaken()*PI/180)+(robot->getSonarReading(i)-
>getSensorTh()*PI/180);//get current sonar orientation=robot
orientation+sonar orientation on robot
      double rng=robot->getSonarReading(i)-
>getRange()+sensor_diag;//range of sonar

      for (int l=0;l<201;l++) //fill map
      {
          for (int m=0;m<201;m++)
          {
              double ex=(100*(m-100))-robc[0][0];//element x
              double ey=(100*(l-100))-robc[0][1];//element y
              double er=sqrt((ey*ey)+(ex*ex));//element distance
              double ea=atan2(ey,ex);//element angle

              if(abs(ea-ang)<11.5*3.142/180&&er<5000+sensor_diag) //if
elelement within sonar cone
              {
                  if(rng-150<er&&rng+150>er&&!(rng==5000))//region 1
                  {
                      if(!(mapg[l][m]==10))
                      {
                      double prob=(((5000-er)/5000)+((3.89-(ea-
ang))/3.89))*0.5*0.98;

mapg[l][m]=(prob*mapg[l][m])/((prob*mapg[l][m])+((1-prob)*(1-
mapg[l][m])));//bayesian update
                      }
                  }

                  if(rng-150>er)//region 2
                  {
                       if(!(mapg[l][m]==10))
                      {
                      double prob=1-((((5000-er)/5000)+((3.89-(ea-
ang))/3.89))*0.5);

mapg[l][m]=(prob*mapg[l][m])/((prob*mapg[l][m])+((1-prob)*(1-
mapg[l][m])));//bayesian update
                      }
                  }
              }
```

```
        }
    }
 }
}
```

Appendix G.4: Basic Grid map laser and sonar

```
void mapp2(ArRobot *robot,ArLaser *laser)
 {//ofstream gh;gh.open("details map.txt",ios_base::app);gh<<" function
call new  "<<"\n"<<"\n";
     //laser
     vector< double> mod;
     int tilesensor;//this is the tile in which the sensor operates
     int tileread;//this is the tile in which the reading is
     getLaser1(laser);
    //double id=0;
    for (int il=0;il<181;il++)
    {
        double lxc;
        double lyc;
        double olxc;
        double olyc;
        double
lx=laser_pose[0][0]+(laser_range[il][0]+(100*sind(il)))*cosd(laser_rang
e[il][1]+laser_pose[2][0]);//point x
        double
ly=laser_pose[1][0]+(laser_range[il][0]+(100*sind(il)))*sind(laser_rang
e[il][1]+laser_pose[2][0]);//point y
        int k=1;
        if(laser_range[il][0]>3000)
        {

lx=laser_pose[0][0]+(3000+(100*sind(il)))*cosd(laser_range[il][1]+laser
_pose[2][0]);

ly=laser_pose[1][0]+(3000+(100*sind(il)))*sind(laser_range[il][1]+laser
_pose[2][0]);
            k=0;
        }
        lxc=100+ceil((lx-50)/100);// laser xc
        lyc=100+ceil((ly-50)/100);//laser yc
        mod=allocateTile(lxc,lyc);
        tileread=tile;
        lxc=mod[0];
        lyc=mod[1];
```

```
        if(!(k==0))
        {
            fullmap[tile][lyc][lxc]=1;//mapg2[lyc][lxc]=1;
        }

        olxc=100+ceil((laser_pose[0][0]-50)/100);//origin x of laser
grid cell
        olyc=100+ceil((laser_pose[1][0]-50)/100);//origin y of laser
grid cell
        mod= allocateTile(olxc,olyc);
        olxc=mod[0];
        olyc=mod[1];
        tilesensor=tile;
        fullmap[tile][olyc][olxc]=100;//mapg2[olyc][olxc]=100;

        //prepare for loop grid
        lyc=((200*mapstack[tileread][1])+lyc);
        olyc=((200*mapstack[tilesensor][1])+olyc);
        lxc=((200*mapstack[tileread][0])+lxc);
        olxc=((200*mapstack[tilesensor][0])+olxc);
        double m=(lyc-olyc)/(lxc-olxc);//double
m=(((200*mapstack[tileread][1])+lyc)-
((200*mapstack[tilesensor][1])+olyc))/(((200*mapstack[tileread][0])+lxc
)-((200*mapstack[tilesensor][0])+olxc));//gradient of line
        double c=lyc-(m*lxc);//intercept
        if(lxc-olxc==0){

            for (int j=olyc;j<lyc;j++)
        {
            int cell=lxc;

            mod= allocateTile(cell,j);
            //use instead of j mod[0][0];
            cell=mod[1];

if(!(fullmap[tile][cell][mod[0]]==100))//!(mapg2[cell][j]==100))
            {
                double prob=0.3;

fullmap[tile][mod[1]][mod[0]]=(prob*fullmap[tile][mod[1]][mod[0]])/((pr
ob*fullmap[tile][mod[1]][mod[0]])+((1-prob)*(1-
fullmap[tile][mod[1]][mod[0]])));//mapg2[cell][j]=(prob*mapg2[cell][j])
/((prob*mapg2[cell][j])+((1-prob)*(1-mapg2[cell][j])));//bayesian
update
            }
            }}
        else{
```

```
        for (int j=min(olxc,lxc);j<max(lxc,olxc);j++)
        {
            int cell=ceil((m*j)+c);

            mod= allocateTile(j,cell);
            //use instead of j mod[0][0];
            cell=mod[1];

if(!(fullmap[tile][cell][mod[0]]==100))//!(mapg2[cell][j]==100))
            {
                double prob=0.3;

fullmap[tile][cell][mod[0]]=(prob*fullmap[tile][cell][mod[0]])/((prob*f
ullmap[tile][cell][mod[0]])+((1-prob)*(1-
fullmap[tile][cell][mod[0]])));//mapg2[cell][j]=(prob*mapg2[cell][j])/(
(prob*mapg2[cell][j])+((1-prob)*(1-mapg2[cell][j])));//bayesian update
            }
        }}
}



     //laser done
      for (int i=0;i<8;i++)//for each sonar
      {//gh<<"new sonar "<<i<<"\n";
          if(robot->getSonarReading(i)->getRange()<3000){
     double sensor_diag=sqrt(pow(robot->getSonarReading(i)-
>getSensorY(),2)+pow(robot->getSonarReading(i)->getSensorX(),2));//this
gets the sensor diagonal (distance from robot centre)
     //find robot position
     double sonc[1][2];//sonar centre position x,y
     double robc[1][2];//robot centre position x,y
     double robcc[1][2];//robot centre position grids
     double ang=(robot->getSonarReading(i)-
>getThTaken()*PI/180)+(robot->getSonarReading(i)-
>getSensorTh()*PI/180);//get current sonar orientation=robot
orientation+sonar orientation on robot
     sonc[0][1]= robot->getSonarReading(i)-
>getYTaken()+(sensor_diag*sin(ang));//store current y position of sonar
(y of robot + y of sonar on device)
     sonc[0][0]= robot->getSonarReading(i)-
>getXTaken()+(sensor_diag*cos(ang));//store current x position of sonar
(x of robot + x of sonar on device)
     robc[0][1]=0.5*( robot->getSonarReading(0)->getYTaken()+robot-
>getSonarReading(7)->getYTaken());//store current y position of robot
     robc[0][0]= robot->getSonarReading(0)->getXTaken();//store current
x position of robot
     robcc[0][0]=100+ceil((robc[0][0]-50)/100);
     robcc[0][1]=100+ceil((robc[0][1]-50)/100);
     double soncc[1][2];
```

```
     soncc[0][0]=100+ceil((sonc[0][0]-50)/100);
     soncc[0][1]=100+ceil((sonc[0][1]-50)/100);
//gh<<" sonar x "<<sonc[0][0]<<" sonar y "<<sonc[0][1]<<" robot x
"<<robc[0][0]<<" robot y "<<robc[0][1]<<"\n";
     double robl[2][2];//limits array for robot position
     robl[0][0]=robc[0][0]-177;//min x
     robl[0][1]=robc[0][0]+177;//max x
     robl[1][0]=robc[0][1]-177;//min y
     robl[1][1]=robc[0][1]+177;//max y

     double robcr[2][2];//to get coordinates in of cells in grid map
     robcr[0][0]=100+ceil((robl[0][0]-50)/100);//min x cell
     robcr[0][1]=100+ceil((robl[0][1]-50)/100);//max x cell
     robcr[1][0]=100+ceil((robl[1][0]-50)/100);//min y cell
     robcr[1][1]=100+ceil((robl[1][1]-50)/100);//max y cell
     mod= allocateTile(robcr[0][0],robcr[1][0]);
        robcr[0][0]=mod[0];
        robcr[1][0]=mod[1];
        tilesensor=tile;

        mod= allocateTile(robcr[0][1],robcr[1][1]);
        robcr[0][1]=mod[0];
        robcr[1][1]=mod[1];
        tileread=tile;
    // gh<<"robot cells x " <<robcc[0][0]<<" y "<<robcc[0][1]<<"\n";
    // gh<<"robcr[0][0] "<<robcr[0][0]<<" robcr[0][1] "<<robcr[0][1]<<"
robcr[1][0] "<<robcr[1][0]<<" robcr[1][1] "<<robcr[1][1]<<"\n";
        robcr[0][1]=((200*mapstack[tileread][0])+robcr[0][1]);//max x
        robcr[0][0]=((200*mapstack[tilesensor][0])+robcr[0][0]);//min x
        robcr[1][1]=((200*mapstack[tileread][1])+robcr[1][1]);//max y
        robcr[1][0]=((200*mapstack[tilesensor][1])+robcr[1][0]);//min y

     for (int l=robcr[1][0];l<robcr[1][1]+1;l++)//fill cells with
current robot poition i.e. occupied by robot
     {
         for (int m=robcr[0][0];m<robcr[0][1]+1;m++)
         {
             mod= allocateTile(m,l);
             fullmap[tile][mod[1]][mod[0]]=100;
             //mapg2[l][m]=100;//cells occupied by robot
         }
     }

     double rng=robot->getSonarReading(i)-
>getRange()+sensor_diag;//range of sonar
     double mang=tan(ang);//find gradient
     double nang=-1/(mang+0.001);// gradient of normal
     nang=atan(nang);//ange of normal
     double cnt[1][2];
```

```
     cnt[0][0]= (cos(ang)*rng)+sonc[0][0];
     cnt[0][1]= (sin(ang)*rng)+sonc[0][1];//centerline point
     double wdth=rng*23*3.142/360;//this is the half width of sonar
beam
     double endpt[2][2];
     endpt[0][0]=cnt[0][0]-(wdth*cos(nang));//x low
     endpt[0][1]=cnt[0][1]- (wdth*sin(nang));//y low
     endpt[1][0]=cnt[0][0]+(wdth*cos(nang));//x high
     endpt[1][1]=cnt[0][1]+(wdth*sin(nang));//y high
     double cntc[1][2];
     cntc[0][0]=100+ceil((cnt[0][0]-50)/100);
     cntc[0][1]=100+ceil((cnt[0][1]-50)/100);//centre cell
     double endptcmx[1][2];
     double endptcmn[1][2];
     endptcmx[0][0]=100+max(ceil((endpt[0][0]-
50)/100),ceil((endpt[1][0]-50)/100));//max x
     endptcmx[0][1]=100+max(ceil((endpt[0][1]-
50)/100),ceil((endpt[1][1]-50)/100));//cell max y
     endptcmn[0][0]=100+min(ceil((endpt[1][0]-
50)/100),ceil((endpt[0][0]-50)/100));//min x
     endptcmn[0][1]=100+min(ceil((endpt[1][1]-
50)/100),ceil((endpt[0][1]-50)/100));//cell min y

//gh<<" angle "<<ang<<" range "<<rng<<"\n";
//gh<<" Tx  "<<cnt[0][0]<<" Ty "<<cnt[0][1]<<" cells x "<<cntc[0][0]<<"
cells y "<<cntc[0][1]<<"\n";

//gh<<" wdth "<<wdth<<" endpt xl "<<endpt[0][0]<<" endpt xh
"<<endpt[1][0]<<" endp yl "<<endpt[0][1]<<" endpt yh
"<<endpt[1][1]<<"\n";
//gh<<" cell endpt xl "<<endptcmn[0][0]<<" cell endpt yl
"<<endptcmn[0][1]<<" cell endpt xh "<<endptcmx[0][0]<<" cell endpt yh
"<<endptcmx[0][1]<<"\n";
//gh<<"cntc[0][0] "<<cntc[0][0]<<" cntc[0][1] "<<cntc[0][1]<<"
robcc[0][0] "<<robcc[0][0]<<" robcc[0][1] "<<robcc[0][1]<<"\n";
//gh<<" soncc x "<<soncc[0][0]<<" soncc y "<<soncc[0][1]<<"\n";

        mod= allocateTile(soncc[0][0],soncc[0][1]);
        soncc[0][0]=mod[0];
        soncc[0][1]=mod[1];
        tilesensor=tile;

        mod= allocateTile(cntc[0][0],cntc[0][1]);
        cntc[0][0]=mod[0];
        cntc[0][1]=mod[1];
        tileread=tile;
```

```
        cntc[0][0]=((200*mapstack[tileread][0])+cntc[0][0]);//max x
        soncc[0][0]=((200*mapstack[tilesensor][0])+soncc[0][0]);//min x
        cntc[0][1]=((200*mapstack[tileread][1])+cntc[0][1]);//max y
        soncc[0][1]=((200*mapstack[tilesensor][1])+soncc[0][1]);//min y


    double m=( cntc[0][1]- soncc[0][1])/( cntc[0][0]- soncc[0][0]);
        double c= cntc[0][1]-(m* cntc[0][0]);//intercept


//cout<<" m "<<m<<"\n";
if(cntc[0][0]- soncc[0][0]==0)

{for (int m=soncc[0][1];m<cntc[0][1]+1;m++)
     {

             mod= allocateTile(cntc[0][0],m);
             double ex=(100*(cntc[0][0]-100));//element x
             double ey=(100*(m-100));//element y
             double er=sqrt((ey*ey)+(ex*ex));//element distance
             double ea=atan2(ey,ex);//element angle
             if(!(fullmap[tile][mod[1]][mod[0]]==100)&&er<rng-
100&&abs(ea-ang)<11.5*3.142/180)
             {

                 double prob=1-((((5000-er)/5000)+((3.89-(ea-
ang))/3.89))*0.5);

fullmap[tile][mod[1]][mod[0]]=(prob*fullmap[tile][mod[1]][mod[0]])/((pr
ob*fullmap[tile][mod[1]][mod[0]])+((1-prob)*(1-
fullmap[tile][mod[1]][mod[0]])));//bayesian update
             }


}}//if

else{

         for (int
jj=min(cntc[0][0],soncc[0][0]);jj<max(cntc[0][0],soncc[0][0])+1;jj++)
         {
                  int cell=ceil((m*jj)+c);
                  mod= allocateTile(jj,cell);

             double ex=(100*(jj-100))-sonc[0][0];//element x
             double ey=(100*(cell-100))-sonc[0][1];//element y
             double er=sqrt((ey*ey)+(ex*ex));//element distance
```

```
            double ea=atan2(ey,ex);//element angle


            if(!(fullmap[tile][mod[1]][mod[0]]==100)&&er<rng-
100&&abs(ea-ang)<11.5*3.142/180)
            {

                double prob=1-((((5000-er)/5000)+((3.89-(ea-
ang))/3.89))*0.5);

fullmap[tile][mod[1]][mod[0]]=(prob*fullmap[tile][mod[1]][mod[0]])/((pr
ob*fullmap[tile][mod[1]][mod[0]])+((1-prob)*(1-
fullmap[tile][mod[1]][mod[0]])));//bayesian update
            }
         }}

     //for cells in range
    // gh<<"\n"<<"the debug  "<<endptcmn[0][1]<<"
"<<endptcmx[0][1]+1<<"  "<<endptcmn[0][0]<<"  "<<endptcmx[0][0]<<"\n";

        mod= allocateTile(endptcmn[0][0],endptcmn[0][1]);
        endptcmn[0][0]=mod[0];
        endptcmn[0][1]=mod[1];
        tilesensor=tile;

        mod= allocateTile(endptcmx[0][0],endptcmx[0][1]);
        endptcmx[0][0]=mod[0];
        endptcmx[0][1]=mod[1];
        tileread=tile;


endptcmx[0][0]=((200*mapstack[tileread][0])+endptcmx[0][0]);//max x

endptcmn[0][0]=((200*mapstack[tilesensor][0])+endptcmn[0][0]);//min x

endptcmx[0][1]=((200*mapstack[tileread][1])+endptcmx[0][1]);//max y

endptcmn[0][1]=((200*mapstack[tilesensor][1])+endptcmn[0][1]);//min y
double mm=( endptcmx[0][1]- endptcmn[0][1])/( endptcmx[0][0]-
endptcmn[0][0]);
           c= endptcmx[0][1]-(mm* endptcmx[0][0]);//intercept


if(endptcmx[0][0]- endptcmn[0][0]==0)

{for (int m=endptcmn[0][1];m<endptcmx[0][1]+1;m++)
     {
```

```
            mod= allocateTile(endptcmx[0][0],m);
            double ex=(100*(endptcmx[0][0]-100));//element x
            double ey=(100*(m-100));//element y

            double ea=atan2(ey-sonc[0][1],ex-sonc[0][0]);//element
angle

            if(rng<5000&&abs(ea-ang)<11.5*3.142/180)
            {



if(!(fullmap[tile][mod[1]][mod[0]]==100))//(!(mapg2[l][m]==100))
                {
                    double prob=0.98*0.8;//((3.89-(ea-
ang))/3.89)*1*0.98;

fullmap[tile][mod[1]][mod[0]]=(prob*fullmap[tile][mod[1]][mod[0]])/((pr
ob*fullmap[tile][mod[1]][mod[0]])+((1-prob)*(1-
fullmap[tile][mod[1]][mod[0]])));



                }
            }
}}//if

else{

     for (int m=endptcmn[0][0];m<endptcmx[0][0]+1;m++)
     {
            int cell=ceil((mm*m)+c);
            mod= allocateTile(m,cell);
            double ex=(100*(m-100));//element x
            double ey=(100*(cell-100));//element y

            double ea=atan2(ey-sonc[0][1],ex-sonc[0][0]);//element
angle

            if(rng<5000&&abs(ea-ang)<11.5*3.142/180)
            {



if(!(fullmap[tile][cell][mod[0]]==100))//(!(mapg2[l][m]==100))
                {
```

```
                    double prob=0.98*0.8;//((3.89-(ea-
ang))/3.89)*1*0.98;

fullmap[tile][cell][mod[0]]=(prob*fullmap[tile][cell][mod[0]])/((prob*f
ullmap[tile][cell][mod[0]])+((1-prob)*(1-
fullmap[tile][cell][mod[0]])));



                }
            }
     }}

      }}
 }
```

N.B. For including the UKF update in the map, the UKF updated pose should be used instead of the pose declared by the robot.

Appendix G.5: Allocate tile code

```
vector<double> allocateTile(double a,double b)//ord 0=x 1=y
{// ofstream gh;gh.open("details map.txt",ios_base::app);gh<<" function
call new  "<<"\n"<<"\n";

vector<double> newloc;
vector<double> mapindex1;
int outx=0;
int outy=0;
int found=0;//gh<<"a txl txh b tyl tyh "<<a<<" "<<txl<<" "<<txh<<"
"<<b<<" "<<tyl<<" "<<tyh<<"\n";
if(a<txl||a>txh)//outside current tile xwise
{
    outx=1;
}

if(b<tyl||b>tyh)//outside current tile xwise
{
    outy=1;
}//gh<<"outx out y "<<outx<<" "<<outy<<"\n";

if(outx==1&&outy==0)// not in tile xwise
{ //gh<<"not xwise"<<"\n";
double desx=ceil((a-200)/200);//desired index for tile
double desy=ceil((b-200)/200);//desired index for tile
 found=0;//not found
//search if index exists
```

```
for (int i=0;i<mapstack.size();i++)
{
if(mapstack[i][0]==desx&&mapstack[i][1]==desy)
{
found=1;//found
outx=0;
outy=0;
tile=i;
}
}
}

if(outx==0&&outy==1)// not in tile ywise
{//gh<<"not ywise"<<"\n";
int desiredx=ceil((a-200)/200);//desired index for tile
int desiredy=ceil((b-200)/200);//desired index for tile
 found=0;//not found
//search if index exists
for (int i=0;i<mapstack.size();i++)
{
if(mapstack[i][0]==desiredx&&mapstack[i][1]==desiredy)
{
found=1;//found
outx=0;
outy=0;
tile=i;
}
}
}

if(outx==1&&outy==1)// not in tile x/y wise
{//gh<<"not x/ywise"<<"\n";
int desiredx=ceil((a-200)/200);//desired index for tile
int desiredy=ceil((b-200)/200);//desired index for tile
 found=0;//not found
//search if index exists
for (int i=0;i<mapstack.size();i++)
{
if(mapstack[i][0]==desiredx&&mapstack[i][1]==desiredy)
{
found=1;//found
outx=0;
outy=0;
tile=i;
}
}
}
```

```
if(outx==0&&outy==0&&found==1)//if in found tile
{//gh<<"in tile"<<"\n";
    txl=mapstack[tile][0]*200;
    txh=txl+200;
    tyl=mapstack[tile][1]*200;
    tyh=tyl+200;
    newloc.push_back(a-txl);
    newloc.push_back(b-tyl);
    return newloc;
}


if((outx==1||outy==1)&&found==0)//if no tile found tile
{//gh<<"no tile found"<<"\n";
int desiredx=ceil((a-200)/200);//desired index for tile
int desiredy=ceil((b-200)/200);//desired index for tile
mapindex1.push_back(desiredx);
mapindex1.push_back(desiredy);
mapstack.push_back(mapindex1);
tile=mapstack.size()-1;
vector<vector< double > > mapn ( 201, vector<double> ( 201, 0.5 ) );
fullmap.push_back(mapn);
txl=mapstack[tile][0]*200;
txh=txl+200;
tyl=mapstack[tile][1]*200;
tyh=tyl+200;
newloc.push_back(a-txl);
newloc.push_back(b-tyl);
return newloc;
}


if(outx==0&&outy==0&&found==0)//if in found tile
{//gh<<"all zero"<<"\n";
    txl=mapstack[tile][0]*200;
    txh=txl+200;
    tyl=mapstack[tile][1]*200;
    tyh=tyl+200;
    newloc.push_back(a-txl);
    newloc.push_back(b-tyl);
    return newloc;
}
 }
```

Appendix H

Dynamic objects related code

Appendix H.1: Dyn Det function code

```
vector<double>  dyn_det(vector<double> fr,double
frp[181][1],vector<double> fr2,double frp2[181][1],double rdr,double
rdth,double tme)//dynamic object function
{
    ofstream boi;boi.open("lshtot.txt",ios_base::app);
    boi<<"\n"<<"\n"<<"\n"<<"New call to function ";
    boi<<"\n"<<"rdr rdth  "<<rdr<<" "<<rdth<<" size of frames
"<<fr.size()<<" "<<fr2.size()<<" objn "<<objn;
  vector<double> DOIi;//return dynamic objects
if(objn==0)
{
//no objects detected yet
boi<<"\n"<<"entered no objects if cond";
for (int i=0;i<fr.size()/2;i++)
{
   obji.push_back(fr[2*i]);
   obji.push_back(fr[(2*i)+1]);
   obji.push_back(0.5*(fr[2*i]+fr[(2*i)+1]));
   double meanfrp=0;  int index1;int index2;
   for (int j=fr[2*i];j<fr[(2*i)+1]+1;j++)//add one to max array
   {
     meanfrp=meanfrp+frp[j][0];
   }
   meanfrp=meanfrp/(1+fr[(2*i)+1]-fr[2*i]);//one was added in the
denominator
index1=fr[2*i];index2=fr[(2*i)+1];
   boi<<"\n"<<" objects in fr "<<fr[2*i]<<" "<<fr[(2*i)+1]<<"
"<<0.5*(fr[2*i]+fr[(2*i)+1])<<" "<<frp[index1][0]<<"
"<<frp[index2][0]<<" "<<meanfrp;
   obji.push_back(meanfrp);
   obji.push_back(1);
   obji.push_back(0);
   obji.push_back(0);
 boi<<"\n"<<" The parameters for object "<<i<<" is "<<fr[2*i]<<"
"<<fr[(2*i)+1]<<" "<<0.5*(fr[2*i]+fr[(2*i)+1])<<" "<<meanfrp;
   obj.push_back(obji);
   obji.clear();
   objn=objn+1;
}
}

boi<<"\n"<<" objn is "<<objn;
```

```
if(objn>0)
{boi<<"\n"<<"entered if due to objects greater than 0 ";
    //comparing frames
  double rn;double thn;
   vector<vector<int>> newobj;// array to hold new objects
   vector<double> newobja;// array to hold new objects
    vector<double> cnda;//comparison array
 vector<vector<double>> cnd;//comparison array
 boi<<"\n"<<"now comparing obj and fr2 sizes "<<obj.size()<<"
"<<fr2.size();
   for (int i=0;i<obj.size();i++)
   {boi<<"\n"<<"for every object in obj";
         int fnd=0;
       //find possible candidates assume 50cm/s speed
       //assume dth=500.57/R about 3 deg at 1000mm

       int j=0;
       int sz=fr2.size()/2;

       while (j<sz)//was 1 now removed one
       {
       if(fr2[(2*(j))]!=180)
       {
             double meanfrp=0;
             for (int ii=fr2[2*j];ii<fr2[(2*j)+1]+1;ii++)
             {
              meanfrp=meanfrp+frp2[ii][0];
             }

              double mnr=meanfrp/(fr2[(2*j)+1]+1-fr2[2*j]);

              double mnth=0.5*(fr2[2*j]+fr2[(2*j)+1]);//mean th
location of object
       int index1=fr2[2*j];int index2=fr2[(2*j)+1];
       //'checking match'
       //here i will correct for object th change or just use x and y
       boi<<"\n"<<"\n"<<"The objects compared are "<<"fr2
"<<fr2[2*j]<<" "<<fr2[(2*j)+1]<<" "<<mnth<<" "<<frp2[index1][0]<<"
"<<frp2[index2][0]<<" "<<mnr;
       boi<<"\n"<<"The objects compared are "<<"obj "<<obj[i][0]<<"
"<<obj[i][1]<<" "<<obj[i][2]<<" "<<obj[i][3]<<" "<<obj[i][4];
              int dne=0;//index to know if this i was used before
              for (int y=0;y<cnd.size();y++)
              {
           if(cnd[y][0]==i)
           {
               dne=1;
           }
              }
```

```
            for (int y=0;y<cnd.size();y++)
            {
          if(cnd[y][1]==j)
          {
              dne=1;
          }
            }

    //the change in theta of the feature due to robot motion
       thn=180*atan2((obj[i][3]*sind(obj[i][2]))-
rdr,obj[i][3]*cosd(obj[i][2]))/3.142;//change in theta robott motion
       rn=sqrt(pow((obj[i][3]*sind(obj[i][2])-
rdr),2)+pow((obj[i][3]*cosd(obj[i][2])),2));//change in range due to
robot motion
       boi<<"\n"<<"mnr mnth thn rn "<<mnr<<" "<<mnth<<" "<<thn<<"
"<<rn;
       if(abs(mnr-obj[i][3])<1.3*rdr&&abs(mnth-thn-
rdth)<10||abs(obj[i][2]-mnth)<10+abs(rdth))//abs(mnth)<10+thn-
(rdth)&&dne==0)//match found
       {
            boi<<"\n"<<"match found between "<<i<<" "<<j;

cnda.push_back(i);cnda.push_back(j);cnda.push_back(mnr);cnda.push_back(
obj[i][3]);cnda.push_back(obj[i][2]);
          cnda.push_back(rn);cnda.push_back(thn);cnda.push_back(rdth);
          cnd.push_back(cnda);

          obj[i][5]=mnr-rn;//-rdr;//(0.5*obj(i,6))+(0.5*(0-
obj(i,4)+mnr+(rdr*sind(obj(i,3)))));%*cosd(90-obj(i,3))));%mnr-
obj(i,4)+(rdr*cosd(90-obj(i,3)));%obj(i,6)+mnr-obj(i,4);
          obj[i][6]=mnth-thn-rdth;//(0.5*obj(i,7))+(0.5*(mnth-
obj(i,3)+rdth));%obj(i,7)+mnth-obj(i,3);
          //update matrix
          obj[i][3]=mnr;obj[i][2]=mnth;
          obj[i][0]=fr2[(2*(j))]; obj[i][1]=fr2[(2*(j))+1];
 boi<<"\n"<<" The object speed  r th "<<obj[i][5]<<" "<<obj[i][6];
          fr2[(2*(j))]=180;
          fr2[(2*(j))+1]=180;
          fnd=1;

          j=189;
       }
       }//////i placed if condition here
       sz=fr2.size()/2;
       j=j+1;

       }boi<<"\n"<<"ended while ";
```

```
        double vthrs=sqrt(pow(obj[i][6],2)+pow(obj[i][5],2));
        boi<<"\n"<<"threshold 50"<<vthrs;
        if(fnd==0||vthrs<60)//no match found increment nill counter and
move object
        { boi<<"\n"<<"no match found for "<<i;
            obj[i][4]=obj[i][4]+1;
            obj[i][3]=obj[i][5]+rn+rdr;
            obj[i][2]=obj[i][6]+thn+rdth;

        }
    }

    //eliminate old objects
    boi<<"\n"<<"eliminating old objects ";
    int i=0;
    while(i< obj.size())
    { boi<<"\n"<<"i index"<<i;
    if(obj[i][4]>3)
    {
         boi<<"\n"<<"erased i = "<<i;
         obj.erase(obj.begin()+i);
        i=i-1;
    }
    i=i+1;
    }
    objn=obj.size();
     boi<<"\n"<<"size of objn "<<objn;
   //noww find new objects
     boi<<"\n"<<"find new objects ";
    for (int ji=0;ji<(fr2.size()/2);ji++)
    {
     if(fr2[2*ji]!=180)
    {
    newobja.push_back(fr2[(2*(ji))]);
    newobja.push_back(fr2[(2*(ji))+1]);
    newobja.push_back(0.5*(fr2[2*ji]+fr2[(2*ji)+1]));//was i
    double meanfrp2=0;  boi<<"\n"<<"new object found "<<fr2[(2*(ji))]<<"
"<<fr2[(2*(ji))+1]<<" "<<0.5*(fr2[2*ji]+fr2[(2*ji)+1]);
    for (int jj=fr2[2*ji];jj<fr2[(2*ji)+1]+1;jj++)//add one to max array
    {
      meanfrp2=meanfrp2+frp2[jj][0];
    }

    meanfrp2=meanfrp2/(1+fr2[(2*ji)+1]-fr2[2*ji]);boi<<" "<<meanfrp2;
    newobja.push_back(meanfrp2);
    newobja.push_back(1);
    newobja.push_back(0);
    newobja.push_back(0);
   // newobj.push_back(newobja);
```

```
   obj.push_back(newobja);newobja.clear();
   objn=objn+1;//was 0.5
   boi<<"\n"<<" due to new objn is "<<objn;
    }
   }
   boi<<"\n"<<" objn is "<<objn;
   //boi<<"\n"<<"sort in order ";
   boi<<"\n"<<"The old and new are "<<" "<<obj.size()<<"
"<<newobj.size();
   //now sort in order
   vector<vector<double>> interobj;//intermediate obj vector
   for (int i=0;i<newobj.size();i++)
   {
      double srt=newobj[i][0];
      int j=0;//was 1 now changed to zero
      int   sz=obj.size();
      while (j<obj.size())
      {
        if(obj[j][0]>srt)
        {

          for(int h=0;h<j;h++)//push back all of objects except j

          {
          interobj.push_back(obj[h]);
          }



          vector<double> fill;//vector used to fill new object
          for(int u=0;u<newobj[0].size();u++)//add new object
          {
              fill.push_back(newobj[i][u]);
          }
          interobj.push_back(fill);



       //   interobj.push_back(newobj[i]);

          for(int h=j;h<sz;h++)//fill rest of obj

          {
          interobj.push_back(obj[h]);
          }
          obj.swap(interobj);
          j=180;
       }///////////////if obj greater than start
        else if(srt>maxvec(obj))
        {
```

```
            vector<double> fill;//vector used to fill new object
            for(int u=0;u<newobj[0].size();u++)
            {
                fill.push_back(newobj[i][u]);
            }
            obj.push_back(fill);
            j=180;
        }
        j=j+1;
      } ////////////////////////////////////////while loop
  }////////////////////////////for loop
}

//boi<<"\n"<<"Final draft ";
 for (int i=0;i<obj.size();i++)
 {boi<<"\n"<<obj[i][0]<<" "<<obj[i][1];
 DOIi.push_back(obj[i][0]); DOIi.push_back(obj[i][1]);
 boi<<"\n"<<" size "<<obj.size()<<" "<<obj[0].size()<<" "<<DOIi.size();
 }
 return DOIi;
}
```

Appendix H.2: Dyn Extract function code

```
void DynExtrct(double dx, double dy, double dth, double tme)
{
int i=0;//index for while loop
int ky=0;//key


    ofstream minfilt;minfilt.open("filter data.txt",ios_base::app);
    minfilt<<"\n"<<" entered new call";
while (i<180)
{
    double rf;
    int str;
    if(laser_range[i][0]<=3200&&ky==0)//if reading is greater 3000
     {  ky=1;
        rf=laser_range[i][0];//reference point
        str=i;//start of leg proposed
        i=i+1;//increment
     }
    else if(laser_range[i][0]>3200&&ky==0)//if reading not less than
3000
    {
        i=i+1;
```

```
    }

    if(ky==1)//if ref point found
    {
        ky=0;
        double x1=rf*cosd(str);
        double y1=rf*sind(str);
        double x2=laser_range[i][0]*cosd(i);
        double y2=laser_range[i][0]*sind(i);
        double dstr=sqrt(pow(x1-x2,2)+pow(y1-y2,2));//length of segment

        double rng1=rf;//range equal to refernce
        double rng2=laser_range[i][0];//next point range
        double bmp=abs(rng1-rng2);//bump in range

        int loopent=0;//index to confirm while loop is entered
        while(dstr<300&&i<180&&bmp<120)//to stop after 300 mm max
expected length
        {
           i=i+1;
           x2=laser_range[i][0]*cosd(i);
           y2=laser_range[i][0]*sind(i);
           dstr=sqrt(pow(x1-x2,2)+pow(y1-y2,2));//length of segment

           rng1=laser_range[i-1][0];
           rng2=laser_range[i][0];
           bmp=abs(rng1-rng2);
           loopent=1;
           }

        if (i-str<2)
        {
             i=i+1;
             arr2.push_back(str);arr2.push_back(i-1);
        }


        else if(i-str>=find_interv(laser_range[str][0]))  //if dynamic
leg found
    {
       if(str>2&&i<178)
       {minfilt<<"\n"<<" entered as str and i are "<<str<<" "<<i<<" ";
          double ff[181][1]={0};
    for (int l=str-2;l<i+3;l++)
    {minfilt<<"\n"<<"for loop at l="<<l;
       double minn=laser_range[l][0];
       int mindx=l;
```

```
     minfilt<<"\n"<<"minn llaser[l] laser[l-1] laser[l+1] "<<minn<<"
"<<laser_range[l-1][0]<<" "<<laser_range[l+1][0];
       if(laser_range[l-1][0]+100<minn)
       {minfilt<<"\n"<<"mindx and minn "<<mindx<<" "<<minn;
           minn=laser_range[l-1][0];
           mindx=l-1;
       }


       if(laser_range[l+1][0]+100<minn)
       {minfilt<<"\n"<<"mindx and minn "<<mindx<<" "<<minn;
           minn=laser_range[l+1][0];
           mindx=l+1;
       }

       ff[mindx][0]=ff[mindx][0]+1;
    }


    if(ff[str][0]>1||ff[i-1][0]>1)
    {minfilt<<"\n"<<"pushing back "<<str<<" "<<i-1;
        arr.push_back(str);arr.push_back(i-1);
    }




       }
        }

    }}//here is supposed brackets!!!!!!

//%now we use dyn function'

ind.clear();
for (int iu=0;iu<arr.size()/2;iu++)
{

   if(arr[2*iu]!=arr[(2*iu)+1])//was arr[2*iu]!=arr[2*iu]+1
    {
      ind.push_back(arr[2*iu]);ind.push_back(arr[(2*iu)+1]);//same here
    }
}
if(dyn==1)

   {
   fr2.clear();
   for(int i=0;i<ind.size();i++)
```

```
    {
    fr2.push_back(ind[i]);
    }

    for(int i=0;i<181;i++)

    {
    frp2[i][0]=laser_range[i][0];
    }
    vector<double> rest;DOI.swap(rest);
    ofstream cor;cor.open("cors.txt",ios_base::app);
          for (int f=0;f<181;f++)
          {
           cor<<frp[f][0]<<" ";
          }
          cor<<"\n";
          for (int f=0;f<181;f++)
          {
           cor<<frp2[f][0]<<" ";
          }cor<<"\n"<<laser_pose[0][0];
           cor<<"\n"<<"\n";

    DOI=dyn_det(fr,frp,fr2,frp2,sqrt(pow(dx,2)+pow(dy,2)),dth,tme);


    fr.clear();
     for(int i=0;i<ind.size();i++)

    {
    fr.push_back(fr2[i]);
    }

    for(int i=0;i<181;i++)

    {
    frp[i][0]=frp2[i][0];
    }


   }

if(dyn==0)
 {
    for(int i=0;i<ind.size();i++)

    {
```

```
   fr.push_back(ind[i]);
   }

   for(int i=0;i<181;i++)

   {
   frp[i][0]=laser_range[i][0];
   }

    dyn=1;
 }

 }
```

Appendix H.3: Find_interv function

```
int find_interv(double rng)//find interval expected for dynamic object
{
int interv;
if(rng>=2200)//?????
{
    interv=3;
}


if(rng>=1500&&rng<2200)
{
    interv=4;
}


if(rng>=1200&&rng<1500)
{
    interv=6;
}


if(rng>=1000&&rng<1200)
{
    interv=8;
}

if(rng<1000)
{
    interv=10;
}
return interv;
```

```
}
```

Appendix H.4: maxvec function code

```
double maxvec(vector<vector<double>> obj)//find max of obj function for
dynamic
{
double val=obj[0][0];
double val2=val;
for (int i=1;i<obj.size();i++)
{
  val2=obj[i][0];

if(val2>val){val=val2;}
}
return val2;
}
```

Appendix H.5: Code to acquire dynamic objects for matlab display

```
int o=1;



//robot.setVel(200);
double tt=(clock()-lastime)/(double)
CLOCKS_PER_SEC;//cout<<"\n"<<"\n"<<tt<<"\r"<<"\n"<<"\n"<<"\n"<<"\n"<<"\
n"<<"\n"<<"\n";//lastime=clock();
if(tt>0.1){lastime=clock();simdyn++;
getLaser1(laser);ofstream dynp;dynp.open("dynp.txt");ofstream
dynk;dynk.open("dynke.txt");
//for(int e=0;e<181;e++)
//{laserMotion[e][0]=(0.5*laserMotion[e][0])+
laserFrame[e][0]=laser_range[e][0]*cosd(laser_range[e][1]);
//laserFrame[e][1]=laser_range[e][1]*sind(laser_range[e][1]);
{
ofstream dyn;dyn.open("dyna.txt",ios_base::app);for(int e=0;e<181;e++){
dyn<<laser_range[e][0]<<"  ";dynp<<laser_range[e][0]<<"
";}dyn<<";"<<"\n";dyn.close();dynp.close();dynk<<o;dynk.close();//}
ofstream dyn1;dyn1.open("dyna1.txt",ios_base::app);{dyn1<<tt<<"
"<<laser_pose[0][0]<<"  "<<laser_pose[1][0]<<"  "<<laser_pose[2][0]<<"
";}dyn1<<";"<<"\n";dyn1.close();}}
//break;

if(my.getButton(8)&&gotit==1){gotit=0;cout<<"8";pos2=robot.getPose();sa
mp<<"\n"<<"distance "<<robot.getOdometerDistance()<<" rot pose
```

```
"<<pos2.getTh()<<" ";for(int e=0;e<181;e++){samp<<laser_range[e][0]<<"
";}samp<<";"<<"\n";}
if(my.getButton(9)&&gotit==0){gotit=1;cout<<"9";}
```

Appendix I

Image processing codes

Appendix I.1: MATLAB codes for preparing neural network

Part 1: Data preparation

```
arrn=[];

for i=1:823%for negative flows
if(negone(i,3)-negone(i,1)<0)%if floww vector negative
       af=abs(atand((negone(i,4)-negone(i,2))/(negone(i,3)-
negone(i,1))));%angle of flow vector

if(af<30)%if vector is not due to noise
arrn=[arrn;negone(i,:)];%add to array of negative values
end
end

if(negtwo(i,3)-negtwo(i,1)<0)%repeat with data for two degree rotation
       af=abs(atand((negtwo(i,4)-negtwo(i,2))/(negtwo(i,3)-
negtwo(i,1))));

if(af<30)
arrn=[arrn;negtwo(i,:)];
end
end

if(negthree(i,3)-negthree(i,1)<0) %repeat with data for three degree
rotation
    af=abs(atand((negthree(i,4)-negthree(i,2))/(negthree(i,3)-
negthree(i,1))));
if(af<30)
arrn=[arrn;negthree(i,:)];
end

end

end


arrp=[];
for i=1:823%for positives
if(posone(i,3)-posone(i,1)>0)
       af=abs(atand((posone(i,4)-posone(i,2))/(posone(i,3)-
posone(i,1))));
```

```
if(af<30)
arrp=[arrp;posone(i,:)];
end
end

if(postwo(i,3)-postwo(i,1)>0)
       af=abs(atand((postwo(i,4)-postwo(i,2))/(postwo(i,3)-
postwo(i,1))));

if(af<30)
arrp=[arrp;postwo(i,:)];
end
end

if(posthree(i,3)-posthree(i,1)>0)
    af=abs(atand((posthree(i,4)-posthree(i,2))/(posthree(i,3)-
posthree(i,1))));
if(af<30)
arrp=[arrp;posthree(i,:)];
end

end

end
```

Part2: Training the network

```
cmb=[arrn; arrp ;zero];%combine arrays for negative positive and zero
optical flows
cmb(:,[1 3])=cmb(:,[1 3])-320;
cmb(:,[2 4])=cmb(:,[2 4])-240;
indice=randperm(length(cmb));%select random vectors for training and
different ones for validation
indice=indice(1:4500);
p=cmb(indice,1:2)'/100;
p=[p ;cmb(indice,5)'];%creat vector of inputs
flow=[cmb(indice,3)-cmb(indice,1)]';%xoptical flow
flow2=[cmb(indice,4)-cmb(indice,2)]';%y optical flow

indice=randperm(length(cmb));
indice=indice(1:1500);
pv=cmb(indice,1:2)'/100;%validation vecotr
pv=[pv ;cmb(indice,5)'];
flowv=[cmb(indice,3)-cmb(indice,1)]';
flowv2=[cmb(indice,4)-cmb(indice,2)]';%y
```

The network was then trained using the commands:

```
net=newff(p,[flow;flow2],5,{'tansig'}); %creaate nonlinear feedforward
network with 5 neurons with tan sigmoid
net.inputs{1}.processFcns={}; %do not scale inputs
net.outputs{2}.processFcns={};%do not scale outputs
net.trainparam.min_grad=1e-0010; %set minimum gradient for training alg
net.trainparam.max_fail=100; %max times to repeat before failure
train(net,p,[flow;flow2]); %train
```

Part 3: Simulating the network

```
fd=sim(net,pv);
errr=(fd(1,:)-flowv);
sum(abs(errr))/size(errr,2)
figure(1)
plot(1:size(errr,2),errr,'.r')
xlabel('index of validation data')
ylabel('error in x component of optical flow in pixels')
Title('Error plot X component')
errr=(fd(1,:)-flowv2);
sum(abs(errr))/size(errr,2)
figure(3)
plot(1:size(errr,2),errr,'.r')
xlabel('index of validation data')
ylabel('error in y component of optical flow in pixels')
title('Error plot Y component')
```

Appendix I.2: Optical flow code

```
ArJoyHandler my;//inistantiate joystick
     my.init();//initialise joystick
ArDPPTU ptu(&robot);
ptu.init();
int x;
int y;
x=ptu.getMaxPosPan();
y=ptu.getMaxPosTilt();
if(ptu.getMaxPosPan()>ptu.getMaxNegPan()){x=ptu.getMaxNegPan();}
if(ptu.getMaxPosTilt()>ptu.getMaxNegTilt()){x=ptu.getMaxNegTilt();}
my.setSpeeds(-0.1*x,-0.1*y);

              //ArUtil::sleep(3000);
              //ptu.setDeviceConnection(con);


        //start camera
        Camera cam;//inistantiate camera object
        PGRGuid *pguid=NULL;//instantiate connection guide setting to null will
make it look for anywhere the camera is connected.
     Error error;// instantiate error to output error message (optional)
```

```
      // Connect to a camera
    error = cam.Connect(pguid);// connect and return error if connection fails
    if (error != PGRERROR_OK)//if failed return -1 exit
    {
       // return -1;
    }

       ArUtil::sleep(2000);//sleep to wait for camera to connect

       int frstfr=0;//key to indicate if first image frame was acquired

       ofstream imd;imd.open("dats.txt",ios_base::app);

       int acc;
       while(1){/*while loop is created to ensure infnite operation of this code*/
              //this section is for displaying contours remove to take pictures
also displaying optical flow has been disabled
              Mat fr1;
              Mat neural;//neural
              Mat diff;//neural difference
              Mat mono_fr1;//mat to hold mono
              Rect myROI;

              if(frstfr==0)//if no frame taken then take it!
              {
                     double snrng=laser->currentReadingPolar(-15,15);//get front
laser readings
                     cout<<"\n"<<snrng;
                  acc=0;
                     if(snrng>1400)
                     {
                            acc=1;
                     }


                     Image rawImage;//instantiate image
                     double startTime=clock();

                     error = cam.StartCapture();//capture image and return error if
any.
                     // Retrieve an image
                     error = cam.RetrieveBuffer( &rawImage );//save  image and
return error if any
                     // Create a converted image
                     Image convertedImage;//instantiate class for converted image


                     error = rawImage.Convert( PIXEL_FORMAT_RGB, &convertedImage
);//convert to RGB
                     JPEGOption uu;//get instant of JPEGoption
                     convertedImage.Save("fr1.jpeg",&uu);//save as frame 1

                     unsigned int rowBytes
=(double)convertedImage.GetReceivedDataSize()/(double)convertedImage.GetRows();
              fr1 = Mat(convertedImage.GetRows(), convertedImage.GetCols(), CV_8UC3,
convertedImage.GetData(),rowBytes);//frame one as RGB MAt.
              neural =fr1.clone();
                     diff=fr1.clone();
```

```
                        if(acc==1)
                        {
                                double ang=atan(280/snrng)*21.223*180/3.142;
                                myROI=Rect(0,240,640,(int)ang-2);
                                Mat sonimg=fr1(myROI);
                //              fr1=sonimg.clone();
                        }
                        else if(acc==0)
                        {
                                myROI=Rect(0,240,640,240);
                                Mat sonimg=fr1(myROI);
                //              fr1=sonimg.clone();
                        }
                        cvtColor(fr1, mono_fr1, CV_BGR2GRAY);//convert to gray
                        cout <<"\n"<<"time to capture and convert image "<< double(
clock() - startTime ) / (double)CLOCKS_PER_SEC<< " seconds." << endl;
                        frstfr=1;
                        imshow("my image 1",fr1);
                        imshow("my image 1m",mono_fr1);
                        waitKey(0); //wait infinite time for a keypress
                        robot.move(100);
                        //robot.setDeltaHeading(-2.5);
                        ArUtil::sleep(1000);
                }



                if(frstfr==1)//if  frame taken
                {
                        frstfr=0;
                        Image rawImage;//instantiate image
                        double startTime=clock();

                        error = cam.StartCapture();//capture image and return error if
any.
                        // Retrieve an image
                        error = cam.RetrieveBuffer( &rawImage );//save  image and
return error if any
                        // Create a converted image
                        Image convertedImage;//instantiate class for converted image

                        error = rawImage.Convert( PIXEL_FORMAT_RGB, &convertedImage
);//convert to RGB
                        JPEGOption uu;//get instant of JPEGoption
                        convertedImage.Save("fr2.jpeg",&uu);//save as frame 1
                        unsigned int rowBytes
=(double)convertedImage.GetReceivedDataSize()/(double)convertedImage.GetRows();
                Mat fr2 = Mat(convertedImage.GetRows(), convertedImage.GetCols(),
CV_8UC3, convertedImage.GetData(),rowBytes);//frame one as RGB MAt.
                        if(acc==1)
                        {
                                Mat sonimg=fr2(myROI);
        //                      fr2=sonimg.clone();
                        }
                        else if(acc==0)
                        {
                                myROI=Rect(0,240,640,240);
```

```
                        Mat sonimg=fr2(myROI);
                //              fr2=sonimg.clone();
                }
                Mat mono_fr2;//mat to hold mono
                cvtColor(fr2, mono_fr2, CV_BGR2GRAY);//convert to gray



                cout <<"\n"<<"time to capture and resort "<< double( clock() -
startTime ) / (double)CLOCKS_PER_SEC<< " seconds." << endl;
                startTime=clock();
                //do optical flow
                vector<Point2f> corn;//corner points for optical flow first
frame
                vector<Point2f> corn2;//corner points or optical flow second
frame
                Mat stat;
                Mat erre;

                goodFeaturesToTrack(mono_fr1,corn,300,0.01,10);//get features
from image
                calcOpticalFlowPyrLK(mono_fr1,mono_fr2,corn,corn2,stat,erre);
                cout <<"\n"<<"time to optical flow "<< double( clock() -
startTime ) / (double)CLOCKS_PER_SEC<< " seconds." << endl;

                double robv=0;
                double pxls=0;double noisey=0;//noise in optical flowy
                double numneg=0;double numpos=0;double numnegy=0;double
numposy=0;
                double y2=0;//py^2


                double py=0;
                double four=0;
                //paper method variables
                double vi2=0;//vi^2
                double ubi=0;//ui.bi
                double vbi=0;//vi.bi
                double uvi=0;//ui.vi
                double ui2=0;//ui^2

                imd<<"\n"<<"New motion  flow"<<"\n";
                //neural
                vector<Point2f> flow=netrot(corn,-0);
                vector<double> avg(20,0.0);
                vector<double> avg1(32,0.0);
                for(int i=0;i<corn.size();i++)
                {
                        int fc=(int)stat.at<uchar>(i);
                        if(fc==1)
                        {

                                imd<<corn[i].x<<"  "<<corn[i].y<<"
"<<corn2[i].x<<" "<<corn2[i].y<<";"<<"\n";
                                double fc2=(corn2[i].x-
corn[i].x);//sqrt(pow(corn[i].x-corn2[i].x,2.00)+pow(corn[i].y-
corn2[i].y,2.00));//
```

```
	//if(fc2<0){numneg=numneg+1;cout<<"\n"<<"neggg";}else
if(fc2>0){numpos=numpos+1;cout<<"\n"<<"pos";}
                                double fc3=(corn2[i].y-corn[i].y)*0.047118;
                                      if(fc3<0){numnegy=numnegy+1;}else
if(fc3>0){numposy=numposy+1;}
                                noisey=noisey+pow(fc3/0.047118,2);
                                //matlab vote method
                                double htc=corn[i].y;
                                double htc1=corn[i].x;
                                int avgin=ceil(htc/24);
                                int avgin1=ceil(htc1/20);
                                avg[avgin-1]=avg[avgin-1]+(fc3/0.047118);
                                avg1[avgin1-1]=avg1[avgin1-1]+(fc2);


                                //paper method varialbes
                                vi2=vi2+pow(fc3/0.047118,2);
                                ui2=ui2+pow(fc2,2);
                                double bi=(corn[i].x*fc3/0.047118)-
(corn[i].y*fc2);
                                ubi=ubi+(fc2*bi);
                                vbi=vbi+(fc3*bi/0.047118);
                                uvi=uvi+(fc3*fc2/0.047118);


                                //now workout vehicle forward motion using v=d X
opf X sin(a);
                                double pxlang=(corn[i].y)*0.047118;//pixel angle
in camera
                                double xang=(320-corn[i].x)/21.33;//angle in
laser corresponding to pixel location
                                double pxlrn=laser->currentReadingPolar(xang-
0.5,xang+0.5);//range of feature ppixel is refering to


	//robv=robv+(pxlrn*fc3/sin(3.142*pxlang/180));//supposedly the robot
velocity

                                //new method
                                double k1=sin(3.142*pxlang/180)-
(cos(3.142*pxlang/180)*pxlang);
                                double k2=cos(3.142*pxlang/180);

                                robv=robv+(pxlrn*fc3/(k1+(k2*fc3)));
                                cout<<"\n"<<pxlang<<" "<<xang<<"  "<<pxlrn<<"
"<<fc3<<" "<<k1<<" "<<k2<<" "<<(pxlrn*fc3/(k1+(k2*fc3)))<<"\n";
                    pxls=pxls+1;

	//circle(img2,corn2[i],0.06*fc2,Scalar(255,200,0),1,3,0);

	//circle(fr2,corn2[i],1*abs(fc2)/1.0047118,Scalar(255,200,0),1,3,0);
                                circle(fr2,corn[i],3,Scalar(200,200,100),2,3,0);
                            line(fr2, Point(corn[i].x, corn[i].y),
Point(corn2[i].x,corn2[i].y),Scalar(0,00,0),1,8,0);
```

```
	circle(neural,corn[i],3,Scalar(200,200,100),2,3,0);
                            line(neural, Point(corn[i].x, corn[i].y),
Point(corn[i].x+flow[i].x,corn[i].y+flow[i].y),Scalar(0,00,0),1,8,0);
	 circle(diff,corn[i],3,Scalar(200,200,100),2,3,0);
                            line(diff, Point(corn[i].x, corn[i].y),
Point(corn[i].x+corn[i].x+flow[i].x-corn2[i].x,corn[i].y+corn[i].y+flow[i].y-
corn2[i].y),Scalar(0,00,0),1,8,0);

                            //my method 31/1

	py=py+(fc3/0.047118);y2=y2+pow(fc3/0.047118,2);four=four+4;

                        }
                }

            //matlab method y
                    int sgnchng=0;
                    int crntsgn=1;
                    int t=0;int tc=0;
                    while (t<20-2)
                    {
                            int sgn[3][1];
                                sgn[0][0]=avg[t]/abs(avg[t]);//sign of first
element
                                sgn[1][0]=avg[t+1]/abs(avg[t+1]);//sign of 2nd
element
                                sgn[2][0]=avg[t+2]/abs(avg[t+2]);//sign of 3rd
element
                                   int modsgn=0;
                            for(int y=0;y<3;y++)
                            {
                                    if(sgn[y][0]<0)
                                    {
                                            modsgn=modsgn+1;
                                    }
                            }

                            if(modsgn>1)
                            {
                                    crntsgn=-1;
                            }
                            else
                            {
                                    crntsgn=1;
                            }

                            if(sgnchng==0)
                            {
                                    sgnchng=crntsgn;
                            }
                            else
                            {
                                    if(sgnchng/crntsgn<0)
                                    {
                                            tc=t;
                                            t=100;
                                    }
```

```
                t=t+1;
        }
        }

        //x matlab method
        sgnchng=0;
        crntsgn=0;
        t=0;int tcy=0;
        while (t<32-4)
        {
               int sgn[5][1];
                   sgn[0][0]=avg1[t]/abs(avg1[t]);//sign of first
element
                   sgn[1][0]=avg1[t+1]/abs(avg1[t+1]);//sign of 2nd
element
                   sgn[2][0]=avg1[t+2]/abs(avg1[t+2]);//sign of 3rd
element
                      sgn[3][0]=avg1[t+3]/abs(avg1[t+3]);//sign of 3rd
element
                      sgn[4][0]=avg1[t+4]/abs(avg1[t+4]);//sign of 3rd
element
                      int modsgn=0;
               for(int y=0;y<5;y++)
               {
                      if(sgn[y][0]<0)
                      {
                             modsgn=modsgn+1;
                      }
               }

               if(modsgn>3)
               {
                      crntsgn=-1;
               }
               else
               {
                      crntsgn=1;
               }

               if(sgnchng==0)
               {
                      sgnchng=crntsgn;
               }
               else
               {
                      if(sgnchng/crntsgn<0)
                      {
                             tcy=t;
                             t=100;
                      }
                      t=t+1;
        }
        }


        //paper method
        double det=(ui2*vi2)-(pow(uvi,2));
        double xfoe=((vi2*ubi)-(uvi*vbi))/det;
```

```
                        double yfoe=((ui2*ubi)-(uvi*ubi))/det;
                        yfoe=480-(tc*24);xfoe=20*tcy;
                        circle(fr2,Point(xfoe,yfoe),3,Scalar(200,50,50),2,3,0);
                        double ae=four+(2*py)+y2;double be=-1*(y2+(2*py));
                        double res1=(-1*be+sqrt(pow(be,2)-(4*ae*y2)))/(2*ae);
                        double res2=(-1*be-sqrt(pow(be,2)-(4*ae*y2)))/(2*ae);
                        double k11=sqrt(res1);double k12=-1*sqrt(res1);
                        double k21=sqrt(res2);double k22=-1*sqrt(res2);
                        double foey1=pow(k11,2)*(py-(four*0.5));
                        double foey2=pow(k12,2)*(py-(four*0.5));
                        double foey3=pow(k21,2)*(py-(four*0.5));
                        double foey4=pow(k22,2)*(py-(four*0.5));
                        //cout<<"\n"<<" FOEY "<<foey1<<"  "<<foey2<<"  "<<foey3<<"
"<<foey4<<"\n";
                        cout<<"\n"<<" FOEY "<<xfoe<<"  "<<yfoe<<"\n";

                        cout<<"vel "<<(double)robv/pxls<<"\n";
                        cout<<"the average of noise "<<sqrt(noisey/corn.size())<<"
percentage right "<<(double)(numpos/(numneg+numpos))*100<<"  percentage left
"<<(double)(numneg/(numneg+numpos))*100;
cout<<"/n"<<"  percentage right "<<(double)(numposy/(numnegy+numposy))*100<<"
percentage left "<<(double)(numnegy/(numnegy+numposy))*100;

                        //copy latest frame to previous
                        /*fr1=fr2.clone();
                        mono_fr1=mono_fr2.clone();
                        double snrng=laser->currentReadingPolar(-15,15);//get front
laser readings
                        acc=0;
                        if(snrng>1400)
                        {
                                acc=1;
                        }
                        if(acc==1)
                        {
                                double ang=atan(280/snrng)*21.223*180/3.142;
                                myROI=Rect(0,240,640,(int)ang-2);
                        }*/



                        imshow("my image 2",fr2);
                        imshow("my image 2m",mono_fr2);
                        imshow("neural expectation",neural);
                        imshow("difference between two",diff);
                        waitKey(); //wait infinite time for a keypress
                //      break;

                        destroyAllWindows();
                }
```

```
}
```