ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΗΜΑΤΩΝ, ΕΛΕΓΧΟΥ ΚΑΙ ΡΟΜΠΟΤΙΚΗΣ

Πλοήγηση κινούμενου ρομπότ σε άγνωστο περιβάλλον προς ένα προκαθορισμένο στόχο
Mobile robot navigation through an unknown environment towards a predefined target

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Χαράλαμπος Μ. Ρωσσίδης

Επιβλέποντες:
Δρ. Κωνσταντίνος Τζαφέστας - Επίκουρος Καθηγητής ΗΜΜΥ ΕΜΠ
Δρ. Στασινός Κωνσταντόπουλος - ΕΚΕΦΕ Δημόκριτος

July 16, 2014

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΗΜΑΤΩΝ, ΕΛΕΓΧΟΥ ΚΑΙ ΡΟΜΠΟΤΙΚΗΣ

Πλοήγηση κινούμενου ρομπότ σε άγνωστο περιβάλλον προς ένα προκαθορισμένο στόχο
Mobile robot navigation through an unknown environment towards a predefined target

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Χαράλαμπος Μ. Ρωσσίδης

Επιβλέποντες:
Δρ. Κωνσταντίνος Τζαφέστας - Επίκουρος Καθηγητής ΗΜΜΥ ΕΜΠ
Δρ. Στασινός Κωνσταντόπουλος - ΕΚΕΦΕ Δημόκριτος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την Τετάρτη 16.7.2014

Δρ. Κωνσταντίνος Τζαφέστας	Δρ. Στασινός Κωνσταντόπουλος	Δρ. Αλέξανδρος Ποταμιάνος
Επίκ. Καθηγητής ΗΜΜΥ ΕΜΠ	ΕΚΕΦΕ Δημόκριτος	Αν. Καθηγητής ΗΜΜΥ ΕΜΠ

July 16, 2014

Χαράλαμπος Μ. Ρωσσίδης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Acknowledgements

This project sums up all the effort, the passion and the devotion that have characterised me during the past five years in my pursuit for knowledge and specialisation in the fields of electrical and computer engineering. It has been quite a journey indeed and the truth is that many people deserve my gratitude.

To begin, I would like to thank my supervisors. To Dr. Stasinos Konstantopoulos, thank you for the steady coaching, for the availability and the priceless advice. Thank you for the flexibility you provided that allowed me to put my personal touch on this project. I would like to thank Dr. Costas Tzafestas for the eye opening courses on Robotics during my studies at the school of ECE-NTUA. It was the inspiration gained during those courses that have always fuelled my passion to pursue not only this project but the fields of Robotics, Automation and Control as well. Also, thank you for your availability and advice, especially for reviewing this project.

It would be an omission not to express my acknowledgement to the National Centre for Scientific Research "Demokritos" for the availability on expensive equipment like the differential drive robot (Sek) used for this project.

Special greetings to Mr. Andreas Lydakis, the operator of Sek that despite his crowded schedule and the fact that he had been working on a different project, he has always been there to help with anything I needed. I would also like to thank my best friend Mr. George Trigeorgis as well as my friend Mr. Panayiotis Kkolos for their advice on issues related to programming and their continuous support.

To my parents, thank you for your lifetime support. Thank you for being there no matter what, thank you for your care and for financing my life and education all those years.

Finally, I would like to thank my closest friends and classmates for without their company, support and encouragement the whole task of studying abroad for five years would be unachievable.

Abstract

This thesis project describes an implementation of a Human Computer Interaction (HCI) system for intelligent mobile robots. It aims to provide the ability to indoor mobile robots to understand and execute direction orders described in a pseudo-human language. The robot is positioned in an unknown indoor environment and it is given directions like ”Go to the third office on your right”.

To achieve that, the robot should be able to create a map of its environment and localise itself in the map. Thus, Simultaneous Localisation And Mapping (SLAM) techniques are used. Two key novel challenges are confronted. Firstly, the navigation towards a target in a yet unknown map, and secondly, how informative and useful the map is.

The main objective of existing navigation techniques, is to drive the robot towards the unknown area of the map in order to minimise the yet unseen environment. That is not enough if the robot must move towards a predefined target. In order to achieve this kind of behaviour, an existing navigation technique is augmented.

Secondly, the robot should be able to recognise entities that signify the way-points given by the human. To cope with that, a map of entities like door, chair, box, etc. is created and provided to to the robot. To identify these entities, pattern recognition techniques may be employed. The result is a map of objects that can be used not only for navigation and path planning, but also for a better illustration of the actual map.

Keywords

mobile robot navigation, pseudo-human language, navigator, path planner, trajectory planner, utility function, SLAM, EKF-SLAM, scanSLAM, LCscanSLAM, LC-SLAM

Contents

List of Figures

Chapter 1

Introduction

Although it is really unclear what a robot is, nowadays everyone has an impression. Truth is that robots have spread in a variety of places performing tasks ranging from simple assembly lines to space exploration. The versatility of the application fields causes robots to be extremely different depending on their working environment. However, one can identify a machine as a robot if it has the following attributes:

A robot is a machine that can **sense**, **think** and **act**. The first two capabilities are those who differentiate a robot from any other machine. Thus, a robot uses sensors to perceive its environment. Using this perception, an algorithm is responsible to do the reasoning and make decisions on how the robot should act. Finally, it designs its actions which can either affect its inner state or interact with the environment through its actuators. A robot possesses three basic properties:

- Programmability
  It has a computer available that implements the necessary algorithms that can be changed to augment its behaviour.

- Mechatronic Device
  It is a machine that can interact with its environment through mechanical parts.

- Adaptability, Versatility, Flexibility
  It is a complex intelligent system that can adapt to different environment and task requirements.

Breaking down the vastness of robot applications into distinct categories is definitely nor straightforward neither it can be done in a unique way. However, robots can be classified as *Industrial Manipulators*, which are robotic arms fixed on a static base, and *Mobile Robots*, which vary from car like vehicles to drones and submarines.

Machines in those two fields confront very different challenges. While the important feature of an industrial arm is accuracy and dexterity, a mobile robot focuses on localisation issues and autonomy. Focusing on mobile robots, they can be further subdivided to robots that interact with humans and those that do not.

Nowadays, robots tend to break out of the industries and be used increasingly often in indoor environments. Thus, emanates the need to design intelligent robots that have the ability to communicate with humans in a high level of abstraction. Providing this ability to a robot, would result in an increased ease of use and in an easier acceptance by the society, boosting the production of indoor commercial use of highly efficient mobile robots.

This project tries to resolve issues that arise when mobile robots interact with humans. It is a Human-Computer Interaction system that gives the ability to the robot to understand direction orders described in human language. It builds on existing techniques used for mobile robot localisation and mapping in unknown environments, and expands its capabilities by integrating the ability to move towards an abstractly predefined target within the unknown environment.

The ultimate contribution of this project consists of two factors:

- The design of a pseudo-human language framework that feels human while being understood by a robot and enables the high level Human - Robot communication.
- The description of the necessary components of a system that enables a robot to execute the directions given in the pseudo-human language.

This thesis project is structured as follows: Chapter 2 describes previous work on mobile robot localisation and mapping. Chapter 3 describes the objectives and the challenges encountered in the development of this system. In Chapter 4 a complete system is designed from scratch to serve as a host for the innovation of this project which lies in two key parts. Firstly, a pseudo-human language framework is designed to formalise the human - robot communication. Secondly, a standard subsystem called the Navigator is augmented to facilitate the new features. In Chapter 5 the functionality of the key components of the developed system is demonstrated via a simulation that has been contacted. Finally, in Chapter 6 the overall performance of the system is tested through a set of carefully designed tests. Final remarks and future work are presented in Chapter 7

Chapter 2

Theoretical Background

2.1 SLAM

This chapter introduces the core component of a mobile robot moving in an unknown environment called SLAM. A set of key features of SLAM systems is outlined and the essential SLAM variations are described in brief.

2.1.1 Overview

During the last decades, a great attention has been focused on the solution of the *Simultaneous Localisation And Mapping (SLAM)* problem by the mobile robotics community. It is the problem that "asks if it is possible for a mobile robot to be placed at an unknown location in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its location within this map."[5] SLAM formulations have been implemented for indoor, outdoor, underwater and airborne systems, each of them facing different challenges and difficulties. A great amount of solutions has been proposed in the literature and SLAM can now be considered as a solved problem.

Although it is not explicitly mentioned in the bibliography, each SLAM variant needs a navigator to work properly. A *Navigator* is a system that produces the control signals for the robot to move towards the next desired location at each time step. A great variety of navigators have been designed in a strain to maximise the final quality of the SLAM map. The effort so far has been focused on minimising the yet unseen map while keeping the uncertainty in low levels.

As a basic subset of capabilities, a mobile robot should be able to move and perceive its environment. Focusing on indoor, wheeled vehicles, one can consider movement as wheel driving and perception as either combining, or using as a stand alone sensor one or a combination of a laser sensor, a camera, an RGBD sensor (e.g. the KINECT sensor), ultra-sound sensors, etc.

In order to create a map, the robot shall move within the world and take measurements of the environment using its sensors. Moving changes the robot's *pose* (i.e.

location and orientation). Determining the new pose of the robot after a move direction is executed, is not an easy task. The new pose is always prone to a significant uncertainty due to external factors such as wheel slipping, air currents, inaccurate environment modelling, etc. In addition, the perception of the environment by the robot, i.e. the measurements provided by its sensors, are noisy. Consequently, one of the main hassles to cope with is the existence of noise. Modeling the system in a stochastic framework in order to cope with the existence of noise, gave birth to the probabilistic SLAM [4, 10]. Since, a variety of different probabilistic methods emerged, all of them using a probabilistic filter as a *kernel*, to confront the noise issues. Namely, the most popular ones are the *Extended Kalman Filter (EKF)*, and the *Rao Blackwellized Particle Filter*.

The *map* is a vector that contains the robot's pose and all other objects of the world. As a rule of thumb, the different methods to solve the SLAM problem are distinguished by the way they define and manipulate the world's objects. The simplest way to define an object is "The space occupied by obstacles, i.e. the positions of the environment that are not accessible by the robot". This leads to huge maps and computationally infeasible solutions. More modern approaches tend to group these positions into sets that define a bigger part of the space occupied by the obstacles. This is exactly what T. Bailey et al. do in their *scanSLAM* solution [1, 2, 3] which is the method used as a base to implement the contributions of this thesis as described above.

The SLAM process works as follows. A kernel (e.g. EKF or Rao Blackwellized) creates and maintains a map. It works in discrete time steps incrementally expanding and improving the map. Two external systems are responsible to provide an observation and a movement order at each time step. Using the current observation, the kernel updates the map estimation in order to reduce its uncertainty. This means that in case that the observation is of bad quality (i.e. we are pretty unsure that the position and the orientation of the observed object is correct), the current map should not be changed radically. In other words, a kernel determines the importance of the new information according to its uncertainty in order to update the map properly. Having the new, improved map in hands, the second system called the "Navigator" computes the next desired pose of the robot and provides a move order.

2.1.2 Key Components

Taking a deeper look in the components of a SLAM system, one can break them down to an indicative required set. Obviously, the methodologies used differ from implementation to implementation:

Observation Model

A robot perceives its environment using a set of sensors. An observation almost never consists of the raw sensor data. Instead, a pre-processing of the sensor measurements shall be conducted in order to produce an entity called *Observation*, and provide it to the kernel. This "Observation" could be anything from a set of laser data points to a set of fused multi-sensor data. Obviously, the kernel should be properly formulated, utilising the observation model, so that it can use this observation suitably.

System (Motion) Model

In order to identify the errors in the map, the kernel should have the ability to compute an expected map, a *Prediction*. The robot should be able to answer the question: “Given the current map and a move order, what should the world look like when I move to that position?” Apparently, the kernel should be aware of the structure of the robot, and use the move order to simulate its movement. It would then compare the new world perception with the predicted one and use the result to improve the map. Most advanced techniques (e.g. Particle Filtering SLAM), try to avoid the errors induced by an inaccurate model, by avoiding to use one (see Chapter 2.1.3). Instead, they create a set of hypothesis of the future map, and they discard the bad ones later, if they realise that they are of low quality. However, even this method, needs a mechanism to create the set of hypothesis in order to produce plausible candidates of the map and increase the system’s efficiency.

Data Association

When the robot observes its environment at the next time step, it gets a set of new data representing a portion of an object in the real world. It is of great importance to be able to identify that the currently visible object, is already saved in the map and that the robot has been here before. This will enable the kernel to decide whether it should incorporate the new observation to the map as a new object or correct the map appropriately. A detailed view of a simple data association mechanism is described in Chapter 4.2.1. It is worth to note that data association issues seem to be the Achilles heel of SLAM algorithms. If the data association fails, then the input to the kernel, the “Observation”, is faulty and consequently the results differ from the desired. It is extremely difficult to ensure the proper function of data association, because it is done in real time, and it works on real world data. However, techniques which manage to “forget” data association history, such as the Particle Filters, make the system immune to faulty associations conducted in the past, and thus, they perform better in practice.

Identification of Moving Objects

Both kernels being discussed assume that the map is static. As, moving objects can not be a part of the map, it is of great importance that the observation does not contain any currently moving object. Consequently, the system responsible for raw sensor data pre-processing should reject an observation that contains moving objects. A lot of work has been done in recognising moving objects while building a map, not only in order to reject them from observations, but to track their motion as well [14]. Focusing our attention on indoor environments such as offices and homes, the moving objects of interest are mainly humans and thus, it is important to identify and reject human walking patterns [13].

Object Removal

During the SLAM procedure, a lot of objects that seem to be static for a while, may change position. For example, a chair might be moved to a different place. These objects can practically not be classified as moving during the first observation because they are static at that time. Thus, they are saved in the map. The SLAM algorithm should be able to remove objects from the map that are not re-observed near their previous position.

Exploration Strategies and Path Planning

As mentioned before, the robot must move in order to visit unknown places and construct the map. A lot of effort has been put to design navigation algorithms that not only maximise the visited areas but minimise the map's uncertainty as well. A *Path Planner* is responsible to design a path, and a *Trajectory Planner* produces the control signals so that the robot follows the desired path. The whole process is done in a way that the followed trajectory is optimal, in terms of distance, energy consumption, motion uncertainty etc. (see Chapter 2.3). This thesis aims to enrich the path planner capabilities in a way that the main purpose of the robot is not to maximise the known map, but move towards a predefined target. This target could lie in an unknown, not yet visited part of the map. The robot gets its orders which consist of way-points such as door, corridor, staircases, etc. In order to identify those way-points, the algorithm must characterise the contents of the map and associate them with predefined object templates.

These essential features of the SLAM algorithm, are used either as parts of the whole system or as stand alone sub-systems. Traditionally, the SLAM variations have been named after the kernel used, although this is not a rule. The first successful SLAM applications used the Extended Kalman Filter (EKF) as a kernel, while in more modern variations the Rao Blackwellized Particle Filter managed to outperform it and it is currently the kernel used by the state of the art algorithm.

2.1.3 Current State of the Art SLAM

The first successful steps towards a solution to the SLAM problem, were taken when researchers started to formulate it in a stochastic framework. The genesis of the *Probabilistic SLAM* resulted in the extensive use of the EKF as a kernel. The EKF uses a motion and an observation model, both non-linear and it makes the assumption of white, zero mean, uncorrelated, Gaussian motion and observation system noise (see Chapter 2.2.2). It works in two steps, where it firstly estimates the expected map that would result after the current control is asserted, and secondly it takes an observation and improves the estimated map. A great variety of solutions has been proposed throughout the years, and the EKF SLAM has prevailed for almost 15 years.

However, the EKF SLAM has two serious disadvantages that prevent its application on large environments: it has quadratic complexity and it is very sensitive to faulty data association. Although it has been the dominant SLAM solution for so long, it has been recently outperformed by the uprising particle filters [8]. Especially, the most efficient Rao-Blackwellized Particle Filter gave birth to the currently prevailing algorithm known as *FastSLAM*. FastSLAM not only scales logarithmically with the number of Landmarks in the map, but it is less sensitive on ambiguous data association, and it consequently outperforms the EKF, especially in large environments.

Particle Filters

Particle filters make no restrictive assumptions about the robot's dynamics, or the structure of the environment. Instead, they treat the system's state and the observations as probability density functions (pdf). The conditional pdf $p(x_t|u_t, x_{t-1})$ is called the motion (or actuation) model and it is used to predict the *State* of the system (i.e. the map containing the robot's pose and all the known Landmarks). It assumes that the current state x_t depends exclusively on the previous state x_{t-1} and on the control that has just been asserted u_t.

However, the state x_t is not directly observable. The observation z_t is available instead, which is a stochastic projection of the true state. It is produced by a stochastic process, implemented by the observation (or measurement) model, which is given by the conditional pdf $p(z_t|x_t)$ The two models described above are usually highly geometric, and they capture the whole structure of the robot generalising notions such as kinematics and dynamics by introducing non-deterministic noise [11]. In contrast with EKF SLAM, Particle Filtering SLAM can manage non linear models without the need of linearisation, as well as non Gaussian noise models.

In order to use a Particle Filter for SLAM purposes, it is designed such as each particle represents a candidate state of the map. The particles are updated according to the law induced by the motion model. Each particle, a possible instance of the map, is weighted according to the observation model which diminishes the improbable particles and reinforces the most likely ones.

Unfortunately, although Particle Filters tend to provide more accurate results than the EKF approach, they do not perform well when applied to high dimensional systems, mainly due to dimensionality issues. To address this problem, a hybrid solution called FastSLAM uses the Rao-Blackwellized Particle Filter as a kernel. It actually consists of a Particle Filter to estimate the robot's pose and an Extended Kalman Filter for each Landmark, to estimate its pose.

FastSLAM

The Particle Filter is a very powerful tool for SLAM purposes. It is able to capture any kind of robot motion dynamics and cope with noise models with no Gaussianity assumption or any other imposed restriction. Sadly, a great expense has to be paid, in matter of dimensionality and computational cost. However, taking a careful look at the SLAM problem's structure, one can see that the whole potential of Particle Filter is actually not essential.

The Landmarks on the map are stationary, and as such no motion dynamics need to be captured. The designer should only ensure that the best estimation of their pose is provided. This, easier task, can be achieved by an inferior system that is on the other hand cheaper to implement. This is the idea behind the Rao-Blackwellized Particle Filter, used as a kernel by the FastSLAM Algorithm.

Along with the computational overhead reduction, many advantages emerge from this implementation, which mainly affect data association and uncertainty reduction. Each particle represents a different robot pose hypothesis, and data association is done separately for each particle. In case of faulty association, for any kind of reasons, the pose hypothesis (particle) will eventually be discarded through its weight decay.

Moreover, by cleverly implementing the filters, representing particles as binary trees of Kalman Filters, the FastSLAM algorithm can reduce its complexity down to O(M logK), where M is the number of particles, and K is the number of Landmarks. FastSLAM has been demonstrated with up to 100,000 Landmarks, problems far beyond the reach of the EKF [8].

2.1.4 scanSLAM

For the purposes of this project, an EKF SLAM variation (see Chapter 2.2.3) is used as a base system. Tim Bailey et al. have described their EKF SLAM variation called scanSLAM in their paper “Scan-SLAM : Recursive Mapping and Localisation with Arbitrary-Shaped Landmarks” [1]. Of course, assembling a complex system such as a SLAM algorithm is obviously not a simple task. Thus, they focus on the components of their system in [2] providing clarifications and recommendations for improvement. scanSLAM as described in the previous two papers is just a proof of concept of their innovative technique described below, and the components of their system are the simplest possible. Finally, an advanced technique which tries to improve the covariance estimation is described in [3]. The latter is not essential for this thesis project and it is not furtherly discussed.

Please note that scanSLAM is used abusively throughout this project. It refers to the SLAM variation of T. Bailey et al. exclusively and it must not be confused with any scan-matching SLAM techniques which existed prior to it.

Basic Idea

The central idea in scanSLAM is to find sets of points in a laser sensor scan and group them in entities called *Landmarks*. It is important to emphasise that the contribution of scanSLAM is that no templates are used to define the Landmarks, but they are arbitrary shaped segments of raw sensor data. These Landmarks should have some key properties that would allow the algorithm to identify them when the site is revisited (recall the “Data Association” requirement: Chapter 2.1.2). A local coordinate frame is assigned to each Landmark to indicate its position and orientation (i.e. its pose).

Although the current state of the art SLAM algorithms use particle filters as kernels, T. Bailey et al. chose to use the EKF, mainly because of the simplicity in which their approach could be implemented in this framework. The scanSLAM approach is no different than any other EKF-SLAM algorithm, except from the definition of the observation model. Instead of defining an observation as a point of the laser scanner, the observation is defined by a Landmark, which is represented, as far as the EKF is concerned, by its local coordinate frame, i.e. its pose. Consequently, the observation model of the EKF works on the poses of the Landmarks.

Thus, the SLAM "map" consists of the pose of the robot and the poses of the Landmarks. This provides a few advantages in contrast to the naive approach of storing each laser point in the map. Firstly, it reduces the size of the EKF state vector (i.e. the map) since large sets of points are stored in groups. Secondly, it improves the overall behaviour of the EKF because it treats the map points macroscopically. As a result, minor changes on individual points do not affect the filter drastically.

2.2 The Extended / Kalman Filter (KF and EKF)

The extended Kalman Filter used as a SLAM kernel is a generalisation of the Kalman filter which tries to address the filtering problem: Based on the available information (control inputs and observations), estimate the system's state (map) that optimises a given criteria. This paragraph aims to demonstrate how the EKF could be formulated in the SLAM framework to be used as a kernel. Firstly, the mathematical background of the Kalman and the Extended Kalman Filters is outlined concluding to the EKF SLAM.

2.2.1 The Kalman Filter (KF)

KF Overview

The Kalman filter is a linear, discrete time, finite dimensional, time-varying system that evaluates the state estimate that minimises the mean-square error (MSE) [9].

In order to use the KF, one has to define an observation model and a system's model. The KF then works in two steps: Firstly, the prediction step uses the system model and the control inputs to estimate the next state of the system. After the application of the control input, the innovation (filtering) step uses the new observation (passing the sensor readings through the observation model) to improve the estimation of the system's state.

The Kalman Filter works on the assumption that the two models are linear and the noise distributions involved are Gaussian. It can be shown that if the above assumptions apply, then the KF converges to a steady state, minimising the mean-square error between the actual system's state and the state prediction. The system's state is represented by a probability density function that, in the case that all the previously described assumptions apply, is also Gaussian. This means that it can be described only by the first and second moments (i.e. its mean and covariance).

KF Statement

System's Model:

$$x_{k+1} = A_k x_k + B_k u_k + G_k w_k \qquad\qquad k \geq 0 \tag{2.1}$$

Observation Model:

$$y_k = C_k x_k + v_k \tag{2.2}$$

Where:

$$\begin{aligned} State : x_k \in \mathbf{R}^n & \qquad SystemNoise : w_k \in \mathbf{R}^n \\ Control : u_k \in \mathbf{R}^m & \qquad MeasurementNoise : v_k \in \mathbf{R}^r \\ Observation : y_k \in \mathbf{R}^r & \\ JointCovarianceMatrix : & \qquad E\left[\begin{pmatrix} w_k \\ v_k \end{pmatrix} \begin{pmatrix} w_k^T & v_k^T \end{pmatrix}\right] \triangleq \begin{bmatrix} Q_k & 0 \\ 0 & R_k \end{bmatrix} \end{aligned}$$

Interpreting the above equations one can see that they represent a classic linear discrete time system. At time $k+1$, the state vector x_{k+1} is linearly dependent on the state at the previous time x_k, and on a control signal u_k. The white, Gaussian, zero mean noise vector w_k is responsible to model the system's error sources. Namely the control vector uncertainty, the model (A_k) inaccuracy, and other external factors such as wheel slipping etc.

The system's state can not be measured directly. Instead, a linear combination of the state vector's elements, called the observation vector y_k is available. The white, Gaussian, zero mean noise vector v_k models the observation's error sources, videlicet mainly the sensor's uncertainty.

The Kalman Filter, uses the current state x_k, the given control signal u_k and the observation y_k in order to calculate an optimal prediction of the next state x_{k+1} of the system. It approximates the system state with a Gaussian Probability Density Function, which can be described only by its mean and covariance. Thus, the goal of the KF is to compute the mean state and the covariance matrix at time $k+1$ given all information provided at time $k+1$: $\hat{x}(k+1|k+1)$ and $P(k+1|k+1)$ respectively.

KF Dynamics

To accomplish the behaviour described above, the Kalman Filter works in two cycles, the prediction and the filtering cycle.

$$\begin{tikzcd}
\mathcal{P}(x_k|Y_1^k, U_0^{k-1}) \arrow[dr, "\text{Prediction Cycle}"] & & \mathcal{P}(x_{k+1}|Y_1^{k+1}, U_0^k) \\
& \mathcal{P}(x_{k+1}|Y_1^k, U_0^k) \arrow[ur, "\text{Filtering Cycle}"] &
\end{tikzcd} \tag{2.3}$$

Where all probability density functions (pdf) are assumed to be Gaussians ($\mathcal{P} \sim \mathcal{N}$), and the sequences Y_a^b, U_a^b consist of the values of y and u respectively, from time a to time b.

So, Equations 2.3 describe the KF dynamics as follows:
Using the pdf of x_k given all observations until now (Y_1^k) and all control inputs until the previous time step (U_0^{k-1}), predict the pdf of the next time step. i.e. Compute the pdf of (x_{k+1}) given Y_1^k and all control inputs including the control currently being applied (U_0^k).
As soon as a new observation is available, improve the prediction and compute the pdf of x_{k+1} given all available observations (Y_1^{k+1}) and all control inputs until now (U_0^k)

Step 1: Prediction i.e. evaluation of $\mathcal{P}(x_{k+1}|Y_1^k, U_0^k)$ and $\mathcal{P}(y_{k+1}|Y_1^k, U_0^k)$

As mentioned before, all pdf are assumed to be Gaussian:
$\mathcal{P}(x_{k+1}|Y_1^k, U_0^k) \sim \mathcal{N}(\hat{x}_{(k+1|k)}, P_{(k+1|k)})$
$\mathcal{P}(y_{k+1}|Y_1^k, U_0^k) \sim \mathcal{N}(\hat{y}_{(k+1|k)}, P_{(k+1|k)})$

This means that there is no need to compute anything but the means $\hat{x}(k+1|k)$, $\hat{y}(k+1|k)$ and the covariance $P(k+1|k)$ which are shown [9] to be given by the Equations 2.4:

$$\begin{aligned}
\hat{x}(k+1|k) &= A_k \hat{x}(k|k) + B_k u_k \\
\hat{y}(k+1|k) &= C_{k+1} \hat{x}(k+1|k) \\
P(k+1|k) &= A_k P(k|k) A_k^T + G_k Q_k G_k^T
\end{aligned} \tag{2.4}$$

Step 2: Filtering i.e. evaluation of $\mathcal{P}(x_{k+1}|Y_1^{k+1}, U_0^k)$

It turns out that the new mean and covariance result from the correction of the current ones as follows:
Filtered State Estimate = Predicted State Estimate + Gain * Error

KF Equations Summary

Prediction:

$$\begin{aligned}\hat{x}(k+1|k) &= A_k\hat{x}(k|k) + B_k u_k\\ P(k+1|k) &= A_k P(k|k) A_k^T + G_k Q_k G_k^T\end{aligned}$$

Filtering:

$$\begin{aligned}\hat{x}(k+1|k+1) &= \hat{x}(k+1|k) + K_{k+1}[y_{k+1} - C_{k+1}\hat{x}(k+1|k)]\\ K_{k+1} &= P(k+1|k)C_{k+1}^T[C_{k+1}P(k+1|k)C_{k+1}^T + R_{k+1}]^{-1}\\ P(k+1|k+1) &= [I - K_{k+1}C_{k+1}]P(k+1|k)\end{aligned}$$

Initial Conditions:

$$\begin{aligned}\hat{x}(0|-1) &= \bar{x}_0\\ P(0|-1) &= \Sigma_0\end{aligned} \tag{2.5}$$

Remarks

1. The Kalman Filter is a linear, discrete time, finite dimension, time varying system that uses the control and observation sequences, U_0^{k-1} and Y_0^k respectively and computes an optimal estimation of the state $\hat{x}(k|k)$ with respect to the mean square error.

2. The system and the observation noise are assumed to be independent random variables. Notice that they are Gaussians with $E[w_k v_k^T] = E[v_k w_k^T] = 0$

3. The conditional error covariance matrix $P(k|k-1)$ and the gain K_k are not dependent on the observations Y_0^{k-1}

2.2.2 The Extended Kalman Filter (EKF)

EKF Overview

The very strict linearity assumptions of the Kalman Filter make it useless in practice. To improve that, the Extended Kalman Filter is formulated in order to introduce non-linear system and observation models.

The fact that in practice, the system and observation models are not linear, violate the assumption that the conditional pdf $\mathcal{P}(x_k|Y_1^k)$, $\mathcal{P}(x_{k+1}|Y_1^k)$ and $\mathcal{P}(x_{k+1}|Y_1^{k+1})$ are

Gaussian. In this case, in order to evaluate the first and second moments of the non-linear filter, the whole pdf has to be propagated, which is a heavy computational burden [9].

The Extended Kalman filter gives an approximation of the optimal estimate. To achieve this, the EKF linearises the system model around the current state prediction $\hat{x}(k|k)$ and the observation model around the next time step prediction $\hat{x}(k+1|k)$ and it applies the KF on the linearised dynamics. Thus, the need to propagate the whole non-linear pdf is avoided.

As a result, the EKF becomes a powerful tool that can be used in practice. It is computationally efficient and provides estimations that are close enough to the real state of the system. However, everything has a cost. It is important to note that,

1. there is no convergence proof (the EKF may diverge), and
2. the state predicted by the EKF is not optimal

EKF Dynamics

Generalising the KF, the EKF introduces the discrete time non-linear system and observation models. At a given time step, the system state x is given by the non-linear system model $f(.)$ evaluated at the previous time step. The uncertainties are modelled exactly as the KF through the zero mean, white Gaussian process w. The observation is obtained similarly through the non-linear observation model $h(.)$

System's Model:

$$x_{k+1} = f_k(x_k) + w_k \tag{2.6}$$

Observation Model:

$$y_k = h_k(x_k) + v_k \tag{2.7}$$

Where:

$$\begin{aligned} f_k(x_k) &: \mathbf{R}^n \to \mathbf{R}^n \\ h(x_k) &: \mathbf{R}^n \to \mathbf{R}^r \end{aligned}$$

Procedure

In order to compute an estimate using the KF, the problem has to be formulated to respect its assumptions. Thus, at each time step, the two models have to be linearised around the current working point. Obviously, the current state is unknown, so the latest prediction is used instead.

$$\begin{tikzcd}
\mathcal{P}(x_k|Y_1^k, U_0^{k-1}) \arrow[r] & \hat{x}(k|k) \arrow[dr] & \\
& & \boxed{\text{linearise } x_{k+1} = f_k(x_k) + w_k \text{ around } \hat{x}(k|k)} \arrow[dll, "\text{KF}"'] \\
\mathcal{P}(x_{k+1}|Y_1^k, U_0^k) \arrow[r] & \hat{x}(k+1|k) \arrow[dr] & \\
& & \boxed{\text{linearise } y_{k+1} = h_{k+1}(x_{k+1}) + v_{k+1} \text{ around } \hat{x}(k+1|k)} \arrow[dll, "\text{KF}"'] \\
\mathcal{P}(x_{k+1}|Y_1^{k+1}, U_0^k) \arrow[r] & \hat{x}(k+1|k+1) &
\end{tikzcd}$$

Figure 2.1: Extented Kalman Filter dynamic concept

As illustrated in Figure 2.1, the EKF works as follows:

1. Linearise the system model $x_{k+1} = f_k(x_k) + w_k$ around the last state estimate $\hat{x}(k|k)$

2. Apply the KF prediction step to compute $\hat{x}(k+1|k)$ and $P(k+1|k)$

3. Linearise the observation model $y_k = h_k(x_k) + v_k$ around $\hat{x}(k+1|k)$

4. Apply the KF filtering step to compute $\hat{x}(k+1|k+1)$ and $P(k+1|k+1)$

EKF Equations Summary

Prediction:

$$\begin{aligned} \hat{x}(k+1|k) &= f_k(\hat{x}(k|k)) \\ P(k+1|k) &= F_k P(k|k) F_k^T + Q_k \end{aligned}$$

Filtering:

$$\begin{aligned} \hat{x}(k+1|k+1) &= \hat{x}(k+1|k) + K_{k+1}[y_{k+1} - h_{k+1}(\hat{x}(k+1|k))] \\ K_{k+1} &= P(k+1|k)H_{k+1}^T[H_{k+1}P(k+1|k)H_{k+1}^T + R_{k+1}]^{-1} \\ P(k+1|k+1) &= [I - K_{k+1}H_{k+1}]P(k+1|k) \end{aligned}$$

Initial Conditions:

$$\begin{aligned} \hat{x}(0|-1) &= \bar{x}_0 \\ P(0|-1) &= \Sigma_0 \end{aligned} \tag{2.8}$$

Where:

$$\begin{aligned} F_k &\triangleq \nabla f_k|_{\hat{x}(k|k)} \\ H_{k+1} &\triangleq \nabla h_k|_{\hat{x}(k+1|k)} \end{aligned} \tag{2.9}$$

The Jacobians of the non-linear systems (see Equations 2.9) are responsible for the linearisation of the models around the desired points.

2.2.3 EKF SLAM

The Extended Kalman Filter is formulated in such a way that is extremely convenient to use for SLAM purposes. Notice that a mobile robot moves after a control signal u is applied and perceives (observes) the surrounding world through its sensors. In order to use the EKF as a SLAM kernel, it suffices to formulate the “map” as the EKF state and the sensor readings as the EKF observation.

Storing the robot’s view (map) of the world as a state, the EKF is used to produce the best possible estimate. Note that the “map” to be estimated includes not only the objects of the world but the robot’s pose as well. Thus, all the “knowledge” of the robot about its environment and its localisation within the environment is maintained in the EKF state.

At each time step, the robot moves and computes the expected view of the world, i.e. the new state based on the expected move, and the current map. As discussed before, the actual move is almost never the same as the expected one because of a variety of reasons such as, wheel slipping, inaccurate kinematic model, noise, etc. Fortunately, the EKF takes care of that using the system model noise vector w_k.

Then, the robot takes a snapshot of the world using its sensors, an observation. Using the new provided data, it compares the new view of the world with the expected one. Noticing that the objects of the world are static in respect of each other, it is obvious that the main source of uncertainty in the map is the robot's pose. This remark is crucial in order to acquire some intuition on how the EKF works in order to localise a mobile robot within a static environment. It actually, constantly fixes a bad estimation of the robot's pose with respect of a set of static objects.

The formulation of the EKF for SLAM purposes is not straightforward. Different approaches have been proposed, mainly to minimise the algorithm's computational needs. Some focus on the algorithm's complexity [6] whereas others try to reduce the actual size of the "map". The later approach is what the scanSLAM is about, proposed by Tim Bailey et al.

Last but not least, the production of the control signal is of great importance. The robot should move with a purpose. Mainly, the effort made in the past, focuses on maximising the known map while at the same time preferring trajectories that minimise the robot's pose uncertainty [12].

2.3 Navigators

This paragraph describes the system responsible for the robot motion. The so called *Navigator* uses the map to produce the control signals for the wheels so that the robot moves appropriately.

2.3.1 The need for a Navigator

Every SLAM implementation needs a Navigator in order to implement the general exploration strategy. Throughout the years that researches struggled against the SLAM challenge, the great problem has emerged. It is impossible to build a consistent map without a good exploration strategy. Unfortunately, the trajectory followed by the robot is crucial for map building as well.

Because the uncertainty of the robot's pose increases while moving, the observations tend to become unacceptably inaccurate. One of the best ways to cope with this problem is to revisit a known place of the map. Re-observing the same object and associating it with its previous position, makes it possible to improve the estimation of its real position.

A second, obvious reason that the exploration strategy is important for map building, is the need to visit the unknown areas in an optimal way. Thus, a methodical way to move through space is needed in order to visit as many places as possible moving as least as possible.

Apart from the exploration strategy, the trajectory followed is extremely significant. A mobile robot is a moving rigid body, and as such, the laws of physics apply to it.

The applied forces, its inertia and friction on the wheels are some of the elements indirectly induced by the trajectory followed. These elements, greatly affect the robot's pose uncertainty and thus, the trajectory should be carefully designed.

To sum up, the two elements of great importance are both implemented by the Navigator System:

- **The Exploration Strategy**
  i.e. which areas and when they are visited.
- **The Trajectory**
  i.e. which path will be followed and what the velocity of the robot will be at each point of the path.

2.3.2 How it works

A navigator consists of two parts, a high level goal setting system called the "Path Planner", and a trajectory designer to move from goal to goal. The path planner is responsible to choose the successive goals in order to implement the exploration strategy.

The navigator reads the current map, and computes the control signal that is responsible to drive the robot to its next target. This control signal is e.g. for a differential drive robot, the desired velocity of the wheels. The sequence of successive targets combined with the velocity along this path defines the trajectory.

The navigators being used today, employ a utility function that gives a score to each candidate next target. Maximising the utility function yields the best target to follow on the next step. Consequently, the implementation of a navigator is highly dependent on designing the appropriate utility function.

2.3.3 Utility Functions

B. Tovar et al. have described a set of characteristics that a good utility function should have as a part of their project "Planning Exploration Strategies for Simultaneous Localization and Mapping" [12]:

Re-Observe Landmarks

As referred before, re-observing Landmarks is crucial in order to improve the estimate of their pose. A widely used technique known as "Loop Closure", suggests that the robot moves towards the unknown areas of the map for a while, before it returns to a previously known place, to "close a loop". There is a trade-off on how long it shall move before it closes a loop, and how often should it return. Although the loop closure technique mostly relies on the path planner, it is important that the robot finally observes the Landmarks lying around. Thus, it should prefer positions from which maximal number of Landmarks are visible, i.e. observable by its sensors.

Visit New Areas

Maximising the visited areas of the map is not straightforward. The environment is full of obstacles that shall be avoided and moving towards an unknown area requires finding a path towards it. To achieve that, the robot shall be able to recognise the boundaries of the known world and try to expand them. Consequently, it shall prefer positions near the boundaries, that provide maximal views of unexplored areas.

Minimise Localisation Uncertainty
The main source of the uncertainty of the robot's pose that is controlled by the trajectory followed, is inertia. It causes wheel slipping and decreases the motion model's credibility. In order to minimise the effect of Inertia, a good navigator should penalise rotation and prefer trajectories along straight lines. Moreover, it should reduce start and stops as much as possible, since acceleration and decelerations tend to increase the robot's position error.

Minimise Power Consumption
Last but not least, every implemented system has to be as efficient as possible. It is obvious that decreasing the total path travelled improves the power footprint of the robot.

They designed a utility function that works in the framework of their Bayesian based SLAM implementation and integrates all of the features above. Furthermore, positions near walls and objects are discarded because many sensors become blind when the objects are very near. Their utility function optimises criteria such as information gain, uncertainty reduction, etc.

For the purposes of this project, the focus is crowned towards the path planner. Thus, a much simpler form of the proposed utility function is used as a base, in order to design the appropriate navigator. Some of the features are discarded for simplicity and they can definitely be re-introduced in an improved version of this system.

Chapter 3

Topic Description

3.1 Objectives

As robots become more and more popular in indoor environments, rises the need of them interacting with humans. Developing intelligent systems that can communicate with humans in a smooth manner is of high importance when it comes to the robot's penetration and acceptance from the society. This project tries to make a step towards this direction, by providing a mobile robot the ability of navigating through unknown environments following directions described in human language. The directions are given in a form of waypoints followed by orders just like humans describe this kind of directions. For instance, "Go straight until the end of the corridor, and turn left."

3.2 Innovation

In order to achieve the described objective, existing techniques are put in practice. As described in Chapter 2.1, the capability of moving in an unknown environment has been already studied thoroughly and a great variety of SLAM algorithms are available. However, current SLAM systems' primary objective is to create maps with the highest quality possible. To achieve that, they tend to force the robot to navigate along optimal paths in terms of map quality.

This project treats SLAM in a completely different way. The main goal is to navigate the robot towards a target described in human language through an unknown environment. Whether a good map is constructed in the meantime or not, is not of great interest. Working towards this direction, two novel elements are introduced.

- Pseudo-Human Language Framework
  An interface that connects human expression of navigation instructions and the robot's comprehension has been designed. This Language forms a framework to express directions in a way that is familiar to humans while being easily comprehended by a machine.

- Navigator Augmentation
  A Navigator is designed in a way that allows the robot to move towards the waypoints given and execute its directions. Special care has been taken so that if the robot has no directions in hand, the navigator would work as if it was executing a classic SLAM algorithm.

3.3 Challenges

Executing tasks described in the pseudo-human language imposes some requirements. The robot shall be able to identify and move towards the given waypoints. To achieve that, an appropriate SLAM system has to be used as a base, which must inevitably accommodate entities that can represent waypoints. Thus, the proposed solution is limited on using only SLAM systems that use landmarks to represent whole world objects.

The next step is to link these landmarks to the waypoints given in the directions. A tagger system has to be designed that can recognise objects in the world and identify them in the way a human would do e.g. “door”, “chair” etc.

Changing a navigator so that its main goal is to drive the robot towards a target inflicts the side effect of reducing the quality of the resulting map. Designing the navigator so that the whole procedure yields the best possible map along the followed path towards the target is a main concern.

Chapter 4

Analysis and Design

4.1 Problem Statement

The following paragraph describes the problem confronted by this thesis project and outlines the main issues that the solution copes with.

4.1.1 Goal

The goal of the system to be implemented is to provide the ability of moving towards an unknown predefined target, to a mobile robot. The target should be described in a pseudo-human language. The basic requirements of the system is that the mobile robot has the appropriate sensing equipment that allows it to create and identify landmarks in its environment. For instance, a laser scanner could be used to create the landmarks and a camera could be used for visual identification.

The robot should finally be able to pass the test described below. It should be placed in an unknown indoor environment (office) given the instructions "Go to the third office on your right"

Notice that the goal is not described in a global coordinate system and that the robot has no map in its disposal.

4.1.2 Key Features

In order to achieve the described goal, a robot should possess two key features: Self positioning in an unknown environment and Interpretation of the pseudo-human language in the context of the landmarks observable in the environment. Here, it is important to clarify that the language processing part of the pseudo-human language analysis is out of the scope of this project. Instead, the target is to design a system with the ability to comprehend its environment.

SLAM

Firstly, it should be able to move in an unknown environment, while incrementally building a map. Thus, a SLAM (see Chapter 2.1) variant is used as a base system to provide the crucial ability of the environment perception. However, while the classic SLAM approach forces the robot to explore the world, this project aims on driving the robot towards a target. Consequently, information retrieved from the map is used by the Navigator to determine the robot's path.

Object Map

Secondly, the robot should be able to "understand" and execute the pseudo-human language.

In order to achieve that, the robot should have the ability of labeling the objects of the world that have been stored in the map, with a tag from a vocabulary. This vocabulary is defined by the pseudo-human language, and it consists of a set of crucial world object categories such as door, desk, chair, etc. It should be able to identify and tag the objects of the world in order to create a map of objects and use it to execute the navigation commands.

As one could easily understand, the SLAM and the Navigator systems are tightly connected. Thus, the right choice of a SLAM and Navigator system is of great importance in order to implement the desired system.

4.1.3 Implementation Overview

As mentioned above, a great variety of SLAM implementations exists, each one with its unique advantages and disadvantages. However, there is one that best fits the purposes of this thesis project. scanSLAM is a SLAM variation formulated by Tim Bailey et al. [3, 2, 1] This method uses a laser sensor as the main environment perception instrument. It extracts segments of the laser readings and treats them as "Landmarks" positioned in the map. Therefore, a map of Landmarks is maintained during the SLAM process. Consequently, one can easily see how convenient this is for a new system that tries to link the objects of the environment with a tag.

In this thesis project, the scanSLAM is used as a base system. Attached to that, a vision system (e.g the KINECT sensor) and an offline trained pattern recognition system, could be used to label the Landmarks using a tag from a predefined vocabulary.

Having these Landmarks available, a Navigator designed by B. Tovar et al. [12] is augmented in order to design optimal paths that lead to the desired target.

The following analysis and design is organised in two distinct phases. Firstly (Analysis), an abstract description of the system's components shapes the big picture of the proposed solution. Alongside the first phase, a bunch of alternative techniques are proposed aiming on improving the overall result. Phase two (Design) clarifies how each component is actually implemented by introducing all the mathematical background and the required details.

4.2 Analysis

The algorithm is used to control a mobile robot equipped with a laser distance sensor as the primary perception instrument. Any computation conducted is done in discrete time steps. At each time step, the robot moves, takes a scan, updates the map and computes a new move direction.

As soon as a new laser scan is available, it is segmented into clusters. Each cluster is tested to determine whether it is good enough to form a Landmark. The Landmarks resulting from the current scan, form the set of current observations and they are checked one by one. If the Landmark was never seen before, it is a new observation and thus, a new object in the "map" is created. However, if it is a known Landmark (revisited object), the algorithm determines which point in the "map" it corresponds to (Data Association) and computes the new observation's pose so that the EKF takes it from this point on.

4.2.1 scanSLAM Components

The key components of the scanSLAM base system illustrated in Figure 4.1 are explained in an abstract sense below. The exact detailed analysis following, clarifies the actual implementation of those compontents for the purposes of this project.

Segmentation

A laser scan consists of a set of points called the *PointCloud*. This PointCloud represents the world in front of the robot. Each point signifies space not accessible by the robot. One object of the real world (e.g. a wooden box) is represented by more than one points. The segmentation procedure tries to find groups of points that, may or may not represent real world objects, are "easily" recognised throughout scans.

Break into Clusters

The simplest way to extract clusters from a PointCloud is to choose successive points that are close enough to each other. Thus, thresholding the distance between successive points yields a set of clusters within a Pointcloud. If the points are stored in Polar Coordinates (i.e. Range and Angle) then a simple thresholding on Range and Angle will provide the same result.

Choose Clusters to form candidate Landmarks

Not all the clusters are appropriate to form Landmarks. A Landmark has to be unique and easily identified. In order to get a measure of how informative a Cluster is, a metric called the Object Saliency Score (OSS) [2] has been used. It shows how unique a Cluster is. For example a wall, which is represented by a straight line, has a smaller OSS than a box, which is represented by a corner. The OSS is measured as follows: Compute the Cluster's covariance matrix $\mathbf{R}$ and the $\mathbf{OSS} = \frac{1}{trace(\mathbf{R})}$. It is worth to note that, the more regular the points are, the more $\mathbf{R}$ resembles the identity matrix $\mathbf{I}$. On the other

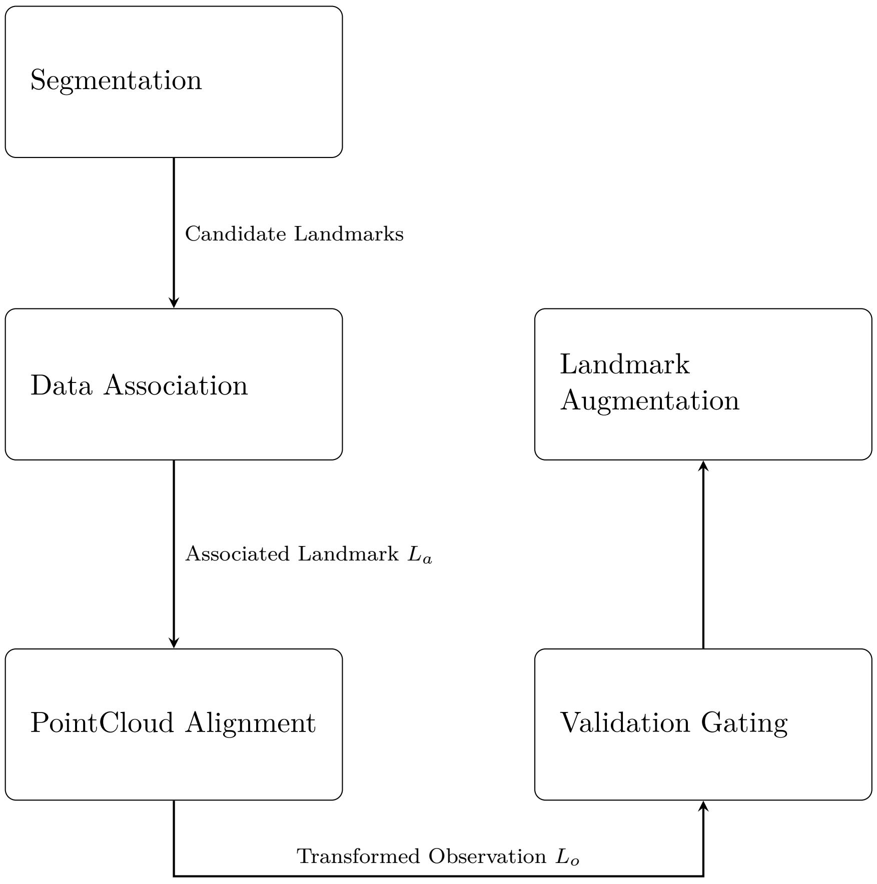

Figure 4.1: scanSLAM Components

hand, the more random the points are, the the more **R** differs from **I**. Thus, the OSS acts as a measure of irregularity. When a Cluster passes the OSS test, a local coordinate frame that represents its pose is assigned on it. This local frame is located at the centroid of the PointCloud of the Cluster and it is rotated by the current robot angle (θ_v degrees).

Data Association

Once a Landmark is observed, the robot must determine whether it has seen it before, and which one from the list of the known Landmarks it is. To achieve that, the candidate Landmark is compared with each of the known (previously visited) Landmarks forming one testing pair $\{\mathbf{o}, \mathbf{x}_i\}$ at a time, where $\mathbf{x}_i$ is the pose of Landmark i being tested, and $\mathbf{o}$ is the pose of the current observation being associated. In order to decide that the two Landmarks of the pair are identical (i.e. they are two instances of the same Landmark), the pair shall pass the innovation gate. When the Data Association procedure is finished, a testing pair wins, and associates the observation's Landmark L_o with one of the candidate Landmarks, say L_a.

Innovation Gate

The primary goal of data association is to determine whether the two Landmarks represent the same space in the world. The easiest way to do that is to compute the distance between the two Landmarks poses, defined as the error $\mathbf{e} = \|\mathbf{x}_i - \mathbf{o}\|$ However, if two different objects of the real world are positioned close to each other, the error $\mathbf{e}$ metric gives rise to ambiguous associations. To avoid that, the uncertainty of the pose of the Landmark has to be taken into account. The more trustworthy the information about the Landmark's pose, the more accurate the error $\mathbf{e}$ calculation is. Thus, normalising the error by the pose covariance norm, which is a measure of the certainty of the pose of the Landmark, a better metric is obtained. This metric is called Normalised Innovation Squared (NIS) and it is defined as follows: $\mathbf{NIS} = \frac{\mathbf{e}}{\|P_{ii}\|}$ where P_{ii} is the $\{i, i\}$ 3x3 sub-matrix of the EKF covariance matrix P, which represents the covariance of the i^{th} Landmark's pose.

PointCloud Alignment

Although the two Landmarks L_o and L_a may match, they are usually slightly displaced in relation to each other. Computing the affine transform (i.e. the rigid body translation and rotation) between the two Landmarks, provides two facilities. Firstly, a validation gate shall be passed after aligning the two PointClouds to ensure the matching is acceptable (Validation Gating). Secondly, the alignment is essential in order to formulate the observation in the way demanded by the EKF to proceed.

Find Affine Transform

The Affine Transform between the observation and the candidate Landmark PointClouds, signifies how the candidate Landmark's PointCloud should be translated and rotated in order to optimally match the observation's PointCloud. Videlicet, to minimise the Mean Square Error (MSE) between them. A well known, efficient enough algorithm that works on PointClouds is the Iterative Closest Point (ICP) algorithm. The ICP algorithm associates each point in the source PointCloud with its closest point in the target PointCloud and finds the translation and rotation needed to minimise the MSE between the associated points. Then, the source PointCloud is transformed according to the result yielded. After a user defined number of iterations, the algorithm provides its output, the final transform. The new observation pose needed by the EKF is provided by transforming the pose (i.e. the local coordinate frame) of L_a according to the computed affine transform.

Cope with Rectilinear Objects

The Data Association procedure is not a simple task. Especially, in the cases where the robot's pose estimation is extremely uncertain. As a result, many issues tend to rise, particularly in large scale environments, always according to the SLAM application. One of those special issues is constituted by the rectilinear objects commonly found in an office environment. When visiting objects whose opposite sides have similar shapes, such as boxes, walls, etc. from different angles, there is the potential to inadvertently align the opposing surfaces. A solution for this problem has been devised by T. Bailey et al. described in [1], which involves the recording of the angle order of scan points in the robot's coordinate frame (the coordinate frame from which the scan was taken).

Validation Gating

Before accepting L_o as a valid observation, it has to pass through a much stricter test than the primary data association procedure, called the Validation Gate. Before applying the validation gate, L_a is transformed using the computed affine transform forming $L_{a'}$. Thus, the PointClouds of $L_{a'}$ and L_o are aligned. The Validation Gate consists of two parts: Firstly, a shape matching gate and secondly, a threshold on the Mahalanobis Distance between the pose of L_o and $L_{a'}$.

Shape Matching

Passing the Innovation Gate ensures that the Landmarks in a testing pair imprint the same space of the real world. The next step is to check whether they represent the same object. Consequently, the shapes emerging from the PointClouds should be checked for dissimilarities. It is crucial to emphasise that two Landmarks that represent the same object of the real world are not necessarily identical, but they may have two completely different PointClouds. This happens for a bunch of reasons such as, they might represent the object as seen by two different angles, they might illustrate different portions of it, the laser readings do not measure the same exact point each time. As a result, it is impossible to compare two Landmarks point by point, but a different method is used instead. The shape matching gate links the two PointClouds point by point. Next, it calculates the percentage of the points that are close enough, i.e. the percentage of the distances that do not exceed a threshold. Thresholding this percentage yields whether the shape match criterion is met or not.

Mahalanobis Distance

The Mahalanobis Distance (M.D.) is used to provide a measure between the two Landmarks treated with a stochastic method. If the information about these two poses is inaccurate, then the M.D. tends to increase, favouring the certain (good) measurements. It is the norm between two points weighted by the covariance between them: $\mathbf{MD^2} = (\mathbf{o} - \mathbf{a'})^T S^{-1} (\mathbf{o} - \mathbf{a'})$ where $\mathbf{o}$ and $\mathbf{a'}$ are the pose's of the observation Landmark and the aligned associated Landmark respectively. Noticing that the observation is obtained from the robot's pose, the covariance involved in the computation described above is the covariance between the robot and landmark L_a. This is stored in the EKF 3x3 covariance sub-matrix {1,a}, which means that $S \equiv P_{1,a}$.

It is crucial to note that, if L_o fails the Validation Gate, then L_a should be unchanged throughout all the structures of the algorithm. Neither its pose, nor its PointCloud should be changed, but they should remain as they wore before the observation. However, if the validation is successful, then the new observation for the purposes of the EKF is given by the transformed associated Landmark's pose ($\mathbf{a'}$)

Landmark Fusion

When a place of the world is revisited, hopefully a testing pair representing it passes the validation gate. In this case, two instances (i.e. two Landmarks) of the same object are available. Thus, there exist the demand of fusing these structures, and maintain only one containing all the useful information. The valid testing pair consists of the observation and the associated Landmark L_o and L_a A Landmark, in the classic scanSLAM version discussed here, holds two crucial pieces of information: a PointCloud and a pose. The EKF uses the PointCloud of L_o to produce the observed pose $\mathbf{a'}$. Then, $\mathbf{a'}$ is used by the EKF in order to consistently fix and maintain the pose of L_a. Consequently, what remains to the designer is to devise a way to maintain the best possible PointCloud for L_a, in order to achieve the best results for future re-observations.

PointCloud Fusion

After alignment, the PointCloud fusion is straightforward. The two PointClouds are matched together (see Figure 4.2), and the easiest way to fuse them is to append the PointCloud of L_o on the PointCloud of L_a. The only problem here is that the resulting PointCloud keeps increasing after each observation. A way to solve this, is DownSampling.

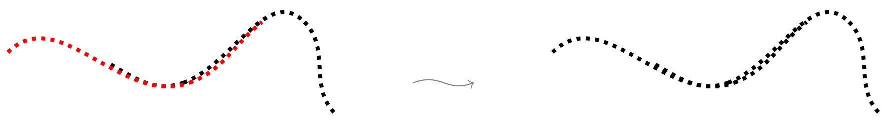
Figure 4.2: PointCloud Fusion

DownSampling

In order to confine the size of the PointCloud in some desired limits, some of its points have to be discarded. The best way to apply DownSampling is not profound. Discarding points from regions with high density, choosing “noisy” - low quality points, or just throwing away every second point are some of the candidate design decisions that are taken for reasons of computational efficiency and performance.

4.2.2 The Simple scanSLAM Algorithm

Using the scanSLAM algorithm in its simplest form, a robot creates a consistent map of its environment. T. Bailey et al. have proposed a great variety of methods to improve its performance throughout their three papers [1, 2, 3]. For the scope of this Thesis project, a very basic implementation is enough in order to create an “acceptable” map. The real value of scanSLAM to this project is the Landmark notion which allows for treating real world objects as entities.

Algorithm 1: The Simple scanSLAM Algorithm

```
repeat
    Initialise Structures
    scan ← Read_Laser()
    cand_land ← Get_Candidate_Landmarks(scan)
    for cand in cand_land do
        L ← Associate(cand, All_Landmarks)
        if L then
            L_alinged ← Align(L, cand)
            if Validate(L_alinged, cand) then
                Append(observations, L_aligned)
        else
            map ← EKF_Add_Landmark(cand)
    map ← EKF_update(control, observations)
    control ← Navigate(map)
    Robot_Move(control)
until Forever
```

The tests supposed to be done, such as the Validation Gate, Shape Matching, etc. are embedded in the pseudo-functions for simplicity. Moreover, the indispensable navigator produces the control commands in a way that maximises the known map while reducing its uncertainty (see Chapter 2.3).

4.2.3 From scanSLAM towards a predefined target

A pseudo language has to be designed, in the framework of which the target of the robot would be described. Having this target in hand, a high level goal setting system (a path planner), will be responsible to produce a sequence of goals to be achieved by the robot. For example, if the target is the "third office on your right" then the three goals would be "go to the nearest door on your right", three times. To achieve that, the meaning of the word "door" has to be familiar to the robot. Moreover, it should be able to extract spacial relationships using the map in order to understand what a right door is, as well as having the ability of going there.

In the scanSLAM framework, two components are essential to implement this kind of behaviour. Firstly, Landmark tagging according to their signification would provide the understanding of the words in the pseudo-language. This can be done using a pattern recognition system, operating on visual data. Secondly, appropriate augmentation of the navigator should provide it the ability to wisely utilise those tags and produce move directions to drive the robot towards the next goal each time.

4.2.4 pseudo-human Language Framework

The directions given to the robot should be described in a language that should feel human while being understood by the robot. The Language designed here aims to be the link between the human language and the machine code. Future extensions of the system could use a pre-processing system to interpret directions given in real language to this pseudo-human form. Special effort has been put in order to develop a general outline, so that it can be used for diverse systems such as drones, submarines etc.

This language is constructed using two components: "Targets" (T) and "Orders" (O) which combined together can describe a direction order. A Target is a set containing a Landmark, the target's achievement requirements and a set of restrictions. For instance, in order to regard a chair as visited, the robot should get 0.5m apart whereas to visit a room's corner, it suffices to go as far as 1m away. Moreover, the robot may be allowed to pass through an open door, or even open the door itself. The Order signifies the actions that should be taken by the robot when it reaches a Target. Each order is stated in an abstract way, e.g. "go through the door" and it is accompanied with a routine that implements the order.

Combining Targets and Orders makes it possible to express directions in a way similar to how humans do, connecting characteristic points of the route with the appropriate actions. Formulating the language in the simplest possible way, a sentence is of the form described by the FSM of Figure 4.3 The robot starts from its current position, and executes one or a sequence of targets followed by one or a sequence of orders. This

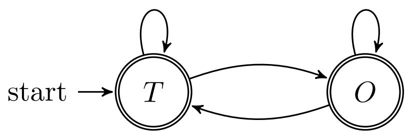

Figure 4.3: The Language Grammar

pattern may be continued infinitely and the robot is allowed to finish either on a Target or on a Order.

Some examples of valid patterns created by the FSM are shown below:

$$T - O - T - ... - T - O - T$$
$$T - O - ... - O - T$$
$$T - O - ... - O$$

Noting the targets with bold, the structure described above can be used to demonstrate how the simple example of ”Go to the second door on your right” could be expressed:

$$\textbf{Here} - \textbf{rDoor} - \textbf{rDoor} - go\_through$$

However, this language can describe more complex directions. For example, a drone could be programmed to perform acrobatics:

$$\textbf{Here} - go\_to\_direction(D) - \textbf{Window} - go\_trough - reverse\_loop - go\_through$$

It is crucial to emphasise that the Language only describes the sequence of targets that have to be achieved, not the way of doing that. It can be thought as explicitly fixing some of the points of the path that has to be designed. There is still the need to design the trajectory between consecutive targets, a process carried out by the navigator. Several issues arising in real applications such as inability to recognise one of the Targets, the need of “Loop Closing” (see Chapter 2.3.3), etc. must be confronted by the navigator, not the target setting language. Moreover, the language outline described here is extremely versatile so that it provides the ability to the designer to fit a custom made language to each application.

In order to design a custom language for a system, it suffices to define the Target vocabulary and the feasible Orders, an example of which is illustrated in Figure 4.4. The vocabulary is strictly connected with the Tagger subsystem, since the Targets have to be recognisable by the system. Once the list has been constructed, the several attributes of each Target should be described. For instance, when a Target is considered to be achieved, what are the available actions of the robot on that target, or anything relevant to the specific application. The Orders list emanates from the robot’s features and capabilities.

Each order is accompanied with a stand alone routine, completely independent from the navigator, that has to be executed as soon as the target is achieved. For example, once the robot has arrived next to a door, thus a Door Target has been achieved, an

Targets		Orders
Identifiers	Tags	
Right	Here	knock
Left	Corner	ask_entrance_permission
Green	Door	enter_room
Big	Box	go_straight
Auckward	Stairs	make_about_turn
	Desk	go_trough
		reverse_loop

Figure 4.4: Language Design

order could be “go_through”. In this case, the appropriate routine is responsible to do the required actions (open the door if it is closed, align the robot in front of the door, move and enter the room) to execute the order. Once the routine is finished the navigator takes charge again.

Taking the intelligence of the system to the next level, the custom language could provide the ability to construct more complex sentences. Consequently, well known techniques from the fields of Information Technology and Programming Languages can be exploited to construct more complex orders and more sophisticated behaviour.

From the robot’s point of view, two crucial tasks have to be executed. Firstly, it should be able to identify each observed Landmark in the map and assign a Tag on it. Secondly, it has to link the desired Target to an existing Landmark.

4.2.5 Tagger

The tagger system is exclusively responsible for characterising the Landmarks by assigning a Tag (see Figure 4.4) on each one. Tagging the Landmarks helps the robot to make sense of the world. The difficulty level of this task could vary from easy to extremely complex, always depending on the language vocabulary.

A pattern recognition system may characterise each Landmark assigning a Tag from the vocabulary, as well as provide vital information on how confident it is about this identification. A visual pattern recognition system could prove to be very useful for this purpose. However, combined sensor information could be utilised for more difficult tasks.

It is extremely important for the whole system to work properly that the robot can satisfactorily comprehend its environment. Consequently, great effort should be put in the design of a good tagger. Not only great variety of techniques has been proposed for 3D objects identification [7], but there exist SLAM algorithms that are based on Landmark characterisation [12].

4.2.6 Target Scouter

The Target scouter is responsible to decide which of the known Landmarks is the desired Target to chase.

To achieve its purpose, this system may employ a variety of techniques. For instance, spatial relationships information could be extracted from the map in order to decide whether a door is either on the right or on the left. Moreover, coupling the Tagger with the Scouter could provide vital information to achieve difficult tasks such as identifying a door as "auckward". Finally, any other available sensor could be used to uniquely characterise a Landmark. For instance, the door of the copy-machine room could be recognised by elevated noise levels.

Obviously, the scouter's complexity extremely depends on the Language's vocabulary. The feasibility of this task is not evident as the vocabulary grows and this shall be of main concern when designing a new Language for a given project. Further analysis must be conducted in this direction, to prove that any combination of {Identifier, Tag} can be correctly recognised and linked to the correct Target by the scouter.

4.3 Detailed Design

For the purposes of this thesis project, a simple system is designed to present an outline of a possible solution. This system will be referred as *Landmark Characterisation scanSLAM (LCscanSLAM).* In this chapter, all the subsystems of "LCscanSLAM" are designed and described in detail.

4.3.1 Discussion

The high level purpose of this system is to provide the ability to a mobile robot to follow direction orders described in a pseudo-human language. In order to achieve that, to begin with, the robot should be able to navigate itself in an unknown environment. This enforces the use of a SLAM system as a base, which as described before (see Chapter 2.1) is composed of three subsystems: the Observation producing system, the Navigator, and the Kernel. Traditionally, the Navigators aim to drive the robot towards the unvisited areas of the map. This project aims to design a Navigator that would drive the robot towards the desired target.

The following design utilises scanSLAM (see Chapter 2.1.4) as a base system because of the very useful way it defines the "Landmark" entity and the ease of its implementation. Fortunately, it is not the only system that the new method is applicable on. More complex systems such as this used by B. Tovar et al. [12] could be used instead. The only premise imposed is that the system manipulates an entity similar to the "Landmark" so that the system is a able to tag it as one of the "targets" of the pseudo-human Language as described in the following chapters.

Like any SLAM solution, this system is composed by three parts. The system responsible to produce the observations, the system that maintains the map, and the system that decides where should the robot go at the next step. The first one is a scanSLAM

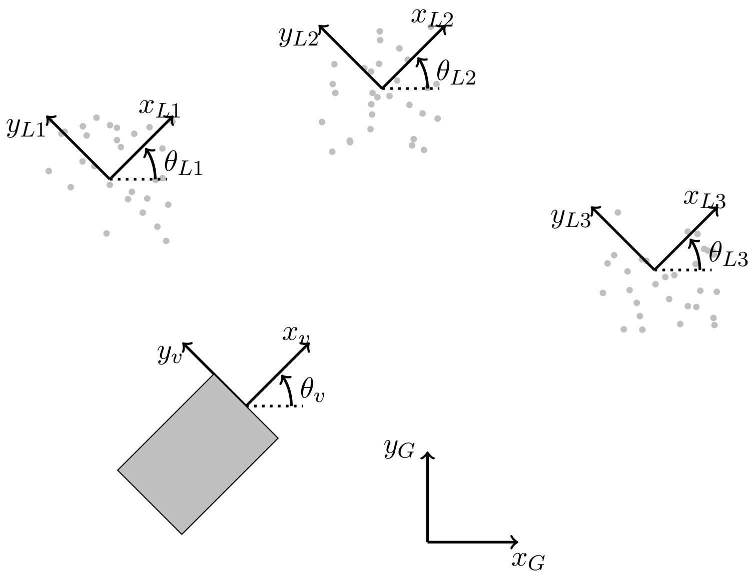

Figure 4.5: Map and Landmark Pose Frames

variation, the second one is an Extended Kalman Filter Kernel and the third one is a cooperation of the pseudo-language, the tagger and the navigator.

All subsystems of "LCscanSLAM" are designed in the most basic version possible.

4.3.2 scanSLAM

A basic SLAM version is implemented based on the methods described by T. Baily et al.'s scanSLAM [1, 2]. The crucial aspect that makes scanSLAM suitable for this project is the use of Landmarks to form the SLAM map. Thus, any divergence from the classic scanSLAM is allowed as soon as it does not interfere with the Landmark entity.

The robot is a two wheel vehicle that moves in the environment and takes scan measurements. As illustrated in Figure 4.5, a local frame attached on the robot signifies its pose $\mathbf{x_v} \triangleq [x_v, y_v, \theta_v]^T$ and localises the robot in the world. A segmentation procedure yields the Landmarks. The pose of the vehicle at the time when the scan is taken provides an indication on the Landmark's poses. Specifically, the frame of each Landmark is attached on the centroid of its cluster (in global coordinates) rotated by θ_v degrees.

The plan described in Chapter 2.1.4 is followed in order to implement a base SLAM system for this project. Around the basic scanSLAM, the Landmark Characterisation pattern recognition system (the Tagger), works to provide the augmented Navigator with the necessary information. What follows is a clarification of the applied techniques.

Algorithm 2: The LCscanSLAM Algorithm

```
Input: Directions described in pseudo-Language
Initialise map
next_direction ← Get_Next(Directions)
repeat
    scan ← Read_Laser()
    {cand_land, free_edges} ← Get_Candidate_Landmarks(scan)
    Tag(cand_land)
    for cand in cand_land do
        L = Associate(cand, Get_All_Landmarks(map))
        if L then
            L_alinged ← Align(L, cand)
            if Validate(L_alinged, cand) then
                Append(observations, L_aligned)
        else
            map ← EKF_Add_Landmark(cand)
    map ← EKF_update(control, observations)
    {control, goal_reached} ← Navigate(map, cand_land, free_edges,
    next_direction)
    if goal_reached then
        next_direction ← Get_Next(Directions)
    Robot_Move(control)
until next_direction == None
```

Segmentation

Break into Clusters

A laser scan consists of an array of range measurements paired with their corresponding angles. The range values are the distances of the points from the laser sensor mounted on the front of the robot. The clustering is done simply by thresholding on the distance of successive points. Thus, a cluster is a set of successive points whose distance does not exceed a threshold. The sets of successive points between the clusters are by definition the free edges used by the Navigator in the following sections (Chapter 4.3.4). Consequently, as illustrated in Figure 4.7 after the processing of each laser scan a set of segments and a set of free edges are available.

Form candidate Landmarks

Each segment forms a candidate Landmark that is put through several tests. Before the next processing phase begins, the coordinates of each segment are transformed to cartesian expressed on a local frame attached at the centroid of the cluster as illustrated in Figure 4.8 This procedure is done according to

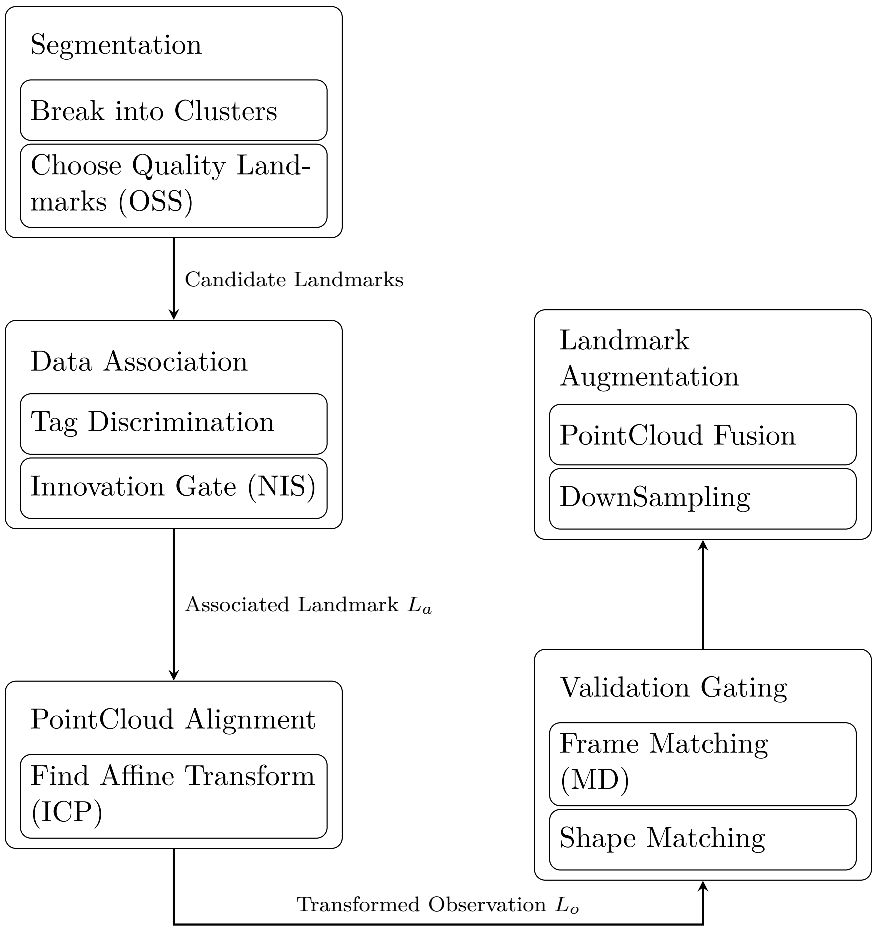

Figure 4.6: scanSLAM Components

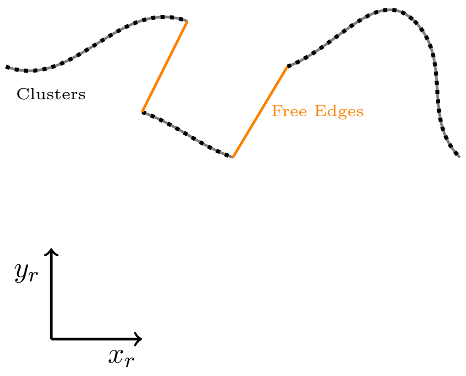

Figure 4.7: Candidate Landmarks and Free Edges

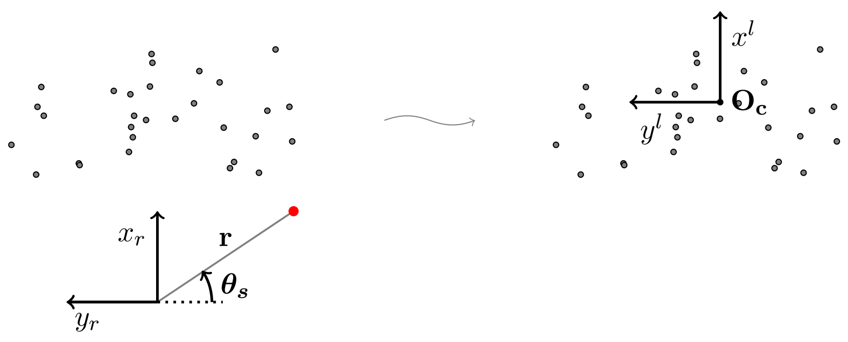

Figure 4.8: Transforming to Cartesian Coordinates

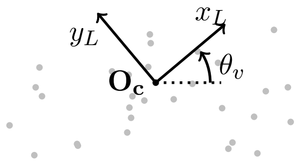

Figure 4.9: Landmark's Pose Frame

Equations 4.1 and 4.2 The points expressed on the local cartesian *frame l* constitute the PointCloud of the candidate Landmark: $PCL \equiv \{(x_i{}^l, y_i{}^l)\}$.

$$\begin{pmatrix} x^l \\ y^l \end{pmatrix} = \begin{pmatrix} rS_{\theta s} - x_c \\ -rC_{\theta s} - y_c \end{pmatrix} \tag{4.1}$$

$$\mathbf{O_c} \equiv \begin{pmatrix} x_c \\ y_c \end{pmatrix} \triangleq \frac{1}{n}\sum_{i=1}^{n} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \tag{4.2}$$

Then, the frame that represents the pose of the Landmark is attached at its centroid $\mathbf{O_c}$ with rotation equal to the robot's angle θ_v at the time of the scan as illustrated in Figure 4.9. Thus, the landmark's initial pose is given by:

$$\mathbf{x_L} \equiv \begin{pmatrix} x_L \\ y_L \\ \theta_L \end{pmatrix} \triangleq \begin{pmatrix} x_v + x_c \\ y_v + y_c \\ \theta_v \end{pmatrix} \tag{4.3}$$

Choose quality Landmarks

Next, the candidate Landmarks are tested to find out how unique they are, i.e. compute their Object Saliency Score (OSS).

$$\mathbf{OSS} = \frac{1}{trace(\mathbf{R})} \tag{4.4}$$

Where:

$$\mathbf{R} = cov(PCL)$$

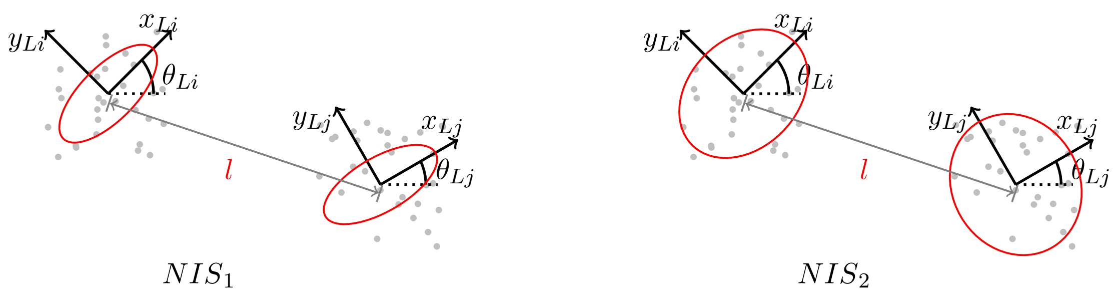

Figure 4.10: Normalised Innovation Squared

The Landmarks with OSS lower than a threshold are rejected.

Data Association

Each of the candidate Landmarks is checked over the Landmarks in the map to find whether it is a new one or if it has been observed in the past. Symbolising the Landmark of the map as L_i and the shortly observed Landmark from the candidate set as L_o, a testing pair is formed $\{L_o, L_i\}$ or by their poses $\{\mathbf{x_o}, \mathbf{x_i}\}$

Tag Discrimination

Assuming that the Landmark in the map is always correctly tagged, a preliminary test rejects the association if the two tags disagree. This is a very strict restriction, but it ensures that the robot moves towards a particular target, since the tags of the stored Landmarks are not allowed to change. A different approach could allow to correct a Landmak's tag according to the Tagger's score if an association is otherwise successful. However, this would require a more sophisticated path planner.

Innovation Gate

The Normalised Innovation Squared (NIS) measures how close the two frames are taking account of the uncertainty as well, as illustrated in Figure 4.10 where $NIS_1 < NIS_2$

$$\mathbf{NIS} = \frac{\|\mathbf{x_i} - \mathbf{x_o}\|}{\|\mathbf{P_{Li}}\|} \tag{4.5}$$

where $\mathbf{P_{Li}}$ is the 3x3 sub-matrix of the EKF map covariance matrix P, which represents the covariance of the i^{th} Landmark's pose.

If there is no Landmark in the map with NIS lower than a threshold, then L_o is considered as a new observation and a new Landmark is initialised in the map. In the opposite case, the Landmark L_i with the lowest NIS is associated with L_o, and from now on it will be notated as L_a

PointCloud Alignment

Find Affine Transform

After the association, the new observation has to be formulated in the EKF

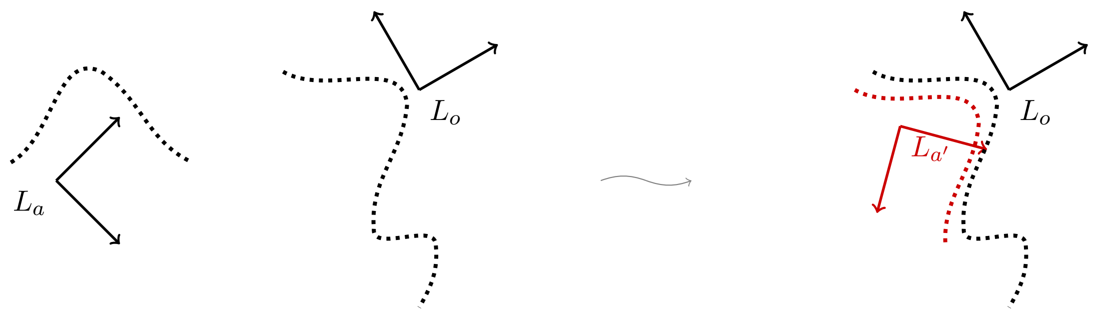

Figure 4.11: New Observation for the EKF

framework. Specifically, using the new observation, the augmentation of $\mathbf{x_a}$ (the pose of L_a), has to be determined. Notice that $\mathbf{x_a}$ comes from the current prediction of the map and so it signifies where the robot thinks that the Landmark is. Thus, computing the difference between the perception of the world and the new observation provides the necessary new information to the EKF to produce a better prediction.

The Iterative Closest Point (ICP) algorithm provides the rotation and the translation needed to be applied on the PointCloud of L_a in order to best match the PointCloud of L_o as illustrated in Figure 4.11 Applying it on the pose of L_a yields the new observation. Equation 4.6 describes how the pose frame is transformed whereas Equation 4.7 describes how the PointCloud is transformed.

$$\mathbf{x_{a'}} = \mathbf{x_a} + \mathbf{x_{off}} \tag{4.6}$$

$$\begin{pmatrix} x_i \\ y_i \end{pmatrix}^{a'} = \begin{pmatrix} C_{\theta_{off}} & -S_{\theta_{off}} \\ S_{\theta_{off}} & C_{\theta_{off}} \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix}^{a} + \begin{pmatrix} x_{off} \\ y_{off} \end{pmatrix} \tag{4.7}$$

where:

$$\mathbf{x_{off}} \triangleq ICP(\scriptstyle PCL_a, PCL_o \textstyle) = \begin{bmatrix} x_{off} \\ y_{off} \\ \theta_{off} \end{bmatrix}$$

Cope with Rectilinear Objects

This part is not implemented as the robot will have to execute simple orders that will not require it to revisit an object from both sides.

Validation Gating

The validation gate provides the final approval or rejection of the association.

Frame Matching - Mahalanobis Distance

Firstly, the Mahalanobis Distance (MD) is used to incorporate information about the pose uncertainty of the Landmarks.

$$\mathbf{MD^2} = (\mathbf{x_o} - \mathbf{x_{a'}})^T {P_{v,a}}^{-1} (\mathbf{x_o} - \mathbf{x_{a'}}) \tag{4.8}$$

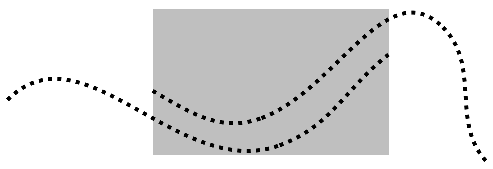

Figure 4.12: Shape Matching

A threshold on MD decides whether the validation is successful or not. If L_o passes the Validation Gate, then the new observation defined by Equation 4.6 is attached to the observations list. When all the Landmarks in a scan are examined, the observations list is used by the EKF to update the map.

Shape Matching

Before applying the shape matching test, a replica of L_a is created and it is transformed using the computed affine transform forming $L_{a'}$. Thus, the PointClouds of $L_{a'}$ and L_o are aligned. This is done in order to reduce the computational effort of transforming $L_{a'}$ back to L_a in case the validation fails. Because the two PontClouds may represent different parts of the same object, the matching is confined on their overlapping region. If it does not contain enough points, the association is rejected. For the points in the overlapping region, the nearest neighbours are paired using the k - Nearest Neighbours (KNN) algorithm. The total error e_t is defined as the sum of the distances between the neighbours that do not exceed a threshold. The percentage of the neighbours that are close enough to each other is then computed using Equation 4.10 A threshold on the percentage provides a decision on the shape matching (see Figure 4.12).

$$e_t \triangleq \sum_{i=1}^{N} d_i : d_i < d_{thres} \tag{4.9}$$

$$Perc = \frac{e_t}{N} \tag{4.10}$$

Landmark Fusion

PointCloud Fusion

To enrich the information that describes the Landmark, the PointCloud of the new observation may be incorporated. Thus, the PointCloud of L_o is firstly transformed using the inverse affine transform in order to match the PointCloud of L_a, and then it is appended on it.

$$PCL_a = PCL_a \cup PCL_{o'} \tag{4.11}$$

where $PCL_{o'}$ is obtained by Equation 4.7 using $-\mathbf{x_{off}}$.

$$\mathbf{X} = \begin{bmatrix} x_v \\ y_v \\ \theta_v \\ \cdots \\ x_{L1} \\ y_{L1} \\ \theta_{L1} \\ \vdots \\ x_{LR} \\ y_{LR} \\ \theta_{LR} \end{bmatrix} \triangleq \begin{bmatrix} \mathbf{x_v} \\ \cdots \\ \mathbf{x_L} \end{bmatrix}_{Mx1} \tag{4.12}$$

State Vector: The Map

DownSampling
If the size of the PointCloud increases over a threshold, then a DownSampling procedure is executed. It is done by sorting the points in the PointCloud according to the x dimension, and throwing away every second point.

When all the associations in a scan are done, the poses of observations are produced and stored in an observation list and the update of the structures is finished, it is time to update the map as well to incorporate the new information. The EKF uses the control signal history from the previous update until the current one and the new observations to improve the prediction of the map. Finally, the Navigator uses the new map and the current scan to decide which the next control signal is to be asserted.

4.3.3 EKF Kernel

Formulating the SLAM problem in the EKF framework, the map is defined as the EKF state vector (Equation 4.12). The first three elements of the state vector correspond to the robot's pose (vehicle) whereas the rest of the state vector is composed by triplets that correspond to the poses of the Landmarks of the map. Each triplet defines the position of the local frame of each Landmark via the first two variables (x_{Li}, y_{Li}). The third variable θ_{Li} refers to the rotation of that frame, as shown in Figure 4.13

The motion model (Equation 4.13) describes how the robot would move when a control $(R, \dot{\theta}_v)$ is asserted on it. As illustrated in Figure 4.14 the local frame of the robot is attached on the laser sensor.

The observation model (Equation 4.15) describes how the robot perceives the world. The pose of each one of the observed Landmarks is transformed to the robot's frame $\mathbf{x_v}$ using $\mathbf{h_i}$ to construct the observation vector $\mathbf{h}$

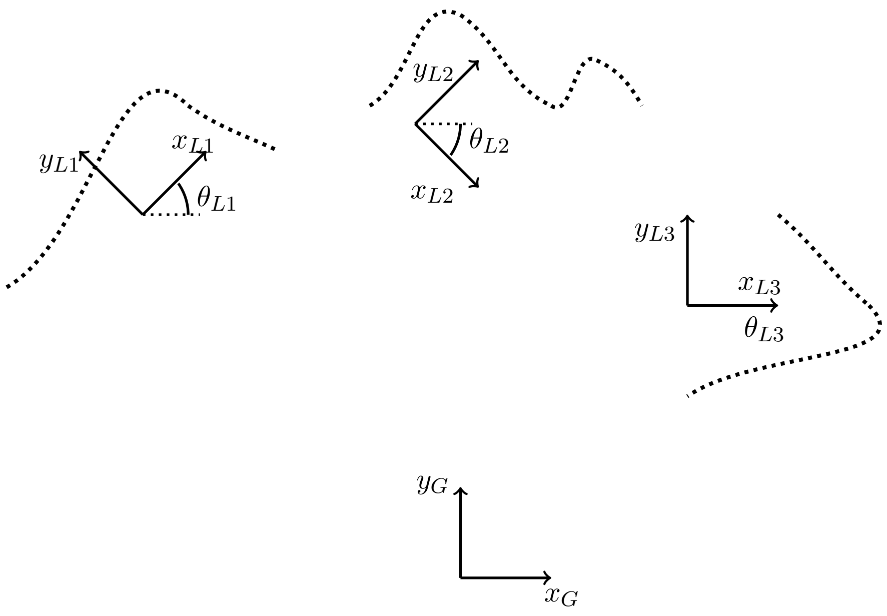

Figure 4.13: Landmarks and Local Frames

$$\mathbf{f}(\mathbf{x_v}, R, \dot{\theta}_v) \triangleq \begin{bmatrix} R(S_{\theta_v}C_{\dot{\theta}_v} + C_{\theta_v}S_{\dot{\theta}_v} - S_{\theta_v}) + x_v \\ R(S_{\theta_v}S_{\dot{\theta}_v} - C_{\theta_v}C_{\dot{\theta}_v} + C_{\theta_v}) + y_v \\ \dot{\theta}_v + \theta_v \end{bmatrix}_{3x1} \tag{4.13}$$

Motion Model

$$\begin{aligned} R &= \frac{l}{2}\frac{V_r + V_l}{V_r - V_l} \\ \dot{\theta}_v &= \frac{V_r - V_l}{l}\Delta t \end{aligned} \tag{4.14}$$

Control

Where:

$$\begin{aligned} V_r &: Right\ wheel\ velocity \\ V_l &: Left\ wheel\ velocity \\ \Delta t &: Time\ period\ of\ V_r,\ V_l\ being\ asserted \end{aligned}$$

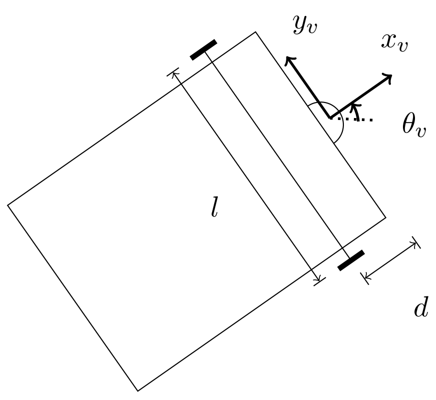

Figure 4.14: Differential Drive Robot

$$\mathbf{h} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \cdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ \mathbf{h_i} \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ \mathbf{h_j} \\ 0 \\ 0 \\ 0 \\ \vdots \end{bmatrix}_{Mx1} \qquad \mathbf{h_i} \triangleq \begin{bmatrix} C_{\theta_v}(x_{Li} - x_v) + S_{\theta_v}(y_{Li} - y_v) \\ -S_{\theta_v}(x_{Li} - x_v) + C_{\theta_v}(y_{Li} - y_v) \\ \theta_{Li} - \theta_v \end{bmatrix} \tag{4.15}$$

Observation Model

$$\mathbf{P} = \begin{bmatrix} \mathbf{P_v} & \mathbf{P_{V,1}} & \\ \mathbf{P_{V,1}} & \mathbf{P_{L1}} & \\ & & \ddots \end{bmatrix}_{MxM} \tag{4.16}$$

$$\mathbf{P_v}|_{k=0} \triangleq \mathbf{I}_{3x3} \tag{4.17}$$

Map Uncertainty Matrix

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q_v} & \mathbf{0}_{3x3R} \\ \mathbf{0}_{3Rx3} & \mathbf{0}_{3Rx3R} \end{bmatrix}_{MxM} \tag{4.18}$$

$$\mathbf{Q_v} \triangleq \begin{bmatrix} {\sigma_x}^2 & & \\ & {\sigma_y}^2 & \\ & & {\sigma_\theta}^2 \end{bmatrix} \tag{4.19}$$

Motion Model Uncertainty Matrix

$$\mathbf{R} = \begin{bmatrix} \mathbf{0}_{3x3} & & & \\ & \mathbf{R_{L1}} & & \\ & & \ddots & \\ & & & \mathbf{R_{LR}} \end{bmatrix}_{MxM} \tag{4.20}$$

$$\mathbf{R_{Li}}|_{k=0} \triangleq \mathbf{R_o} = \begin{bmatrix} {\sigma_{xo}}^2 & & \\ & {\sigma_{yo}}^2 & \\ & & {\sigma_{\theta o}}^2 \end{bmatrix} \tag{4.21}$$

Observation Model Uncertainty Matrix

After defining the observation and motion model, what remains is the computation of the Jacobians as described in Chapter 2.2

$$\mathbf{H} = \begin{array}{cc} & \begin{array}{cccccccc} \phantom{\mathbf{H_{Li}}} & \phantom{\mathbf{0}_{3x3}} & & i & j & \phantom{\mathbf{0}_{3x3}} & & \phantom{\mathbf{0}_{3x3}} \end{array} \\ \begin{array}{c} \\ \\ i \\ j \\ \\ \\ \end{array} & \begin{bmatrix} \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & & \cdots & & & & \mathbf{0}_{3x3} \\ \vdots & \ddots & & \cdots & & & & \mathbf{0}_{3x3} \\ \mathbf{H_{Li}} & \mathbf{0}_{3x3} & \cdots & \mathbf{H_o} & \mathbf{0}_{3x3} & \cdots & & \mathbf{0}_{3x3} \\ \mathbf{H_{Lj}} & \mathbf{0}_{3x3} & \cdots & & \mathbf{H_o} & \mathbf{0}_{3x3} & \cdots & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \mathbf{0}_{3x3} & & \cdots & & & & \mathbf{0}_{3x3} \\ \vdots & \ddots & & \cdots & & & & \mathbf{0}_{3x3} \end{bmatrix}_{MxM} \end{array} \tag{4.22}$$

$$\mathbf{H_{Lk}} \triangleq \begin{bmatrix} -C_{\theta_v} & -S_{\theta_v} & -S_{\theta_v}(x_{Lk} - x_v) + C_{\theta_v}(y_{Lk} - y_v) \\ S_{\theta_v} & -C_{\theta_v} & -C_{\theta_v}(x_{Lk} - x_v) - S_{\theta_v}(y_{Lk} - y_v) \\ 0 & 0 & -1 \end{bmatrix} \tag{4.23}$$

$$\mathbf{H_o} \triangleq \begin{bmatrix} C_{\theta_v} & S_{\theta_v} & 0 \\ -S_{\theta_v} & C_{\theta_v} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.24}$$

$$\mathbf{F} = \begin{bmatrix} \mathbf{F_o} & \mathbf{0}_{3x3R} \\ \mathbf{0}_{3Rx3} & \mathbf{I}_{3Rx3R} \end{bmatrix}_{MxM} \tag{4.25}$$

$$\mathbf{F_o} \triangleq \begin{bmatrix} 1 & 0 & R(C_{\theta_v}C_{\dot{\theta}_v} - S_{\theta_v}S_{\dot{\theta}_v} - C_{\theta_v}) \\ 0 & 1 & R(C_{\theta_v}S_{\dot{\theta}_v} + S_{\theta_v}C_{\dot{\theta}_v} - S_{\theta_v}) \\ 0 & 0 & 1 \end{bmatrix} \tag{4.26}$$

After each scan processing and before the computation of the next direction order, the EKF updates the map's prediction. If a new Landmark has been observed, it is added in the map and the structures defined above have to be updated:

- The state vector $\mathbf{X}$ is extended with the pose of the new landmark $\mathbf{x_L}$ defined by Equation 4.3
- The Map Uncertainty Matrix ($\mathbf{P}$) is extended with a 3Rx3 column to the right and a 3x3R row below. On the bottom right corner, the uncertainty of the new Landmark's pose is initialised to be equal to the current vehicle uncertainty $\mathbf{P_v}$
- The Motion Model Uncertainty Matrix ($\mathbf{Q}$) is extended with a 3(R+1)x3 column to the right and a 3x3(R+1) row below. The vehicle's uncertainty matrix $\mathbf{Q_v}$ is compututed experimentally
- The Observation Model Uncertainty Matrix ($\mathbf{R}$) is extended with a 3Rx3 column to the right and a 3x3R row below. On the bottom right corner, the uncertainty of the new Landmark's observation is initialised to be equal to the laser sensor's uncertainty $\mathbf{R_o}$

To update the map, the Equations 2.8 are executed and the new map $\hat{x}(k+1|k+1)$ along with its uncertainty matrix $P(k+1|k+1)$ are available.

4.3.4 Navigator

As mentioned before, a Navigator consists of two parts, the path planner and the trajectory planner. A sophisticated Navigator tries to design routes that maximise a set of criteria. These routes are defined by a sequence of points - targets. To connect two consecutive targets, the trajectory planner designs the control signal fed to the wheel motors.

Introducing the requirement of moving towards a target is not a simple assignment to the navigator. This thesis project aims to delineate the outlines of a mechanism necessary to provide this utility. Namely, the ability to tag the Landmarks of the map and use them as targets to be reached. Blending this behaviour with the other required tasks of a Navigator has to be examined thoroughly, so that the robot does not necessarily build the best possible map of its environment, but it optimally maps the environment along the route followed towards its destination.

Note that what mentioned above is a completely different task than of what a traditional SLAM algorithm does. The goal now is neither to maximise the visited areas, nor to improve the quality of the map. The main goal now is to go towards a target in an unknown environment. Whether a good map is constructed in the meantime or not, is not of great interest. Undoubtedly, since the map is needed for navigation purposes, it has to be at least as good as necessary to serve its purpose.

In order to demonstrate the new features of the system, a simple navigator is designed.

Language Design

The first step in the process of designing a new system is to define the vocabulary of the pseudo-Language. The system is destinated for an indoor office environment, so the Targets have to be general categories that represent stationary distinctive objects regularly found in an indoor environment. Figure 4.15 illustrates the vocabulary used in this project's simulation. It has been forced to be small but descriptive.

Targets		Orders
Identifiers	Tags	
Right	Here	Turn Left
Left	Corner	Inform
	Door	Stop
	End	

Figure 4.15: LCscanSLAM Language

Tagger and Target Scouter Design

In order to avoid any errors imposed by faulty Landmark identification, the tagging is contacted mannually. Moreover, the Target Scouter is designed in the simplest possible form. The desired Target is associated with the closest Landmark with the proper Tag.

Path Planner Design

The simple path planner designed here assumes that all the Landmarks have been tagged correctly, that the Targets are achieved in the right order and no Target is missed. The essential language is shown in Figure 4.15 Each Target is associated with a set of Orders that have to be executed as soon as the Target is reached.

Algorithm 3: Path Planner Design

```
repeat
    next_Target ← pop(Targets_List)
    move(Trajectory_Planner(next_Target))
    repeat
        Order ← pop(next_Target(Orders_List))
        execute(Order)
    until Orders_List is empty
until Targets_List is empty
```

It is important to notice that the current path planner serves only as a proof of concept. It can not be used in any real world SLAM application because it does not cope even with the minor requirements of such a system. It does neither consider any map quality needs, nor any computational efficiency issues that may rise.

Trajectory Planner

The trajectory planner designed by B. Tovar et al. [12] uses a utility function that works on a sequence of points. The corresponding path planner is responsible to produce a set of candidate paths that the robot can follow starting from its current position given the known map at the time. Having this set of paths available, the utility function evaluates the quality of each path and chooses the best one. Thus, the utility function involved not only works on single points but it can evaluate paths as well, using information provided by the path planner.

On the other hand, the simple path planner used here differs drastically. In fact, what is described above can not be considered as a path planner in the classic sense. Instead, the desired functionality of choosing the next point, and in consequence the final path is hidden in the links between the targets and it is incorporated in the trajectory planner. At each step, the trajectory planner is responsible to choose the position in front of the robot that maximises the utility function (Equation 4.27), and drive the robot to it.

Instead of finding an analytic solution of the utility function, as this task would be too costly and in many cases impossible, the traditional way of avoiding this burden is

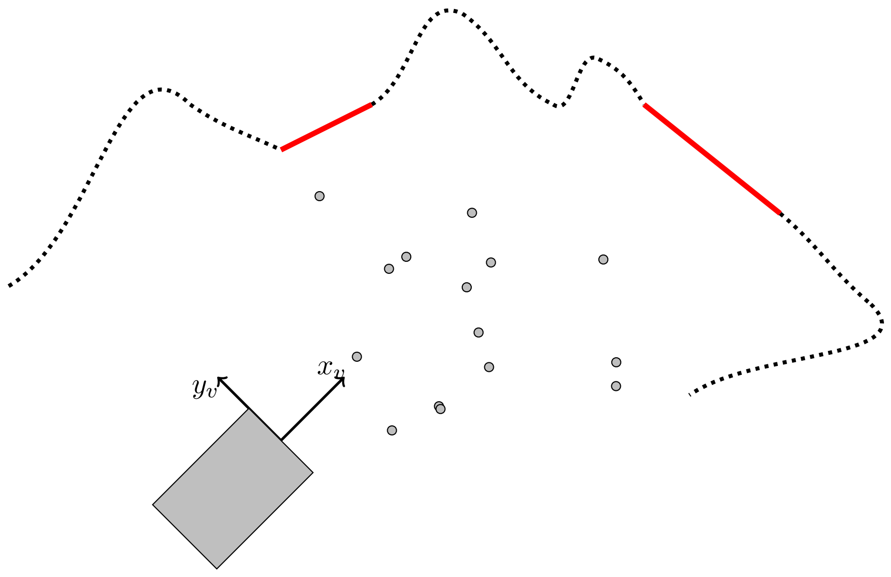

Figure 4.16: Draw Candidate Points

to employ a sampling scheme. Drawing enough points in the desired area and evaluating the utility of each one provides a destination near the optimal one. As illustrated in Figure 4.16, a set of random points laying in front of the robot are drawn at each step. Then, the utility function is evaluated at each point and the robot moves towards the one that yields the highest utility.

$$T = w_t e^{-S_t} + e^{w_e(l_v - S_v)} \frac{e^{-w_u|\theta|}}{1+\sqrt{S}} f_{min}(d) \tag{4.27}$$

where:

$$\begin{aligned} S_t &: Distance\,to\,next\,Target \\ l_v &: Length\,of\,nearest\,free\,edge \\ S_v &: Distance\,to\,nearest\,free\,edge \\ \theta &: Angle\,to\,face\,point \\ S &: Distance\,to\,point \\ f_{min}(.) &: Distance\,to\,obstacle\,penalty\,function \\ d &: Distance\,to\,nearest\,obstacle \end{aligned}$$

Modifying the original utility function to meet this project's requirements yields Equation 4.27 It consists of four terms, each one serving a specific purpose. The significance of each term can be controlled by the weights w_i which are experimentally determined.

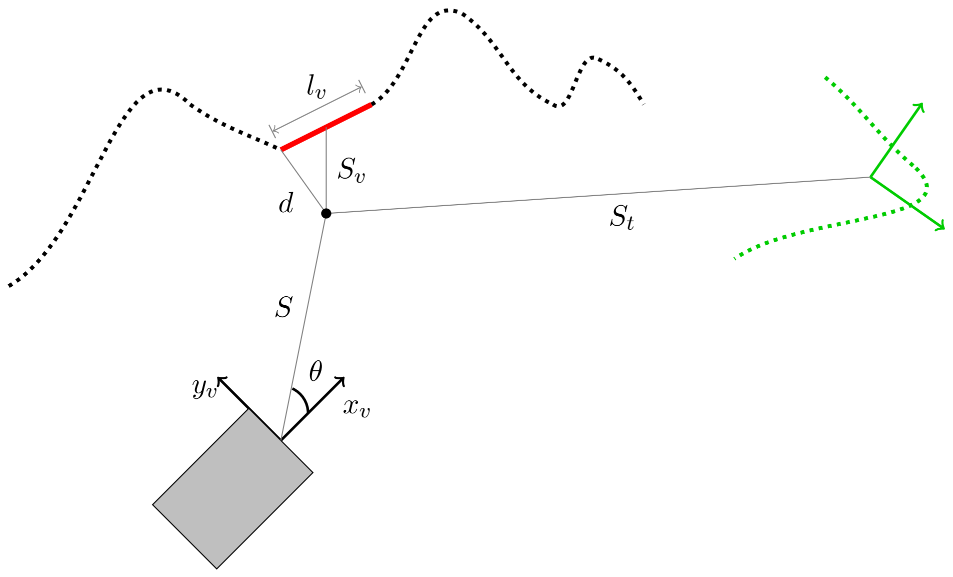

Figure 4.17: Utility Function Components

Go to the Target: e^{-S_t}

The first term of the utility function is the most important one for what this project is concerned. The primary purpose of the system is to drive the robot towards the next target at each step. It is noteworthy that en exponential term is chosen in order to emphasise its effect. If the next Target has not been observed yet, and thus the distance S_t can not be measured, it is manually set to zero. Consequently, the first term is set to 1 and it is identical across all candidate points. Thus, it does not affect the utility function, leading the robot to move as it would if it was running a classic SLAM algorithm.

This is one of the greatest achievements of the proposed method. The solution can be formulated in such a way that the robot can operate as usual in case it has no goal to achieve.

Explore: $e^{l_v - S_v}$

In order to move towards the unvisited parts of the map, the robot should prefer points near the vague areas. To find these places where the perception of the environment is not clear, the "free edge" is defined as the border between regions of explored and unexplored space. Thus preferring points near long free edges drives the robot towards the unknown regions of the map. Maximising the first term of the utility function favours points near free edges ($max\{e^{-S_v}\}$) that are as long as possible ($max\{e^{l_v}\}$).

A proper use of a free edge as an indication of the unexplored regions of the map, would be to store every free edge met in the global map, and retrieve that information in order to design a path. However, this would require to employ

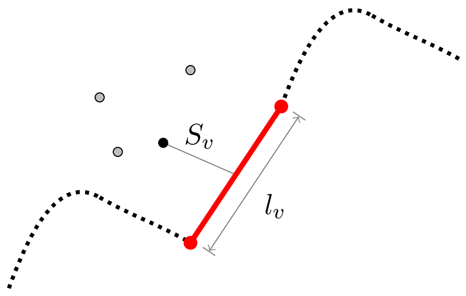

Figure 4.18: Free Edge

special housekeeping techniques of the map that prove to be too complicated in the case of scanSLAM. Although it is possible to implement, this proper treatment is avoided, and only the free edges of the current scan are considered. This is not a big compromise if one notices that the candidate next points to visit all lie in front of the robot. On the other hand, in the case that a better path planner was used, this issue should have been examined with care.

Identifying free edges in laser scan data is not a difficult task. It suffices to notice that when the laser hits a continuous surface, it provides points that tend to be close to each other. Thus, finding the sparse areas in a scan yields the free edges. Luckily, the segmentation procedure demanded by the scanSLAM provides both possible Landmark clusters and free edges between them.

An approximation of its length l_v is given by the euclidean distance between the first and the last point in the PointCloud of the cluster that represents the free edge (see Figure 4.18). Of course, it is important that the points are sorted by angle so that they are consecutive in the real world according to the laser beam scanning direction. The centre of the free edge is given by the centre of the line drawn from the first to the last point. The distance S_v is the length of the line connecting the robot position with the centre of the free edge.

A better model that fits polylines to the laser data is used in [12]. It applies a divide-and-conquer technique combined with a least squares method. This approach has the advantage of removing noisy measurements but it is more computational intensive.

Prefer Short Straight Lines: $\frac{e^{-|\theta|}}{1+\sqrt{S}}$

It has been experimentally proven that rotation boosts the uncertainty of the robot's pose. Thus, the best path to follow from target to target would be the straight line that connects them. It would demand no rotation and the shortest possible distance to be covered. Unfortunately, this is not possible to happen because in many cases the line that connects consecutive targets is not attainable due to obstacles or the geometry of the environment.

This leads to the need of following polyline paths. The second term of the utility

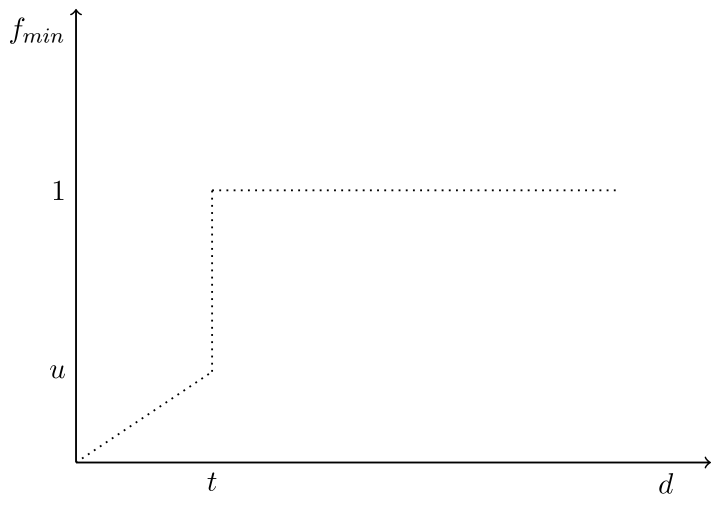

Figure 4.19: Distance to obstacle penalty function

function favours the points that require minimal rotation ($max\{e^{-|\theta|}\}$) and abstain the least possible distance from the current position of the robot ($max\{\frac{1}{1+\sqrt{S}}\}$).

Avoid Obstacles: $f_{min}(d)$
Avoiding the obstacles in the robot's way is not only crucial for collision averting but involves other practical issues as well. Keeping a certain distance away from objects and walls ensures that the sensors used, do not become blind due to short range measurements. On the other hand, the practice of "the more the merrier" does not apply in this occasion. It makes no sense to go very far from the obstacles either. Thus, B. Tovar et al. designed a function ($f_{min}(d)$: Figure 4.19) that penalises points that abstain less than a threshold from an obstacle while being unconcerned for those that lie too far.

The utility function described above serves two objectives. It's form is such that the importance of each term can be configured according to its purpose using the weights w_i. Note that the main objective (moving towards the target, is added to the secondary one (minimise uncertainty) so that their relative importance is controlled through w_t. Focusing on the secondary term though, the multiplicative form proposed by B. Tovar et al. implies that the winning point is good enough in all terms of interest. In other words, if a point meets two out of three requirements, it will yet yield a low score.

At each time step, when the robot has to decide where to go next, the following algorithm produces a set of random points in front of the robot and chooses the best according to the utility function. Finally it returns the control signal for the wheel motors that moves the robot towards the winning point.

Algorithm 4: Trajectory Planner

```
cand_points ← Draw_Random_Points
next_point ← max(Evaluate(cand_points))
return compute_control(next_point)
```

The implementation described here is just one possible solution for a simple application. However, it has to be emphasised that the proposed method is not confined by any assumption made for simplicity purposes. The general framework of the pseudo-language formulation along with a proper utility function design can be fit to any SLAM algorithm that can accommodate the “Target” entity, in order to implement a more sophisticated and practical system.

Chapter 5

Implementation

A simulation has been implemented in PYTHON, aiming on future porting of the system through the *Robot Operating System (ROS)*, on a differential drive robot (named "Sek"[1]) housed at the National Centre for Scientific Research (NCSR) Demokritos. The source code of the simulation is available to the public[2]. This chapter demonstrates the performance of the subsystems of the conducted simulation while indicating any possible weaknesses.

5.1 Overview

The robot has been driven manually along a corridor in NCSR Demokritos collecting data using its laser scanner. The control signals provided and the laser readings have been used in the simulation to produce a map using scanSLAM which has been implemented from scratch according to the analysis and design chapter. The characterisation of the Landmarks according to the designed pseudo-human language has been done manually.

The purpose of the simulation is to prove that at any given time, the navigator takes the right decision in order to move towards the target and follow the given orders.

The resulting system is designed in such a way that can be easily used as a base to implement LCscanSLAM on any mobile robot. It consists of carefully separated, stand alone components that can be fit appropriately for the needs of any other platform.

5.2 Components

Segmentation

Once a new scan is available, it is segmented in a set of clusters. According to the system design, these clusters are chosen to form candidate landmarks based on their Object Saliency Score. Unfortunately, the nature of indoor environments does not favour the formation of unique, distinctive clusters. Oppositely, as seen

[1] `http://roboskel.iit.demokritos.gr/personnel/sek`
[2] `https://github.com/AndLydakis/LCscanSLAM`

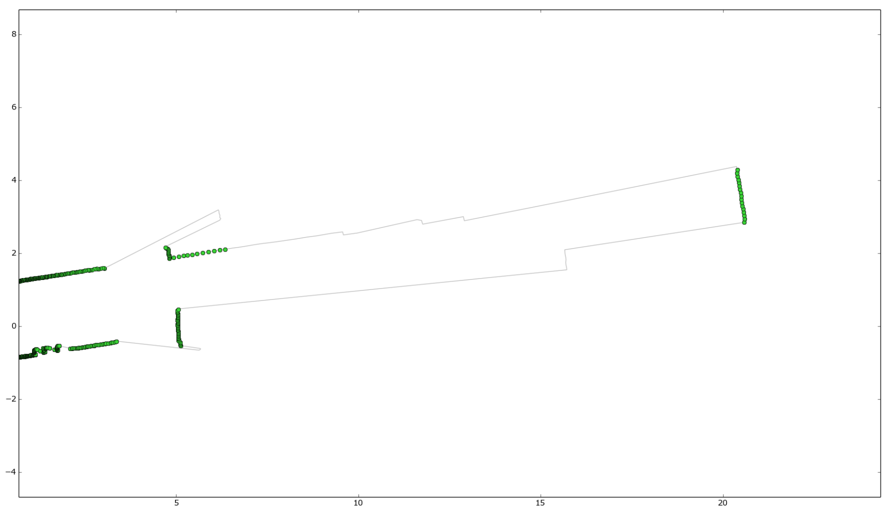

Figure 5.1: Scan segmented in candidate Landmarks

in Figure 5.1 the extracted clusters are mainly straight lines. As a result, the OSS metric tends to be useless. As a matter of fact, discarding clusters because of their low OSS in a corridor could lead in lack of landmarks, leading to complete failure of the algorithm.

What is more interesting here though is the failure of the OSS metric to estimate how unique a cluster is. As illustrated in Figure 5.2, the green corner has smaller OSS than the light blue line and greater OSS than the deep blue wavy cluster. It is noteworthy, that the later actually represents a really distinctive place in the corridor where a set of fire extinguishers are suspended.

Actually, the OSS metric seems to suffer from extreme dependency on the number of points in a cluster. As a result, a cluster that covers a tiny area while containing a big number of points which is actually pure noise, yields a large OSS. While a clear big area cluster with few points, that represents a corner on the wall like the green one, yields a small OSS. A good idea to improve this metric would be to consider the points density of the cluster, i.e. the number of points it contains divided by the area it covers.

However, the computation of the area can be proven to be tricky and computationally intensive. It is not sufficient to estimate its area as the area covered by the rectangle shaped by the boundary points. This estimation would be prone to any rotation of the cluster. The proper way would be to compute the area of the convex hull that surrounds the points.

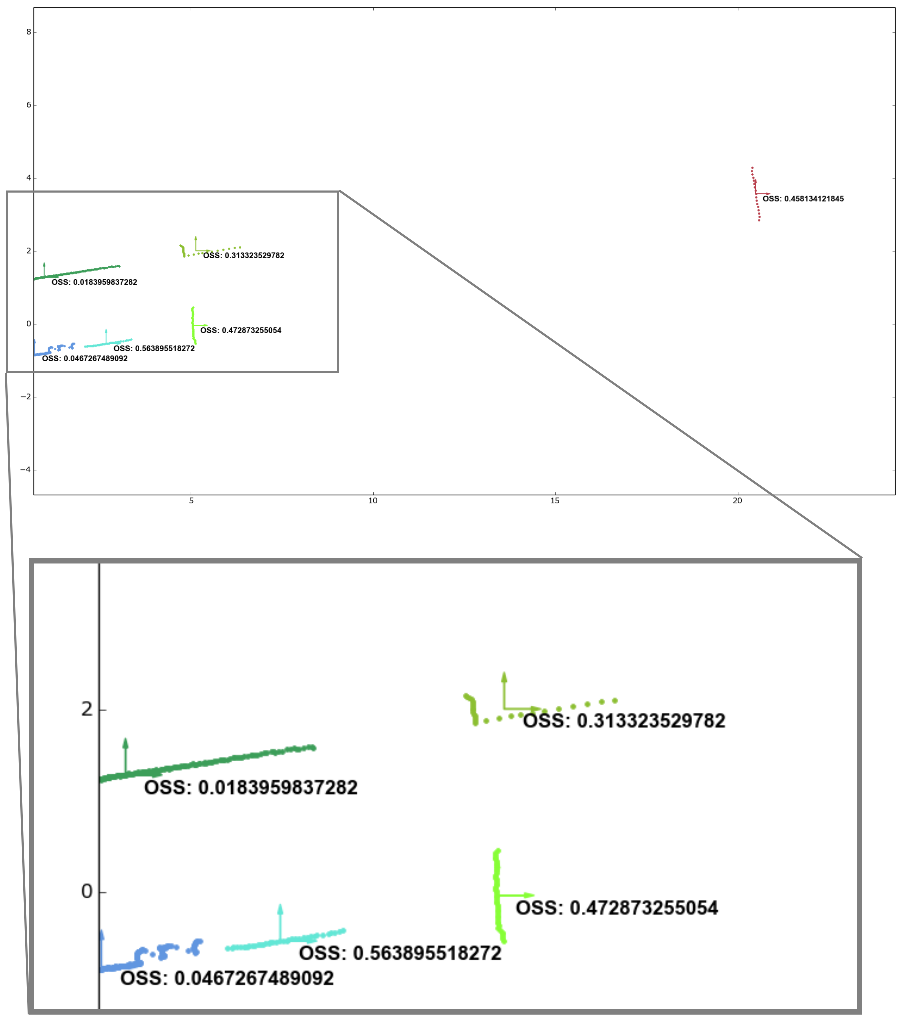

Figure 5.2: Object Saliency Score Failure

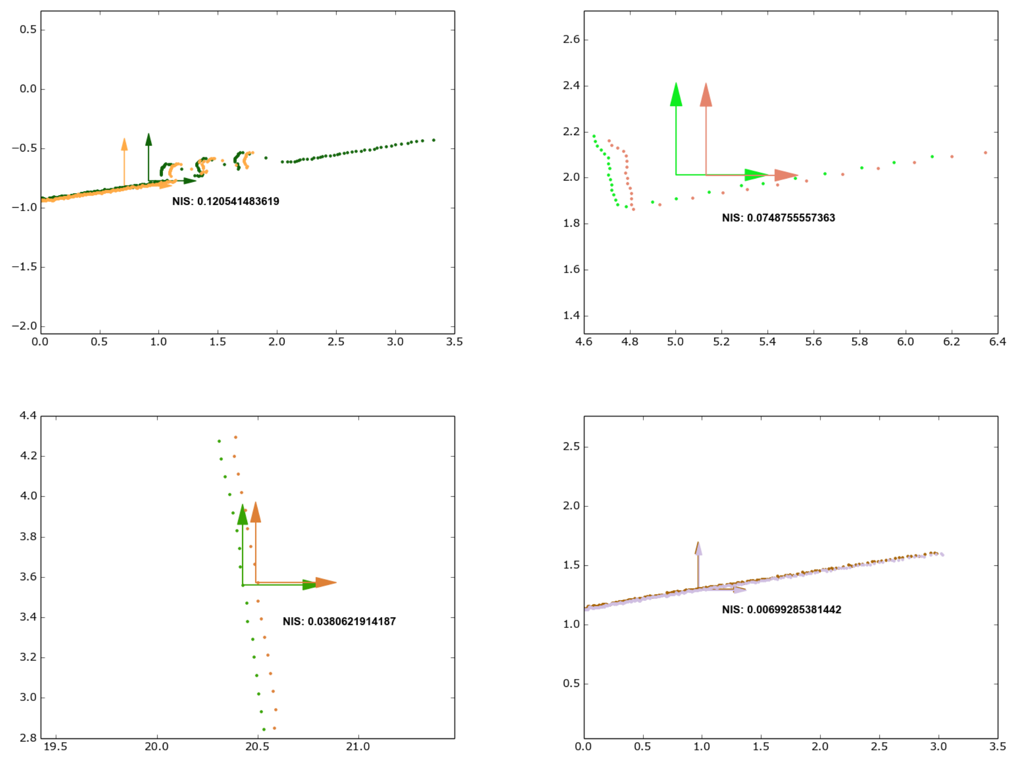

Figure 5.3: Association through NIS

As mentioned above, due to the lack of clusters and their inherent linearity, it is impossible to reject them at the first place. Thus, the OSS test is disregarded.

Data Association

The association is done based on the Normalised Innovation Squared. Each candidate Landmark's position is compared with the position of each Landmark in the map through the NIS metric. The landmarks with the smallest NIS distance are considered to represent the same area of the real world. The outcome of this metric is illustrated in Figure 5.3

Of course the association is not always right. Figure 5.4 shows some faulty associations. These candidate Landmarks are very closely located. Thus the candidate Landmark is not used to initialise a new Landmark in the map, and it is wrongly associated instead. Faithfully, the faulty associations will be rejected by the validation gate.

PointCloud Alignment

The PointClouds of the associated Landmarks are aligned using the Iterative Closest Point algorithm which yields the relative pose of the two Landmark frames.

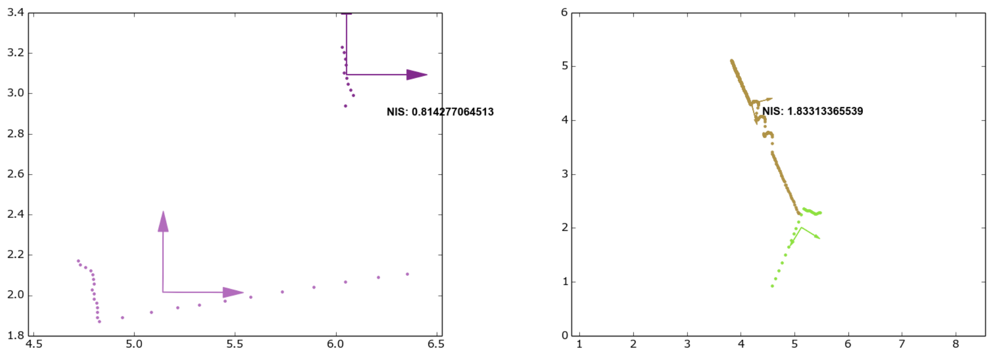

Figure 5.4: Faulty Association

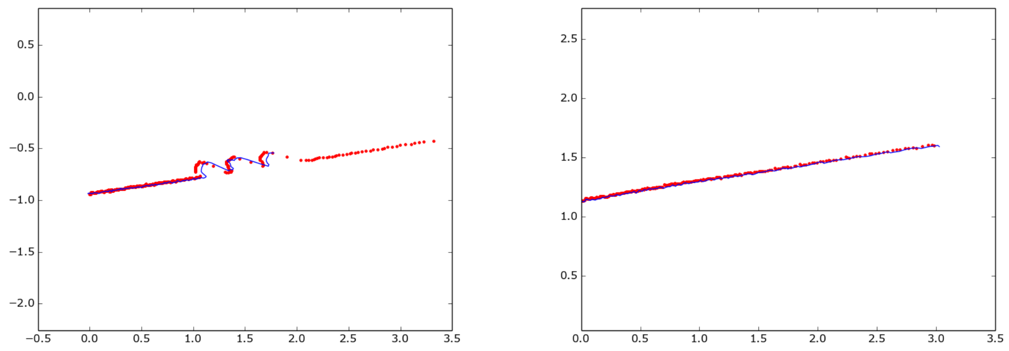

Figure 5.5: PointCloud Alignment using ICP

Figure 5.5 illustrates the results after the alignment. Any possible alignment error across straight lines is a previously known disadvantage, but it is an accepted tradeoff since the OSS test has been put aside.

The true disadvantage of the ICP algorithm is revealed in Figure 5.6. Plots on the left column have axis with equal step whereas the same PointClouds on the right column are plotted with axis stretched accordingly, for better illustration. As expected, the ICP treats both directions in the same way. As a result, “long” PointClouds that are developed along a specific direction tend to result in faulty alignment.

Validation Gating

The validation gate consists of three independent tests. All of them must be passed in order to consider an association as valid. The Mahallanobis Distance test comes first, since it has the lowest computational needs.

Next come the overlap and shape matching criteria. Figure 5.7 shows that the val-

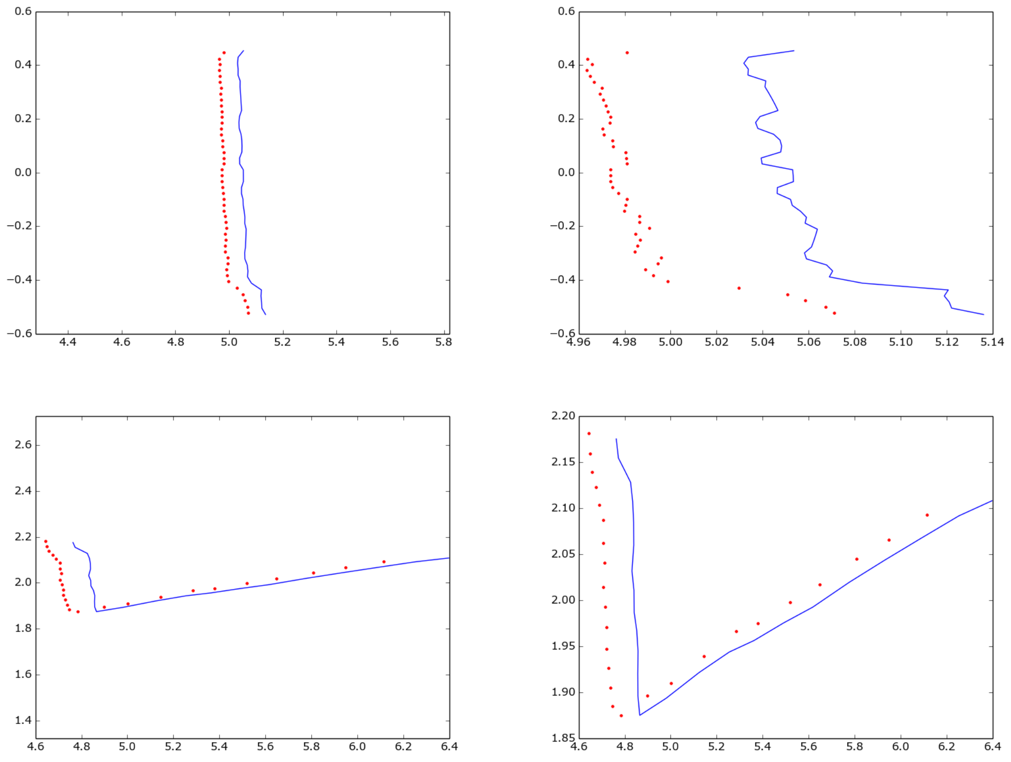

Figure 5.6: ICP Failure

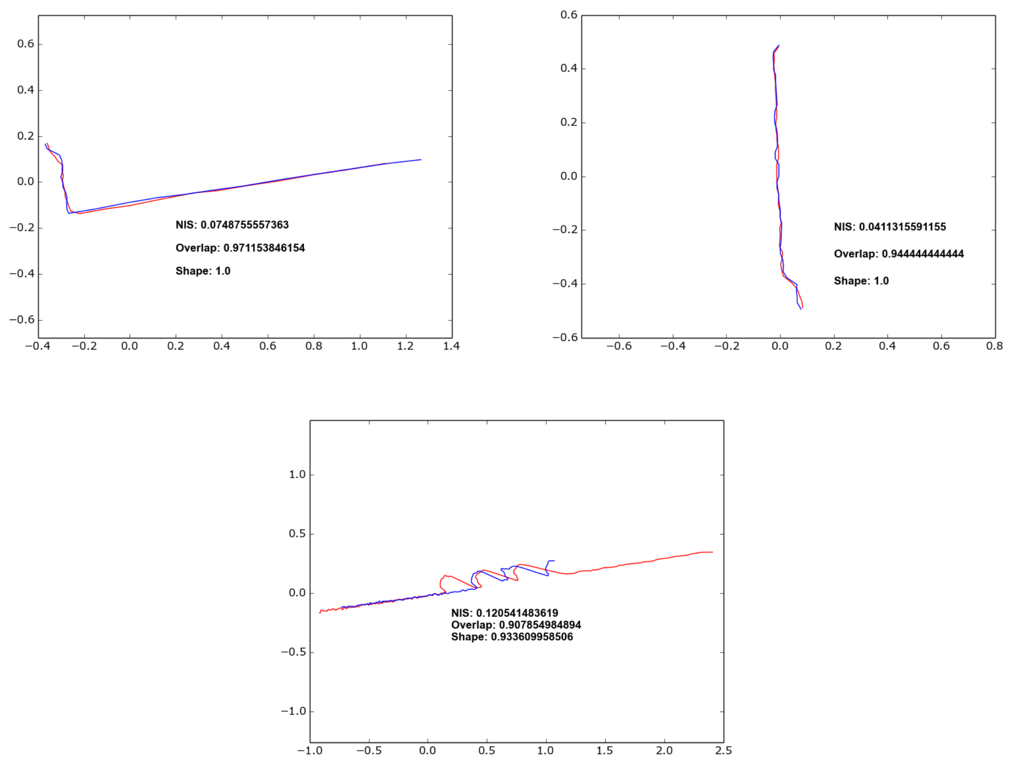

Figure 5.7: Valid Associations

idation gate accepts good matchings but if the ICP provides a bad, but plausible association, like the third plot, the validation gate can not tell the difference. The eventuality of the existence of matchings like that in the third plot raises consideration about the PointCloud fusion technique. It is obviously possible that fusing poorly aligned PointClouds would diminish the quality of the stored Landmark in map.

Figure 5.8 shows associations that have been rejected either because of the overlap threshold or because of the shape matching gate.

Landmark Fusion

After the validation gate test, the aligned PointCloud of the observation is fused with the PointCloud of the Landmark in the map. If the number of points in the resulting PointCloud exceeds a threshold, the new PointCloud is downsampled.

Landmark Fusion could be a valuable technique since it gradually improves the quality of the stored PointCloud as shown in Figure 5.9

However, it is extremely vulnerable on pour alignment. Figure 5.10 clearly illus-

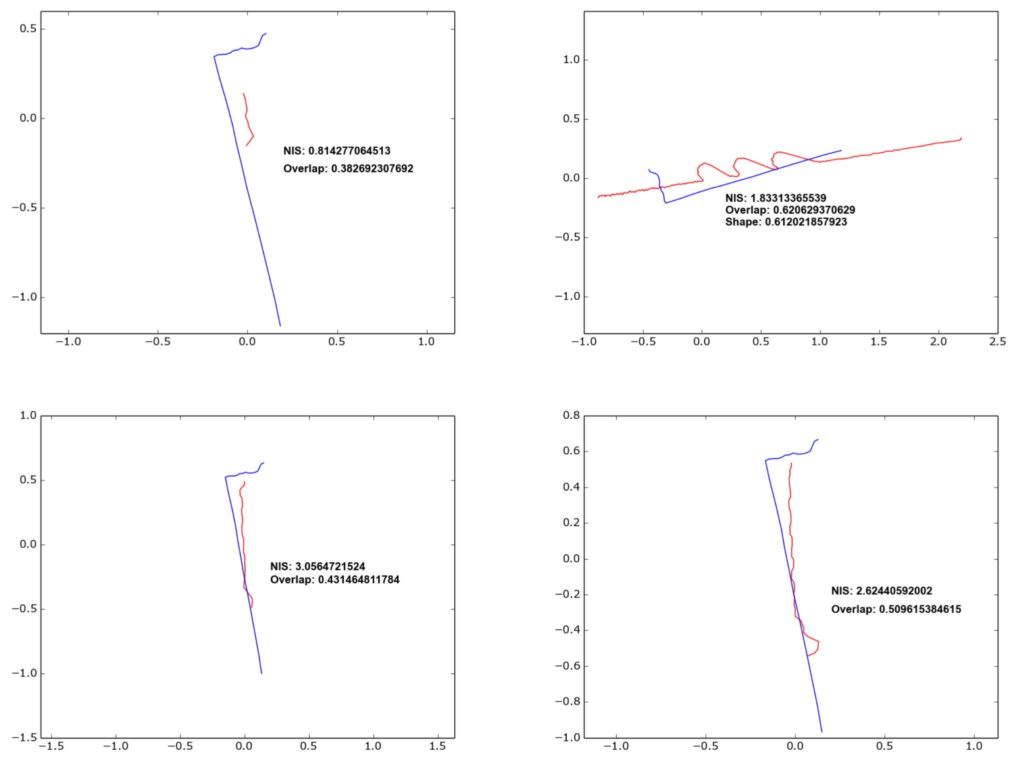

Figure 5.8: Rejected Assoiciations

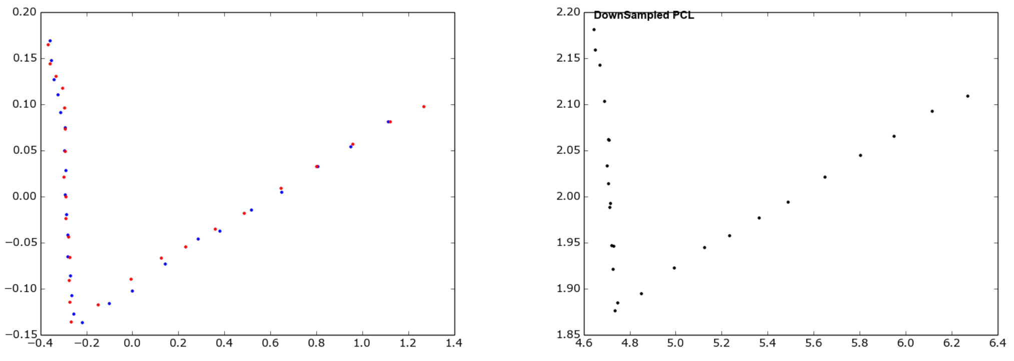

Figure 5.9: Successful Fusion

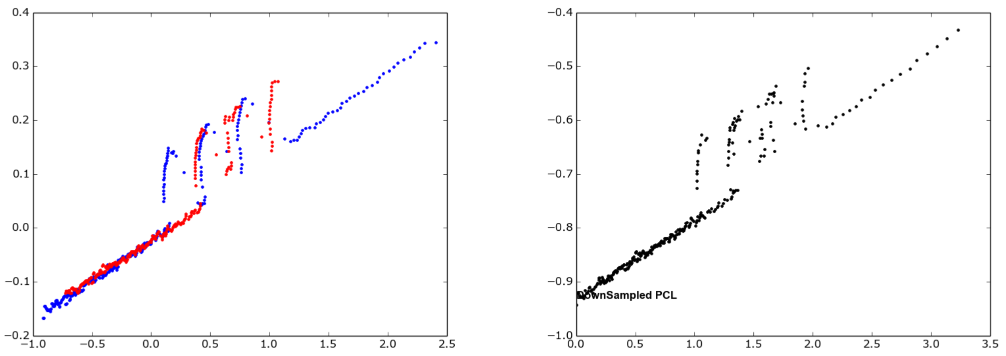

Figure 5.10: Fusion Failure

trates this issue while the quality of the new PointCloud is obviously reduced.

Navigator

The designed Navigator is evaluated based on the decision taken at each step. No sophisticated path planning is performed. Thus, although better choices could be made in some cases the navigator makes the best choice to perform the short term task given.

The importance of each factor that the Navigator considers to chose the next point is configured using the weights as described in the previous chapter. This results in an extremely flexible scheme that makes the evaluation of the Navigator a very difficult and opaque task. However, Figure 5.11 shows a set of cases that it seems to work decently.

However, disadvantages exist as well. Figure 5.12 illustrates the most important one. Considering the centroid of the Landmark as the point to which the robot is attracted to, some of the candidate points located behind the Landmark may wield a better utility. This forces the robot to pass through it or even get stuck behind a wall for example if there is an obstacle between.

A second disadvantage arises when the Landmark is not the final target but an intermediate goal. In this case, the robot should pass by instead of driving directly towards it. Taking into account the last two comments, the need of a sophisticated path planner such as that designed by B. Tovar et al. in [12] seems to be indisputable.

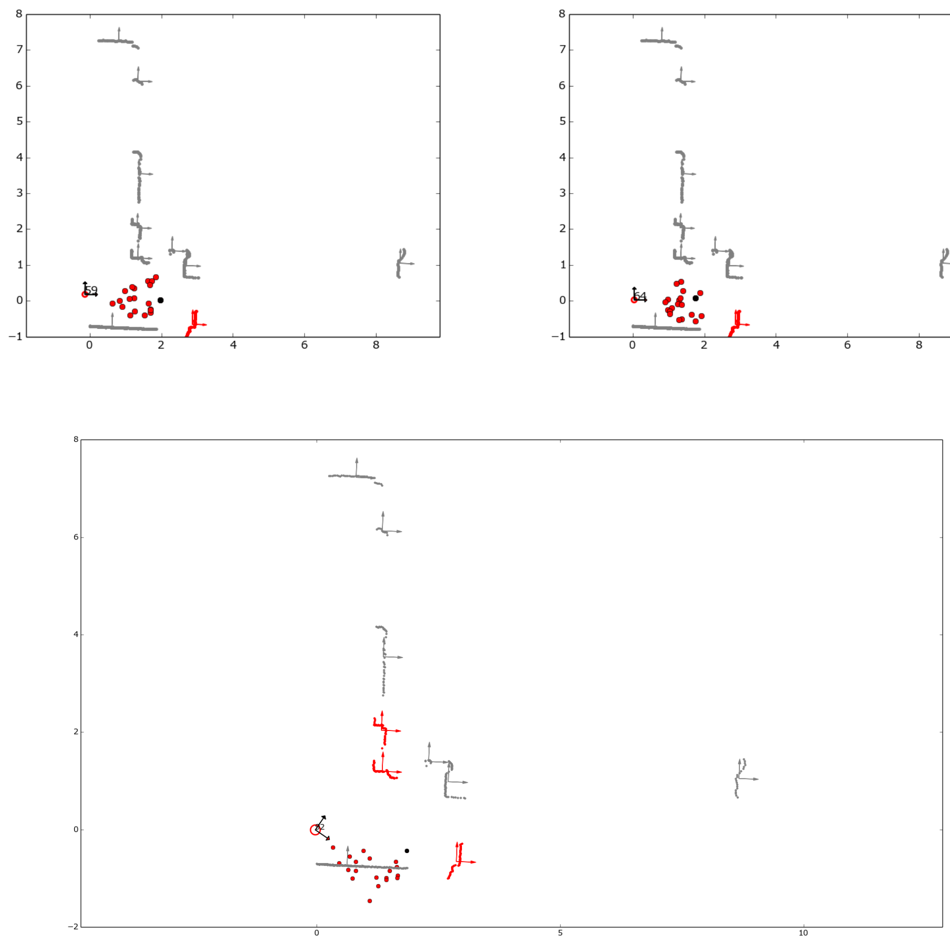

Figure 5.11: Successful Navigation

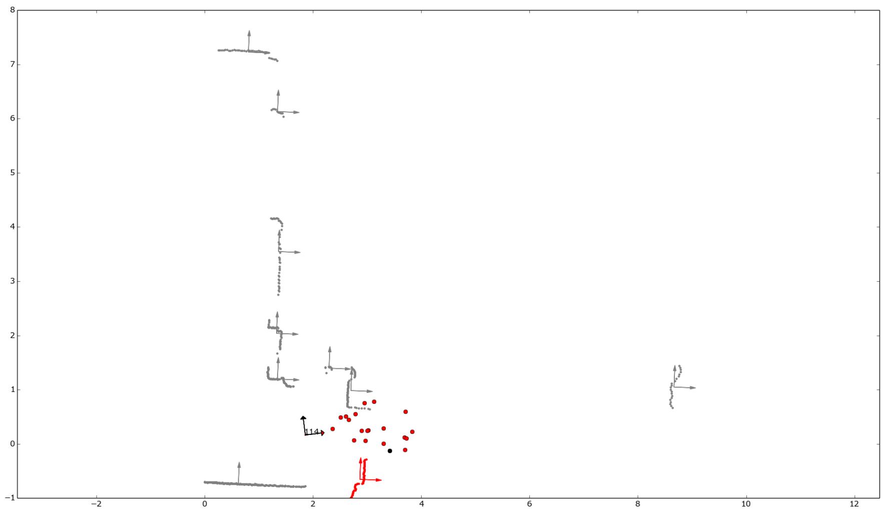

Figure 5.12: Navigator Failure

Chapter 6

Testing

As mentioned before, the proposed navigation method can be applied on any SLAM system that treats real world objects as Landmarks. Thus, the testing phase is focused on the navigator rather than the SLAM subsystem. The series of tests that follow, intent on enlightening the potential of the method while exploring its weaknesses and the necessary improvements.

6.1 Experiments

The first test tries to prove that the utility function works properly. The robot is instructed to go to the first door on the right, print a message when it arrives and stop moving. Figure 6.1 illustrates the behaviour in this case. Notice that although the shortest path would be that of moving straight to the target, the robot prefers to move on a curved line favouring positions that maximise its view of the unknown parts of the map.

The next challenge is to implement a real life scenario, where the waypoints consist of a sequence of targets, where the robot passes by the intermediate ones and ends up to the final Target. On the second test, it is instructed to pass the first door on the right just printing a message when it achieves this first goal and then continue to the end of the corridor where it shall print a message and stop. Figure 6.2 illustrates its behaviour in this case. Here, it is essential to emphasise the importance of the target achievement parameters. In this specific case, the first target is considered to be achieved as soon as the robot gets within a radius of 1.5m from the frame origin of the landmark. Configuring this parameter in a different way, would yield a different path as illustrated in the second example in Figure 6.2

Taking the difficulty to the next level, the robot is instructed to visit the first corner, pass by the door on the right and stop at the end of the corridor. An alternative way to achieve the same goal is to instruct the robot to visit the first corner, turn left, pass by the door on the right and stop at the end of the corridor. The reaction of the robot in these tow cases is illustrated in Figure 6.3

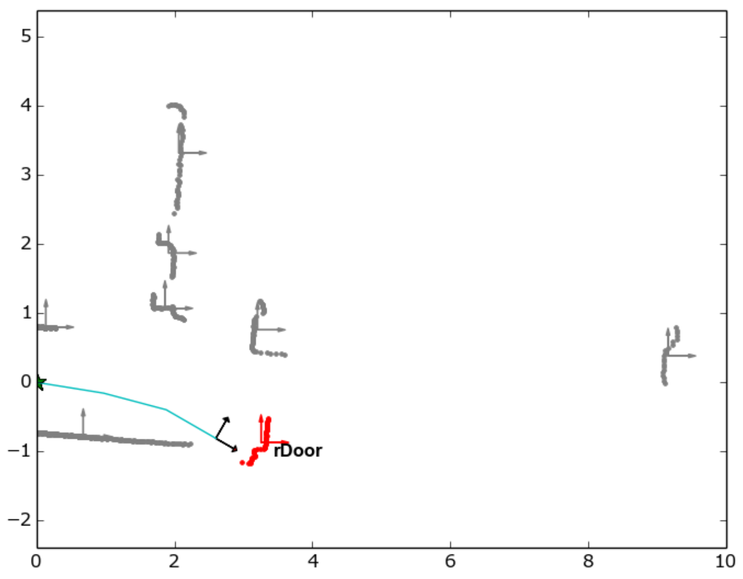

Figure 6.1: Simple Test

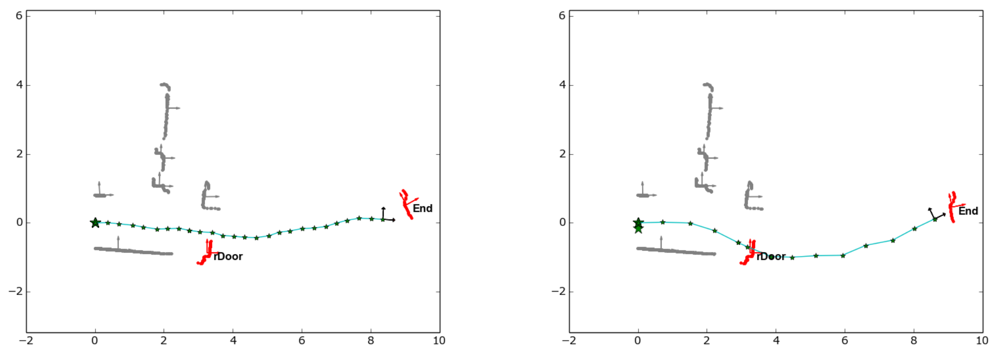

Figure 6.2: Target Sequence

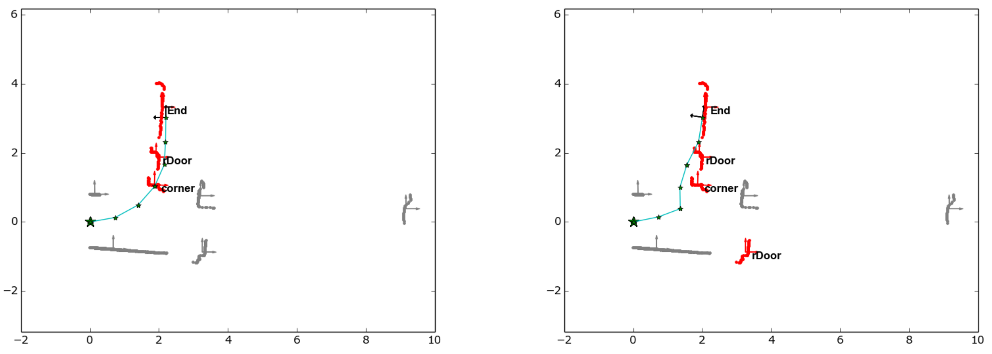

Figure 6.3: Moderate Test

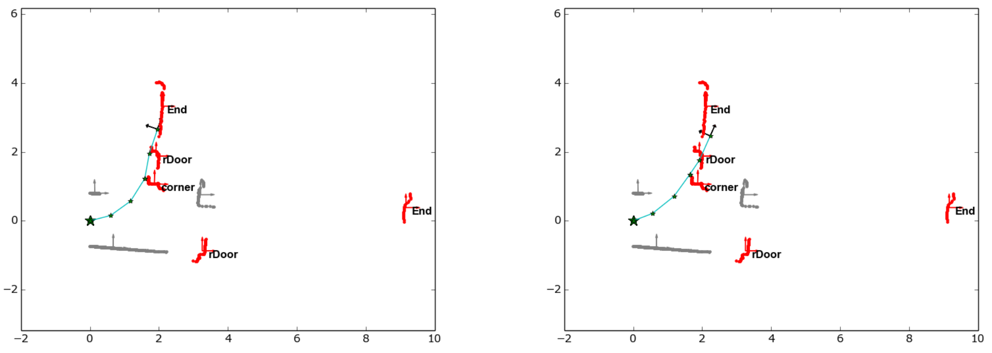

Figure 6.4: Redundant Tags

The next two experiments intent to reveal the Achilles heal of the designed navigator, the behaviour of the Target Scouter. The demonstrated system chooses the closest Landmark with the same tag to represent the given Target. Thus, in the first experiment in Figure 6.4 where the robot is instructed to move towards the corner, pass by the door on the right and finish at the end, it chooses the desired rDoor since it lies closer than the other one.

However, this is not always the case as shown in the second experiment in Figure 6.4 where the robot is instructed to pass by the door on the right and finish at the end of the corridor. Since there is an rDoor in a nearby corridor that is closer than the rDoor located in the current one, the robot moves towards the wrong one.

6.2 Evaluation

The tests described above prove that the first barrier has been successfully surmounted. The utility function serves its purpose by driving the robot from its current location to the next Target in an optimal way. Achieving this first goal is crucial for any *Landmark Characterisation SLAM (LC-SLAM)* system. Having this result in hand, it is important to explore the arising issues in order to design a practical system.

Taking an attentive look on the experiment results, one can notice the following defects:

Paths Across Landmarks: While the desired obstacle avoidance demand is successfully achieved by the navigator (all the chosen points are at least 0.3m away from the nearest obstacle), the resulting path is "unaware" of the obstacle existence. Choosing each next point in isolation results in paths that forget their history and tend to move through obstacles.

Poor "Pass By" Behaviour: Chasing the frame origin of the Landmarks seems to be a bad idea. When the robot tries to achieve an intermediate goal, it moves towards its centroid ignoring that it does not really need to get there, but it should just pass by the Landmark and mark the Target as achieved.

Target - Landmark Association: Choosing the closest Landmark with the desired tag as the next Target seems to work erroneously in complex maps. This method completely ignores the topology of the map and reduces the acceptable level of abstraction that the target setting language tries to achieve.

Map Coherence: Although it is not clearly visible, the definition of the free edge is spurious. Because of the scanSLAM nature, it is impossible to have a coherent map. There is a chance that the robot visits a place of the real world that does not produce any Landmarks on the map. This part, is obviously visited, known but unmapped. The described implementation will yield a free edge in this case that could never be erased after a visit on the place and thus, it will permanently act as an attractor for the robot.

6.3 Solution Outlines:

The evaluation of the behaviour of the designed system shows the way to the required improvements. What follows is a sketch of some possible solutions to the problems described above, that when applied would result to an overall upgrade of the system.

1. **Path Planner:** The incorporation of a more complex utility function is unavoidable. It is necessary to follow the example of B. Tovar et al. in [12] where they designed a utility function that evaluates whole paths rather than isolated points. This leads to the need of a sophisticated path planner to produce candidate paths connecting consecutive targets, in order to be evaluated by the utility function.

2. **Target Definition:** It is inevitable to define a new model for the targets. The utility function should be aware of the area covered by the Target and whether it is an obstacle or not. Moreover, the distance to Target should be defined differently

so that the robot moves towards a point near the Landmark rather towards it's centre.

3. **Target Identification:** One of the most important defects of the new system is the faulty Target - Landmark association conducted by the Target Scouter. This system should be designed carefully, in order to provide the required high level behaviour of the system. It could extenuate any restrictions on the Target setting language expressiveness and transfer the burden to the semantic coupling of the desired Target and the Landmarks in the Map.

4. **Free Edge Revisit:** Although this augmentation is specific to scanSLAM, it helps to emphasise the importance of this delicate issue. There exists the need of erasing free edges located at visited places whether there is a Landmark in place or not.

Chapter 7

Conclusion

This thesis proposes an innovative method that gives the ability to a mobile robot to understand and execute direction orders described in human language. It builds on existing techniques used for mobile robot localisation and mapping in unknown environments (SLAM), and expands its capabilities by integrating the ability to move towards an abstractly predefined target within the unknown environment.

The mission of the robot is to go towards a target in an unknown environment. Whether a good map is constructed in the meantime or not, is not of great interest. Undoubtedly, since the map is needed for navigation purposes, it has to be at least as good as necessary to serve its purpose. Thus, the main purpose of the robot is not to build the best possible map of its environment, but to optimally map the environment along the route followed towards its destination.

The proposed method is based on a SLAM system to introduce a navigator that facilitates the desired features. There exists one prerequisite in order to formulate any SLAM system so that the new navigator can drive the robot towards a target. The SLAM system should treat the objects of the world as whole entities called Landmarks.

A simple system has been designed to prove the feasibility of the methods implementation. During the testing of this system, it has been shown that the proposed method can be used on real world systems and it has shown the pathway for the required future work.

7.1 Summary of Contributions

In order to implement the new system, two key components were essential:

1. **Pseudo-Human Language:** The designed language provides the framework to enable the human - robot communication. It formulates the way that a human can talk to the robot so that it understands the given orders.

2. **SLAM Requirements Formulation:** In addition to the human requirements (speak the pseudo-human language), the robot has to do its part as well. Provided

that the SLAM system can accommodate the Landmark entity, the proposed navigator enables the robot to execute the given orders.

7.2 Future Work

There is yet a lot to be done in order to create a practical LC-SLAM system.

1. **FastSLAM:** Having in hand the know-how regarding the navigation towards a target, what comes next is the formulation of the state of the art SLAM system to accommodate the new method.

2. **Navigator Improvement:** The Navigator should be revisited so that a real path planner takes charge and cooperates with a redisigned utility function that evaluates whole paths.

3. **Target Scouter:** It is inevitable to design a sub-system for the Target-Landmark association. This “Target Scouter” should use the map to extract topological information in order to correctly choose the Landmark that represents each Target in the given Orders.

Glossary

actuator

is a mechanical component that is used by the robot to do physical work on the environment. E.g. a mechatronic arm. 15

control

is the input signal of a system. It is used to change the inner state of the system towards a desired value. 43

HCI

stands for Human Computer Interaction system. It refers to a computer system specially designed to facilitate a smooth and easy communictation and interraction with humans. 7

kernel

is abusively used throughout this thesis to describe a data fusion system used by SLAM algorithms. Data fusion systems are used in order to optimally combine heterogeneous data sources to produce a valuable result. A SLAM kernel combines data acquired by different types of sensors to produce an optimal estimation of the map of the world. 18

MSE

stands for Mean Squared Error. It measures the average of the squares of the errors between respective points of two signals. 23

noise

is an unwanted signal that disturbs a primary signal of importance. Noise can be involved in the primary signal in various different forms. For instance, it can be additive or multiplicative. It may be stochastic or deterministic. 18

observation

is an entity that aims to describe a depiction of the world acquired by the robot. Different designs of the SLAM kernel require different formulations for the observation entity. For instance, a specific kernel could use a laser scan as an observation, whereas a different kernel might use the pose of the laser scan instead. 18

Order

is one of the two components of the designed language. It is an action that has to be done once the robot reaches a Target e.g. knock the door, enter the room, etc. 43

PointCloud

is a term devised for the purposes of this project. It signifies a set of points, usually laser readings. 37

pose

is a vector that completely defines the position and orientation of a robot in space. A car like vehicle's pose can be determined by two values to represent its position on a plane and an angle to represent its orientation. However, a robot that can move in three dimensions needs more parameters to determine its pose, namely three values for its position and three for its orientation. 17

RGBD

An RGBD sensor is a hybrid sensor that is composed by a camera, which yields an RGB (image) signal and a Depth (image) signal. Each pixel of the RGB image is accompanied with a “depth” value that measures the distance of the object represened by the pixel from the sensor. 17

sensor

is a device that measures a physical quantity and converts it to a signal. Some examples of sensors are a thermometer, a camera, etc. 15

stochastic

is a system whose state at any given time is random. A stochastic system could yield different outputs given the same input. 18

Target

is one of the two components of the designed language. It is an abstract description of the world's objects e.g. a Door, a Window, etc. 43

uncertainty

in measurements is a measure of the confidence for the measured value. A value with large uncertainty is more unlikely to be correct. 17

Bibliography

[1] Tim Bailey and Juan Nieto. “Scan-SLAM : Recursive Mapping and Localisation with Arbitrary-Shaped Landmarks”. In: *Robotics: Science and Systems Conference (RSS)* (2008).

[2] Tim Bailey, Juan Nieto, and Eduardo Nebot. “Recursive scan-matching SLAM”. In: *Robotics and Autonomous Systems* (2007).

[3] Tim Bailey, Juan Nieto, and Eduardo Nebot. “Scan-SLAM : Combining EKF-SLAM and Scan Correlation”. In: *International Conference on Field and Service Robotics* (2005).

[4] Hugh Durrant-Whyte. “Uncertain geometry in robotics”. In: *IEEE Trans. Robotics and Automation* (1988).

[5] Hugh Durrant-Whyte and Tim Bailey. “Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms”. In: *Robotics & Automation Magazine, IEEE* (2006).

[6] Jose E. Guivant and Eduardo Mario Nebot. “Optimization of the Simultaneous Localization and Map-Building Algorithm for Real-Time Implementation”. In: *Robotics and Automation, IEEE Transactions* (2001).

[7] Abhinav Gupta et al. “From 3D Scene Geometry to Human Workspace”. In: *Computer Vision and Pattern Recognition(CVPR)* (2011).

[8] M. Montemerlo and S. Thrun. “Simultaneous Localization and Mapping with Unknown Data Association Using FastSLAM”. In: *Robotics and Automation, 2003. Proceedings. ICRA ’03. IEEE International Conference* (2003).

[9] Maria I. Ribeiro. “Kalman and Extended Kalman Filters: Concept, Derivation and Properties”. In: *Institute for Systems and Robotics* (Feb 2004).

[10] R. Smith and P. Cheesman. “On the representation of spatial uncertainty”. In: *Int. J. Robotics Research* (1987).

[11] S. Thrun. “Particle Filters in Robotics”. In: *Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI)* (2002).

[12] Benjamın Tovar et al. “Planning Exploration Strategies for Simultaneous Localization and Mapping”. In: *Robotics and Autonomous Systems* (2006).

[13] Theodoros Varvadoukas, Ioannis Giotis, and Stasinos Konstantopoulos. “Detecting Human Patterns in Laser Range Data”. In: *Frontiers in Artificial Intelligence and Applications* 242 (2012), pp. 804–809. DOI: 10.3233/978-1-61499-098-7-804.

[14] Chieh-Chih Wang and Chuck Thorpe. “Simultaneous Localization and Mapping with Detection and Tracking of Moving Objects”. In: *IEEE International Conference on Robotics and Automation (ICRA)* (2002), pp. 842–849.