Localisation and Navigation in GPS-denied environments using RFID tags

by

Sisa James

Department of Electrical Engineering

at the

University of Cape Town

Supervisor

Robyn A. Verrinder, *University of Cape Town (UCT)*

Co-supervisors

Deon Sabatta and Ali Shahdi, *Council for Scientific Industrial Research (CSIR)*

A dissertation submitted for the degree of

Master of Science in Electrical Engineering

Pretoria, February 2014

PLAGIARISM DECLARATION

I declare that this dissertation is my own unaided work, both in conception and execution, and that apart from the normal guidence of my supervisor(s), I have received no assistance apart from that which has been stated.

SIGNATURE: Signed by candidate

DATE: 17/02/2014

Abstract

This dissertation addresses the autonomous localisation and navigation problem in the context of an underground mining environment. This kind of environment has little or no features as well as no access to GPS or stationary towers, which are usually used for navigation. In addition dust and debris may hinder optical methods for ranging. This study looks at the feasibility of using randomly distributed RFID tags to autonomously navigate in this environment. Clustering of observed tags are used for localisation, subsequently value iteration is used to navigate to a defined goal. Results are presented, concluding that it is feasible to localise and navigate using only RFID tags, in simulation. Localisation feasibility is also confirmed by experimental measurements.

Acknowledgements

I would like to thank the CSIR/MIAS for providing all the resources to perform this research. I would also like to thank my co-supervisors Ali Shahdi and Deon Sabatta from the CSIR. Lastly, I would extend my gratitude to the University of Cape Town (UCT) and specifically my supervisor Robyn Verrinder for all of her support.

Table of Contents

List of Figures

List of Tables

Glossary

CMI - Centre for Mining Innovation, the CSIR group focused on improving mining safetey and efficiency in South Africa

CSIR - Council for Scientific Industrial Research, a state owned research organisation, founded to improve the quality of life of the people of the Republic of South Africa, through scientific research and invention.

Localisation - this is the process of placing and orienting oneself in an environment

MDP - framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker (def: wikipedia)

MIAS - Mobile Intelligent Autonomous Systems, the mobile robots software group of the CSIR

MMM: Mechatronics & Micro-manufacturing, the mobile robotics hardware group at the CSIR

MSP - Mining Safety Platform, the iRobot mobile robot platform used to test robot autonmating/intelligence software

RF - Radio frequency refers to signals transmited between 3 kHz to 300 GHz frequency bands

RFID - Radio Frequency Identification tags, these are small computer chips with antennas, that absorb electroomagnetic energy and use it to transmit a unique tag number and any addional data such as RSS

RSS - Received signal strength, this is an indication of the power present in a recieved radio signal

SLAM - Simultaneous localisation and mapping, refers to the process of concurrently building a map and placing oneself in the map, in an unknown environment

1. Introduction

1.1. Background

South Africa is the fourth largest gold producer in the world and has the second most gold reserves in the world after Australia [1]. As a result of this resource concentration, coupled with a lower temperature vs. depth gradient, it now has some of the world's deepest gold mines, reaching depths in excess of 3000 m [2] [3]. At these depths, the temperature of rocks can reach up to 60 °C. The rocks are also prone to exploding [4] and there is limited air circulation. These deep mines need massive refrigeration and ventilation systems [5]. The working conditions are both uncomfortable and hazardous. Rock explosions are a real hazard and in South Africa kill roughly 20 miners each year [6]. As of 2012, gold mining in South Africa has resulted in more than 120 deaths every year over the past three years [7] and more than 3 500 injuries [8].

The most deaths occur after the blast stage in mining process. The basic mining process in South African gold and platinum mines consists of three stages [9]. First, holes are drilled in the ore-bearing rock wall to place explosives. Second, the rocks are blasted using the explosives. After the blast and once the explosive gases have cleared away, a miner inspects the area and checks if it is secure. This is the most dangerous stage and researchers are investigating the use of autonomous robots [9] to alleviate the danger.

Most efforts in mobile robotics are focused on creating robots that can autonomously perform tasks too tedious or dangerous for humans to do [10]. These can range from driving long distances, to delivering medical equipment in warzones. Some of these tasks are often performed in hazardous and/or unpredictable environments. Such tasks may include search and rescue operations in burning or collapsed buildings. In the South African context at the Council for Scientific Industrial Research (CSIR), autonomous vehicles are being developed to inspect the safety of working areas in mines after the blast stage. In this task a robot should enter and inspect the hanging wall (roof) of a mine for structural integrity and possible rock fall hazards after a blast. This would eliminate the danger in requiring a human miner to perform this inspection, as is currently being done.

1.2. Project Motivation

The CSIR Mobile Intelligent Autonomous Systems (MIAS) group, together with Mechatronics & Micro-manufacturing (MMM), Material Science & Manufacturing (MSM) and Centre for Mining Innovation (CMI) groups have undertaken to design and implement

intelligent algorithms to use on a mobile robot platform for mapping and navigation in an underground mine. This is part of a project known as the *Mining Safety Platform* (MSP) [11].

One of the main issues with sending an autonomous vehicle into an unknown environment is designing a robust navigation and localisation scheme. This task becomes even more difficult in an underground environment where no GPS signals or stationary landmarks are available.

1.3. Problem Statement

This study attempts to solve the underground environment (i.e. no global reference signals) localisation and navigation problem, by using RFID tags spread manually throughout the environment as markers. Later the vehicle may even place these tags itself using a method like a breadcrumb trail. The dissertation will build on previous work done by Forster [9] by implementing a snapshot matching localisation algorithm on the mining safety platform, testing it in a controlled environment and improving clustering of tags, detections and robustness of the system. Some of this work has been published at the RobMech 2012 Conference [12].

1.4. Importance of Research

As mentioned above, mines remain a hazardous environment for humans to work in. In addition to the cost of human lives and injury claims to mining companies, trade unions insist on taking one day of production time-out to mourn & assess safety for every death which occurs in the mine [13]. This can lead to huge losses for South African mines. The CSIR has a duty to assist thought its mandate which reads, *"...through directed multidisciplinary research and technological innovation, to foster, in the national interest...industrial and scientific development, and thereby contribute to the improvement of the quality of life of the people of the Republic..."* [14].

1.5. Scope and Limitations of Research

For the purpose of this research, the environment will not contain any obstacles, as a result obstacle detection and avoidance will not be addressed in this project. Additionally the environment under consideration will be static, not dynamic (i.e. the environment does not change with time). Additionally this work intends to address only localisation within an environment and subsequently navigation within said environment and hence should not

be confused with the simultaneous localisation and mapping (SLAM) problem, which consists of mapping and localisation at the same time.

1.6. Plan of Development

This dissertation will start with the literature review, followed by the main body, which contains the methodology of the simulation and the simulation results, as well as the methodology of the experimental setup and results. We then discuss the results and draw conclusions, followed by possible future work in this area. Finally we have the acnkowledgements and references ending with the code appendix. In the literature review, we discuss the current research efforts concerning navigation and localisation, including methods which employ RFID technology. We also explore the mathematical tools used in these areas. In the simulation and experimental setup sections, we discuss the modelling of the simulator and how the experiments will be conducted. Theses sections each have their corresponding results sections. We then discuss the results from both simulations and experiments in the discussion section. In the future work section, we briefly explore possible future avenues for this research; closing with the acknowledgement of contributions to this work.

2. Literature Review

2.1. The SLAM Problem

For an autonomous vehicle to perform inspection tasks in an underground environment, it needs to be able to know where it is (localise) in a given environment so that it can navigate within this complex environment. Thus the ability for the robot to accurately localise and map its environment becomes very important. This is known as the SLAM problem. Although the scope of this work focuses only on localisation and navigation, it is worthwhile to explore the literature on the SLAM problem, as much of the the work is relevant to localisation. Extensive research has been done to address the SLAM problem, especially with the introduction of the Defence Advanced Research Projects Agency (DARPA) Grand Challenge [15] [16]. For many, it is considered solved at least from a theoretical perspective [17], although implementation in the real world remains a challenge. The *Google Self Driving Car* is an example of a system which implements SLAM in the real world [18]. This vehicle uses laser rangefinders, radar, cameras and GPS; the data from all these sensors is then fused to form the best estimate (belief) of the environment. Based on previously collected data and the current data the vehicle can then localise itself within the map using a variation of Kalman, or other filters [19]. However, "many practical issues remain, especially in complex outdoor environments" [17], such as accuracy and computational limitations when scaling to larger environments.

Most mobile robots utilise a combination of inertial sensors, odometry measurements and GPS antennas for localisation/positioning. The first two methods, although initially accurate, suffer from accumulating errors; GPS might be used to periodically zero these errors. Unfortunately the accuracy of GPS can vary in excess of 7 m [20]. Furthermore in an environment where there is insufficiently accurate or no GPS signal available, such as dense high-rise city blocks, underwater or underground, these methods lead to erroneous navigation. In [15] it is concluded that *"a localisation algorithm is needed to operate indoors"*, even in the case where a "robot is equipped with a state-of-the-art inertial navigation system".

2.2. Localisation and mapping in the absence of GPS

In [15], an autonomous vehicle is tasked with navigating to a predetermined goal in a multi-level parking lot, a case where GPS and inertial sensor data proves insufficient for the purpose. The results showed that it is possible to use a SLAM algorithm based on laser data to navigate a large scale indoor environment. Other solutions such as that in

[21] also employ laser range scanners to map, localise and navigate unknown terrain. This is done by using the scanners to generate a 3D point cloud (by tipping the laser sensor), which can be stitched to form an accurate 3D map; this map in turn is used to build a 2D map illustrating the drivable surface [15]. This method also allows a drivability cost map to be generated. The aforementioned algorithm can lead to accurate positioning in the absence of GPS; unfortunately, optical methods are not well suited to dusty environments, such as those in an underground mine. In addition, in an underwater environment visibility can be limited [22]. There has been some recent research in localisation and mapping in these unconventional environments. It is of interest to look at the approaches taken to address the localisation problem in both these environments, as they offer similar challenges when it comes to solving the localisation problem, such as low visibility and no GPS.

2.2.1. Underwater Localisation

In [22] Hayato et al. look at autonomously exploring the oceans for the purposes of visually investigating and mapping underwater structures. The experiment yielded positive results, although as pointed out by the authors, it is assumed that visibility is sufficient, therefore limiting applications somewhat. Hayato uses two laser beams and a camera to create a simple and accurate laser ranging system. For navigation from a known origin, given the structure of the surfaces, the vehicle can accurately localise and thus navigate underwater. The requirement of having to know the surface structure beforehand further limits the applications of this optical approach and the assumed visibility further illustrates the challenge of implementing optical localisation in sub-surface environments.

To address limitations of optical mapping and localisation underground and underwater, attempts have been made to use sonar/acoustic methods for localisation. Vassilis [23] looked at constructing a map and localising using sonar data; this was an early attempt at using sonar above ground for general mapping and localisation. The author notes improvements can be made to the positioning/localisation, but the work was incomplete. The approach of using sonar for mapping is more prevalent in underwater applications; this may be due to the fact that sound travels faster and further in water than in air [24]. One such example is in the paper by Zeng et al. [25] who investigates the use of forward looking sonar for the purpose of underwater SLAM. This particular approach works much like optical SLAM, in that it produces sonar images, which are used to create a 2D feature

map. This is later used to localise and navigate; the use of the Extended Kalman Filter (EKF) ensures consistent localisation.

2.2.2. Underground Localisation

An underground mine localisation and navigation method, referred to as opportunistic localisation in [26] takes advantage of the tunnel like nature of some mines, and uses a wall following technique in combination with weak localization (odometry). This method only requires fairly accurate localization at intersections and uses odometry for this. While this method works in navigating in haulages, it is unsuitable for navigation in the stope area.

2.2.3. RFID Localisation

Attempts have been made to address the GPS denied localisation problem using radio frequency identification (RFID) tags. These methods can be loosely categorised into two approaches. The first approach models the RFID tag sensor and uses it to estimate the location of the tag (with possibly known global co-ordinates) relative to the robot, and hence infer the pose of the robot. The second approach seeks to to capture snapshots of the RFID tags (from possibly known positions) and then match these snapshots from previously learned ones to infer the robots pose. Both approaches were investigated.

An example of the first approach can be seen in [27], [28], [9] and [29]. These approaches use a sensor model to estimate the locations of tags relative to the robot. This model is build by recording the received signal strength values of tags as well as the number of replies vs the number of queries at different locations around the robot. Hahnel et al [29] uses two antennas facing +45° and -45° from the robot heading, in conjunction with a laser range sensor. This work showed that using a combination of RFID and laser range data to localise a robot in an environment can significantly improve the localisation efficiency (the number of particles needed to accurately estimate the robot pose), when compared to using only laser data. This method showed localisation with an accuracy of at least 2m. It is demosntrated that combining laser range data and RFID data can reduce the computational demand for global localisation [29]. This is shown by the fact that 10000 particles are needed for efficient localisation using only laser data, whereas this is reduced to 50 particles if RFID localisation data is incorporated. Although effective, Hahnel's approach relies heavily on laser data for accurate localisation, using RFID localisation only to supplement/improve the laser localisation. Additionally, localisation accuracy better than 2m may be desirable for many applications.

In the work by Forster [9], one antenna is used, this, combined with the movement of the robot, is used to estimate the location of the tags as the robot traverses the environment. Like Hahnel [29], a sensor model is built for estimating the RFID tag locations. It is shown that it is possible to detect locations of RFID tags with an accuracy of 0.4 m. This approach shows a significant improvement in accuracy from Hahnel [29]; however the accuracy is dependent on the accuracy of the sensor model. Since the RSS values are influenced by the kind of surface they are attached to, it is difficult to find a sensor model which works well for more than one environment. Therefore they decided to use odometry to localise within/around a known RFID location/node over a short distance and use the RFID tag nodes to form a topological map for loop closure. However the dependence of the localisation accuracy on the sensor model remains an issue with these tag estimation approaches.

On attempt to overcome this limitation can be seen in [30], which uses 6 antennas arranged to cover a 360° field of view around the robot. As such, only a physical sensor model (as opposed to the experimental/statistical histogram model used in [9] and [29] above) is needed to estimate the location of each tag. This approach uses the signal loss (the signal strength sent by the antenna vs the signal strength received by each tag) over distance, in conjuction with the 6, 360° antena coverage to estimate the distance and bearing of each tag. By adjusting the antenna power of subsequent tag queries (tags further away will not reply if there is less power), the distance of tags can be estimated and depending on which antenna receives the maximum RSS, the direction can be estimated. This method takes into account the multiple paths the RF signal can propagate through (such as bouncing of walls and ceilings). This approach exhibits 6.1° mean bearing error and 0.69m mean range error [30]. This approach uses many antennas and it may be desirable to have simple system, also the accuracy may also be insufficient for some applications.

The second approach to the localisation problem is to use many tags and record and match snapshots of the RFID environment such as the method employed by Schneegans et al. [31], which uses four RFID antennas. This work is further developed by Vorst et al. [32]. Their work consists of issuing a series of RFID enquiries in an attempt to read all the RFID transponders in the read range. The tag ID’s and their number of responses are used as a fingerprint for each snapshot, essentially matching the robot pose with a corresponding snapshot. This method exploits the inherent spuriousness/irregularity of

transponder replies to construct a reply probability distribution for a given robot pose. It is also assumed that the probability distribution remains fairly constant for small changes in robot pose; this assumption was supported by experimental data with a few random exceptions.

During training, the robot captures a large number of snapshots with their corresponding robot pose. This robot pose is measured through odometry data. During localisation the robot uses these snapshots to compute particle filter weights and hence the most likely robot location/pose based on the current scan and the most recent action (using a motion model for the robot). In this work a B21 service robot is used with four ALR-8780 Alien Technology UHF antennas, used in pairs, one pair to transmit and one to receive.

In their papers Schneegans et al. [31] and Vorst et al. [32] conclude it is possible to localise using densely distributed RFID tags. They achieved a localisation accuracy of 0.4 m or better in an indoor environment. Despite the fairly accurate localisation, the work above still has a few drawbacks; firstly the need for a reference system in the training phase using odometry can lead to bad localisation if the snapshots during the training phase are not recorded accurately. Secondly the current system is unable to incorporate changes in the RFID environment.

This work is motivated by previous work by Forster [9] also carried out at the CSIR, by using some of the experimental data for creating the simulator. The aforementioned method uses groups of sparsely distributed RFID tags to form a hybrid metric-topological map to solve the SLAM problem. Topological maps consist of nodes which are connected in a graph like structure; this connectivity graph represents the topological map. In the work by Forster, the nodes represent groups of sparsely distributed RFID tags. This grouping is achieved in real-time, by either assigning a newly observed tag to the current node or creating a new node, a normalised cut algorithm is used to determine whether to assign the tag to the current node or create a new one. When navigating within a node, the robot uses a metric map. This map is created by estimating the location of each RFID tag using a particle filter (explained in section 2.3.2 Particle filter) and then using that to infer its own pose in the local metric map. First a sensor model is built using experimental data gathered from RFID tag responses at various distances, orientations of tags. Data are also gathered to measure the effect of different antenna heights.

The SLAM method presented in this dissertation uses a method similar to that presented by Vorst et al. in [32], and builds on it by adding planning and self-navigation and unlike Vorst [32] and Schneegans [31], we intend on using a single antenna. We also make use of RFID tag response data gathered by Forster [9] for modelling the reader response in the simulator. Our method presents a novel way of exploring and mapping the RFID environment while building a transition matrix during the exploration phase. The vehicle can then use this, along with the internal map (the created map is topological and internal to the robots programming environment), to navigate to any mapped goal location. Our approach also allows proper vehicle orientation to be preserved when reaching the goal and does this using only passive RFID tags and a single antenna.

2.3. Tools used in practical SLAM

One of the issues experienced in the practical implementation of SLAM is determining the pose of the robot with sufficient accuracy, in the presence of noise. This is a necessary to be able to make sound navigation decisions autonomously. This means the robot's initial location, actions and subsequent locations should be fairly predictable to be able to plan any kind of path to achieve a given goal. In other words the task of placing the vehicle pose accurately in state space consistently, and without sudden jumps between transitions (usually due to noisy sensor data), is very important for path planning and navigation. This is where the Kalman filter and later particle filters have been used extensively [33].

2.3.1. Kalman filter

The Kalman filter is an algorithm that provides an efficient computational (recursive) means to estimate the state of a process, in a way that minimizes the mean of the squared error [34]. One of the major benefits is that it, being a recursive algorithm, does not require knowledge of all prior values of a particular state; this saves memory, although it can also be a disadvantage, as information is lost. The algorithm consists of two steps, a prediction/estimation and a correction/update step and uses the probability distributions to update the state. This filter intends to solve the problem of optimally estimating the value of a state $\boldsymbol{x}$ in the presence of process noise $\boldsymbol{v}$ and measurement noise $\boldsymbol{w}$. Equations (2.1) and (2. 2) show the equations governing a systems behaviour, taking into account system and measurement noise.

$$\boldsymbol{x}(t+1) = \boldsymbol{F}(t)\boldsymbol{x}(t) + \boldsymbol{G}(t)\boldsymbol{u}(t) + \boldsymbol{v}(t) \qquad (2.1)$$

Where $\boldsymbol{F}$ is the system function, G is the input function and u the control input into the system

$$\boldsymbol{z}(t) = \boldsymbol{H}(t)\boldsymbol{x}(t) + \boldsymbol{w}(t) \quad (2.2)$$

Here z is the state measurement and H is the measurement function

The Kalman filter explained in [35] uses the two-step process shown below. These steps are known as prediction and update, and are recursively used to make a best estimate of the state value given the current measurement.

Prediction

State Prediction: estimate of $\hat{\boldsymbol{x}}(t+1)$ i.e. future state value at time *t*

$$\hat{\boldsymbol{x}}(t+1|t) = \boldsymbol{F}(t)\boldsymbol{x}(t|t) + \boldsymbol{G}(t)\boldsymbol{u}(t) \quad (2.3)$$

State Covariance Prediction (predict the uncertainty of the future state)

$$\boldsymbol{P}(t+1|t) = \boldsymbol{F}(t)\,\boldsymbol{P}(t|t)\,\boldsymbol{F}(t)' + \boldsymbol{Q}(t) \quad (2.4)$$

Where $P(t|t)$ is the prior probability distribution and $Q(t)$ is the variance

Measurement Prediction: estimate of measurement $\hat{\boldsymbol{z}}(t+1)$ at time *t*

$$\hat{\boldsymbol{z}}(t+1|t) = \boldsymbol{H}(t+1)\hat{\boldsymbol{x}}(t+1|t) \quad (2.5)$$

Measurement Covariance Prediction (predict the uncertainty of the future measurement)

$$\boldsymbol{S}(t+1) = \boldsymbol{H}(t+1)\boldsymbol{P}(t+1|t)\boldsymbol{H}(t+1)' + \boldsymbol{R}(t+1) \quad (2.6)$$

Filter Gain Computation (computes the importance of the predictions)

$$\boldsymbol{W}(t+1) = \boldsymbol{P}(t+1|t)\boldsymbol{H}(t+1)'\boldsymbol{S}(t+1)^{-1} \quad (2.7)$$

Update

Measurement Update

$$\boldsymbol{v}(t+1) = \boldsymbol{z}(t+1) - \hat{\boldsymbol{z}}(t+1|t) \quad (2.8)$$

$\hat{z}(t+1)$ is the predicted measurement and $z(t+1)$ is the actual measurement at $(t+1)$.

State Update

$$\hat{\boldsymbol{x}}(t+1|t+1) = \hat{\boldsymbol{x}}(t+1|t) + \boldsymbol{W}(t+1)\boldsymbol{v}(t+1) \quad (2.9)$$

This is the best estimate of the true state $x(t+1)$ at time $(t+1)$ given the measurement $z(t+1)$'.

State Covariance Update

$$\boldsymbol{P}(t+1|t+1) = \boldsymbol{P}(t+1|t) - \boldsymbol{W}(t+1)\boldsymbol{S}(t+1)\boldsymbol{W}(t+1)' \quad (2.10)$$

This updates our variance estimate based on the measurement $z(t+1)$ and is known as the posterior probability distribution

Where $\boldsymbol{P}(0)$, $\boldsymbol{Q}(0)$ and $\boldsymbol{R}(0)$ are chosen by intuition. $\boldsymbol{Q}$ and $\boldsymbol{R}$ are state and measurement noise covariances respectively. $\boldsymbol{W}$ is called the Kalman gain and is computed using the state covariance estimate [35]

2.3.2. Particle filter

An alternative to the Kalman filter is the particle filter (also known as a Monte Carlo simulation). This filter generates many points in state space, each representing a possible state value. A measurement is then taken to evaluate the likelihood of a particular particle being the true value of the state. Each particle is then assigned an importance weight according to its likelihood. The particles are then randomly re-sampled (selected as the next generation) with a probability equivalent to each particles' importance weight. Particle filters have an advantage over Kalman filters in that they can be multi modal (as a result they can model complex distributions) at the cost of accuracy due to their discrete nature [36]. The operating principle of particle filters is similar to the Kalman filter, with the difference that particle filters approximate the probability distribution using particle weights, as opposed to a continuous Gaussian distribution. Their operation is shown below.

Initialisation:

Generate a set of *n* particles representing the robot state $(\boldsymbol{x})$ probability density function (pdf) at time $\boldsymbol{t}$

$$\boldsymbol{p}(x_{i\to n})_t = x(rand) \quad (2.11)$$

Sampling:

From $\boldsymbol{p}(x_i)_t$ particles choose (with replacement) $\boldsymbol{N}$ particles with probability proportional to weights $\boldsymbol{w}$ (which can be initialised to equal probability/importance values)

$$\left\{\boldsymbol{x}^{i*}(t)\right\}_{i=1}^{N} \tag{2.12}$$

Prediction:

Using the system model, we predict the particle states one time step ahead for all particles

$$\widehat{\boldsymbol{x}}_{t+1}^{i*} = \boldsymbol{F}_t(\boldsymbol{x}, \boldsymbol{u}, t)\boldsymbol{x}_t^{i*} + \boldsymbol{G}_{\boldsymbol{t}}(\boldsymbol{x}, \boldsymbol{u}, t)\boldsymbol{u}_{\boldsymbol{t}} + \boldsymbol{v}_t \tag{2.13}$$

Where $\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{t})$ is the system model and $\boldsymbol{v}$ is an independent sample drawn from the system noise pdf. **G x** and **u** are defined. This gives us the particles representing the approximate pdf $\boldsymbol{p}(\boldsymbol{x}_{i\to n})_{t+1}$ of $\boldsymbol{x}$ at time $(\boldsymbol{t}+1)$, equivalent to (2.11) in the subsequent time step.

Update:

After a measurement of $\boldsymbol{z}$ at time $(t+1)$ is observed, the weights $\boldsymbol{w}$ are updated according to the likelihood of observing $\boldsymbol{z}$ given $\boldsymbol{x}^*$ at $(t+1)$ as shown (2.14) in below.

$$\widehat{w}_{t+1}^{i*} = p(z_{t+1}|x_{t+1}^{i*}) \tag{2.14}$$

These weights are then normalised according to (2.15)

$$w_{t+1}^{i*} = \widehat{w}_{t+1}^{i*} / \sum_{j=1}^{N} \widehat{w}_{t+1}^{j*} \tag{2.15}$$

2.4. Clustering and its Application

The RFID localisation in this dissertation makes use of data clustering. Clustering is used in many applications where one has a large amount of data, most likely consisting of similar/repeated data or observations. For storage purposes and to speed up computation, it is desirable to group similar data/observations together and represent the group in terms of a cluster prototype/centre.

Clustering is the process of finding a set of groups of similar objects within a data set, whilst keeping dissimilar objects in different groups [37] [38]. There are two types of clustering algorithms, hierarchical and partitioning [39]. Partitioning methods split the data into a predefined number of groups/clusters, and therefore requires some domain knowledge [40]. Hierarcical methods iteratively compute and join the points closest to each other (in a tree-like fashion) until a certain termination condition is met, the a general method of determinating this condition has proven difficult [40]. Although there are many distinct clustering approaches within these two categories, there are only a few which are most prevalent in the literature and as such were the focus. These are single linkage clustering (also known as nearest neigbhor) [41], centroid based clustering (such as kmeans) and density based clustering. Recent efforts have focused on improving the performance of clustering algorithms, largely due to the need to apply these algorithms to very large data sets (such as big data), such as in [42], [43] and [44] (CLARANS).

2.4.1. Hierarchical Clustering

The single link (hierarchical) clustering is one of the oldest methods of cluster analysis [41]. Single link clustering starts with (in the case of agglomerative/bottom-up clustering) considering all the points in the data as clusters; then for each cluster, the nearest neighboring cluster is found and the two clusters are then combined into one.Conversly divisive clustering would start by assuming all the point belong to one cluster, then the cluster is iteratively separated accoding to the furthest points. The metric used to determine the nearest neighbour can be any distance metric, such as Euclidian distance $||a\text{-}b||_2$, squared Euclidean distance $||a - b||_2^2$), manhattan distance $||a\text{-}b||_1$ etc. The point from which this distance metric is computed is what distinguishes the different kinds of single link agglomorative clustering, from complete linkage (uses the points in the clusters which are furthest apart) to minimum linkage (uses the points in the clusters which are closest to the other cluster) or even average linkage (which uses distance between the

average of all the points within each cluster). An example of a single link hierarchical clustering algorithm is illustrated in Figure 2.1.

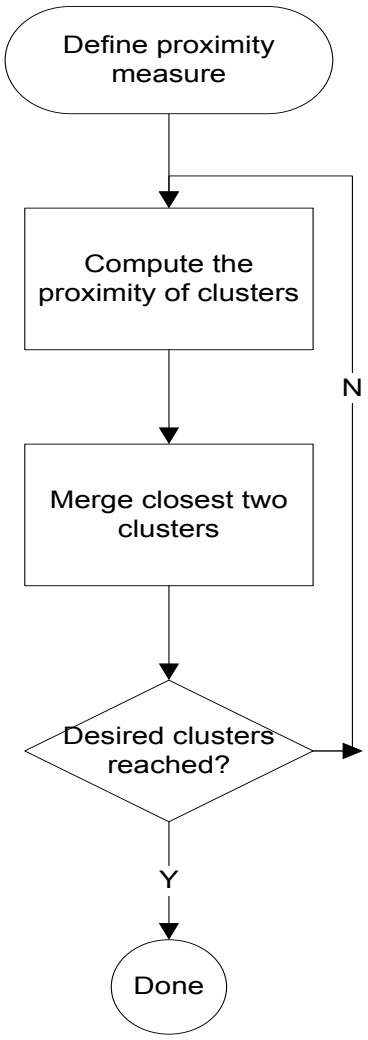

Figure 2.1: Hierarchical Clustering Algorithm

This method is good for finding clusters of arbitrary shapes, but it has a well known disadvantage called the chaining effect [41]. This is a phenomenon in which the algorithm may merge points close to the current cluster even though these may be outliers. In some instances there may be a string of outliers between two distinct clusters, single link clustering may result merging of these clusters, this makes hierarchical clustering sensitive to noise. Another issue with hierarchical clustering is the space and time complexity of the algorithm limits the practical size of data set which can be processed [45]. In this dissertation, three different kinds of clustering algorithms were tested (k-means, k-medoids, hierarchical).

2.4.2. Centroid Based Clustering

Centroid based clustering, and in particular K-means is one of the most widely used of the clustering algorithms [45] [38]. This may be due to is programming simplicity and its computational efficiency, as it requires linear time [46] to compute. Centroid based clustering methods start by (depending on the type of initialisation) creating a number of centroids, which may (in the case of k-medoids) or may not (k-means) be points in the data set. The points closest to each centroids are then assigned to the corresponding centroids' cluster. As an example, assuming we have a set of observations $O = [s_1, s_2, s_3 \dots s_n]$, where s_i is a single observation matrix. K-means is implemented by first choosing K (the number of desired clusters), then initialising K centroids, represented by $C = [c_1, c_2 \dots c_k]$ where k<n, with centroids $M = [m_1, m_2 \dots m_k]$ (i.e m_i is the mean of the observations in cluster c_i). There are a number of ways to initialise the centroids; one is by simply selecting random data points from O, and using these as the initial choice of centroids. Another way would be to draw point's uniformaly at random from O [47], or randomly define points within the hypervolume containing the set O [48]. The last method consists of clustering a random sub-sample of O, and then using these cluster centroids as the starting centroids. Next the observations s_i are assigned to a cluster, such that the distance from $\boldsymbol{s_i}$ to the cluster centroid $\mathbf{m_k}$ is minimised.This can be expressed as the function in equation (2.16). While there are many options for computing this distance, Euclidian distance is the most commonly used in practice and research [49].

$$\min_{arg} \sum_{k=1}^{K} \sum_{s_i \in C_k} ||s_i - m_k||^2 \qquad (2.16)$$

The new centroids are calculated using equation 2.17 below, which sums the observations assigned to the current cluster c_k and divides by their total to get the mean/centroid.

$$m_k = \frac{1}{C_k} \sum_{s_i \in C_k}^{n} s_i \qquad (2.17)$$

This process of assigning observations to the nearest cluster and re-computing the centroid is repeated until the centroids no longer change (i.e. have converged). This algorithm is shown in Figure 2.2.

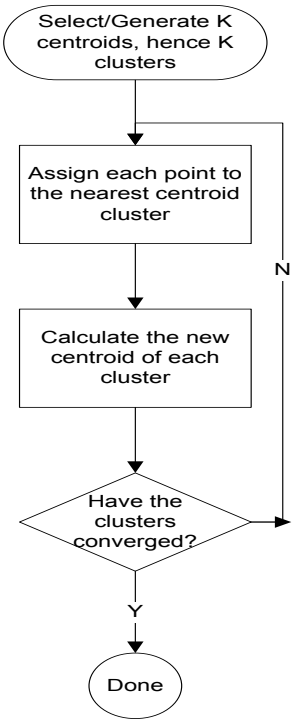

Figure 2.2: K-means clustering algorithm

Although k-means is simple and widely used, it has some practical problems [38]. The first of which is it being sensitive to the initial choice of centroids, and the second being unavailability of a general method of choosing these centroids. A bad choice of initial centroids can lead to sub-optimal clustering. Additionally when dealing with unknown data, it is difficult to know the number of clusters present in the data.

There are a few ways to compensate for the initial centroid selection problem. One option is to use hierarchical clustering to a level of the desired clusters and then using these centroids as the starting point for k-means. This option works well but is only practical if the number of data points is small (a few hundred to a few thousand) and **K** is relatively small compared to the number of data points [45].

Another approach to choosing centroids is outlined in [45]. It consists of first choosing a random point, then choosing your second centroid to be a point furthest away from the current centroids, then choosing your third centroid to be the furthest point away from those centroids, and so on.

This approach however also has a disadvantage when dealing with lots of data, as calculating the furthest point becomes computationally expensive [45]. Another issue with k-means is the possible formation of empty clusters (a scenario occurring when no points

have been assigned to a particular centroid). Typical ways to deal with this issue are to reinitialise the empty cluster centroid to be the furthest point from all the current centroids, or split a cluster with the most spread out points. Both these measures will reduce the Sum of Squared Errors (SSE), which is a measure of the distance between points and their centroids. Although k-means is simple and effective, it is not suitable for all data types, such as non-globular clusters or clusters of varying sizes and densities [45]. In these cases it can fail to identify natural clusters.

K-medoids is another clustering algorithm very similar to k-means; with the exception that the centroids must be one of the data points (as opposed to the mean of the points in the given cluster) close to the mean [38].

2.4.3. Density Based Clustering

Density based clustering is similar to single link clustering; with the additional constraint that for a cluster to be formed a density criteria must be met. This means that even though there may be a neighboring point close by, these points can only be merged into a cluster if there are a certain more than a minimum number of points within a given radius. Wishart's [50] attempt to mitigate the chaining phenomenon in hierarchical clustering resulted in most likely the first density based clustering approach [37]. Density based clustering, like single link clustering, is able to cluster data of arbitrary shape, but unlike the latter, is less susceptible to the chaining phenomenon, due to the additional density requirement for merging clusters (this may not be the case if clusters are very close to each other). There are variations in the kinds of density based clustering methods, these vary in how the density is estimated as well as how the notion of connectivity is defined [37]. The overall principles remain closely related to the method explained above.

The most popular density based clustering is density based spatial clustering of applications with noise (DBSCAN) [40]. Although in some literature it is stated that CLARANS is more well known [40], in the same work DBSCAN was proven to outperform CLARANS when dealing with point data. As such our review will concentrate on DBSCAN, since we will be clustering point data. DBSCAN is designed to cluster spatial data in the presense of noise [40] and claims to be scaleable to large data sets [37]. This algorithim uses the concept of an epsilon neighbourhood ($N\varepsilon$), which is the number of points within a radius ε, along with the notion of density-reachability. This is defined as a point p being density reachable from point q if:

1. the neighbourhood $N\varepsilon(p)$ of p includes q
2. either $N\varepsilon(p) \geq$ minimum neighbhood points $\min(N\varepsilon)$ or $N\varepsilon(Q) \geq$ minimum neighbhood points $\min(N\varepsilon)$ [40]

The algorithm starts by choosing any point in the data set, it then groups into one cluster all the points that are density-reachable from the chosen point. If there are not points which are density-reachable from the chosen point, it is regarded as noise. Next another point which is has not been clustered (or classified as noise) is chosen, and the process is repeated until the complete data set has been clustered. One issue with density-based clustering is algorithms is they have difficulty detecting overlapping clusters as well as underlying cluster forms such as gausian clusters, as the border of these cluster becomes arbitrary depending on the chosen value of the density. Another issue is that it cannot cluster data sets with a large variation in density [51].

From the literature it is apparent that there is no general clustering algorithm which is guaranteed to give good clusters for any given data set, additionally having some prior knowledge about possible cluster locations, densities, sizes or some other information about the data drastically improves the effectiveness of each algorithm [52]. Due to it simplicity and ease of implementation, it was decided to use one of the centroid based clustering algorithms. Three centroid clustering algorithms are compared in this work for use in localisation.

2.5. Path Planning and Navigation

Once a map of a robot's environment has been generated, it may be desirable to have the robot be able to autonomously navigate to a desired goal, either selected by a human operator or identified by the robot itself as an area of interest or for further exploration. To be able to navigate to a desired goal, a robot needs to plan a path or determine which sequence of actions will get it to the desired location. It also needs to be able to deal with the non-deterministic nature of the natural environment, which may give rise to an unexpected result, such as ending up in a different state to that which was predicted given the executed action. The robot path planning problem has received considerable attention in the last ten years [53].

There are a great number of path planning algorithims available, from probabilistic planners [53], heuristic [54], randomised (such as random trees [55]) and shortest path (such as A*, value iteration or Dijkstra [56]), to name a few. For our purposes, we were primarily interested in the shortest path algorithms, since this work was limited to a static environment with no obstacles. The shortest path algorithms provided the simplest potential solution for our requirements and therefore form the focus of our review.

2.5.1. Dijkstra Algorithm

The dijkstra algorithm works by marking a starting point and a goal on the map. Each state/node/intersection in the map is given the highest possible distance value. Starting from the starting state move to the closest unexplored/unlabeled state, make this the current state and update the distance (which is equal to the distance to the current state + distance to update state) to all the states directly connected to this current state (provided the distance is less than the current value). Then move to the closest unexplored state (from the starting point) and repeat the process until the goal is reached [57] [58]. Since the dijkstra algorithm is not directed towards the goal, but rather spreads out from the the starting node, it may be slow in the instance where the starting point is in the centre of the map and the goal is somewhere on the outer edge of the map.

2.5.2. A* Algorithm

A* is similar to the abovementioned dikjstra algorithm, but has the advantage of expanding the nodes in the direction of the goal. This reduces computation and as a result A* is faster than dikjstra, in fact A* is optimal in finding the shortest path under certain conditions [59]. To find the actual path to the goal, the algorithm should keep track of each node's predecessor.

2.5.3. Markov Decision Processes

When a path planning algorithm is combined with a set of possible robot actions, it can allow a vehicle to autonomously navigate to the desired goal. When navigating, a robot executes actions in order to transition along the nodes (also called states) in the desired/planned path to its destination. Each node in this chain represents a possible robot state or pose. This pose can consist of any useful (from a navigation perspective) vehicle information such as orientation, position, speed, acceleration etc.

In the real world, the outcomes of each action are not deterministic, and as such may rather result in the desired outcome with a certain probability or some other outcome with some other probability. This system that transitions from one state to another through a set of actions is known as a Markov Chain [60]. If the transitions from one state to the next do not depend on any history, then this process is known as a Markov decision process (MDP). This pose/state information, together with the actions must be used by a descision making algorithim to navigate to the goal [61]. This problem can be reduced to the following question: given a set of states, each with a given reward for being in the given state, what actions should we take at any given state to maximise the the current reward/utility, bearing in mind the probabilistic nature of the state transitions? The map of of these actions to states is called a policy [62]. Therefore the goal is to find the policy which maximises the expected reward/utility [63]. While the are a number of approaches in the literature to solving this problem, such as brute force (simply evaluating every possible policy) or direct policy search [64], we will focus only on value function approaches, since almost all reinforcement learning algorithms are based on estimating value functions [65].

In practice (where we do not know the fully MDP) there are two categories of value function approaches to solving this optimization problem. The first is category are the Monte Carlo methods approach, which uses random initial state action pairs s_i and a_j to asses the value of the given state action pair (s_i, a_j). This value is the reward received if action a_j is first executed from state s_i after which a greedy algorithm such as ε-greedy or dijkstra). These state action values are averaged over time, giving a progressively better estimate of action values for given states. One disadvantage of monte carlo methods is that they spend time evaluating sub-optimal policies, and use a long path to update only the initial state action pair.

The second value function approaches are the temporal difference approaches

In the case where the full MDP is known, we can use the simplest two (excluding modifications) algorithms commonly used for determining an optimal policy for decision problems, namely policy iteration (and modified versions thereof) and value iteration [63].

2.5.4. Value Iteration

Value iteration works by initialising the goal state with a reward value, then, using the state transition probability matrix $\mathbf{T_m}$, each state is assigned a utility value based on the surrounding state utility values. The utility value of each state is computed as shown in equation (2.18).

$$U_{t+1}(s_i) = U_t(s_i) + max_a \sum_j^N P(S_i|S_j, a) \times U(s_j) \qquad (2.18)$$

The equation above is executed for each iteration of the algorithm for all the states, and reads as follows; the future utility of state $\boldsymbol{S_i}$ is equal to its current utility plus the maximum across all (allowable) actions of the sum of the probability of reaching state $\boldsymbol{S_i}$ from state $\boldsymbol{S_j}$ given action $\boldsymbol{a}$, multiplied by the utility of state $\boldsymbol{S_j}$.

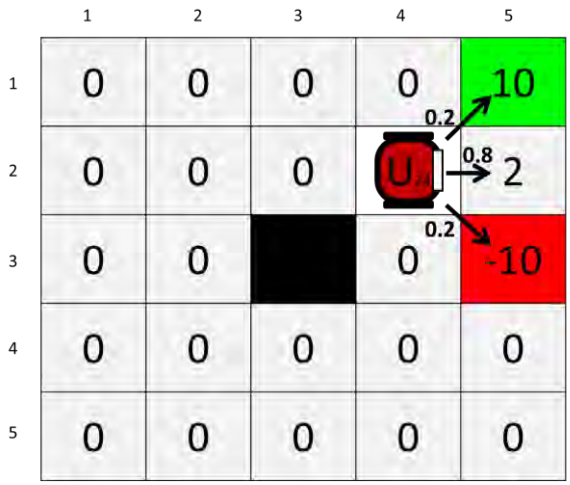

Figure 2.3: Value Iteration Algorithm at Work for a Forward action

An example of this in action is shown in Figure 2.3 for the forward action (each action has its own utility table such as the one in Figure 2.3), which has a 0.8 probability of moving forward, and a 0.2 probability of going left or right. Alongside is shown the calculation of *equation 16* assuming the current utility value of $U_{2,4}$ is 0. This process is repeated for all states until the change in state utility values are less than some threshold $\boldsymbol{\varepsilon}$ usually some small value.

After all state utility values have been computed, the optimal action policy is then worked out based on choosing the action most likely to lead to the maximum utility at each state. Executing this action policy should lead the robot to the desired goal.

2.5.5. Policy Iteration

It has been observated that the policy becomes optimal long before the utility estimates have converged (i.e. change in utility is less than ε) to their correct values [63]. Policy iteration takes advantage of this by focusing, not on the convergence of the utilty values, but rather the convergence of the policy. The algorithm works as follows:

1. Initialie a random policy, by choosing random actions for each state
2. Compute each states value given the current policy
3. Given the above state values, select the best action for each state (hence creating a new policy)
4. Make this the current policy
5. If there is a change in the policy, repeat from step 2
6. End

Policy iteration offers the advantage of converging faster in some cases than value iteration, because its convergence is based on the policy rather than utility values [66], However the policy evaluation step may itself be a protracted iterative computation [65].

In our case the robot builds a state transition matrix autonomously, therefore we have the complete MDP for the navigation problem; hence either of the simpler approaches (value iteration or policy iteration) is suitable. It was decided that value iteration was to be used in implementing the autonomous navigation in this dissertation, the convergence time between the two algorithms are negligible in our context, since we can choose the number of state/clusters.

3. Simulation Setup

To test the feasibility of using RFID tags for localisation and navigation, a simulator was built. The simulator would model the behaviour of randomly distributed tags in an environment. The density of the tags can be varied until an optimum tag density is found. The robot's motion would also be simulated in this environment. This would allow the evaluation of the accuracy of the localisation achieved, the effectiveness of clustering in its various forms (k-mean, k-medoids and hierarchical) and the results of navigation.

3.1. Modelling the Simulator

All the simulations were written in the Matlab programming language, and executed in a Matlab R2011b environment. The RFID reader used was the Alien ALR9900 and the antenna was the ALR8611-AC. A sensor model was derived based on experimental measurements from Forster [9] of the tags' Received Signal Strength (RSS) in various orientations and distances. Table 3.1 below shows some of these resulting RSS values.

Table 3.1: Experimental Data from [9] Showing average RSS values at different distances from the antenna

Distance (m)	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8	2.0
RSS(Units)	6500	6000	4500	2800	2500	1800	1500	1200	1000	600

3.1.1. Modelling Sensor Response

To model the RFID tag response from the reader, we utilised the experimental measurements from Table 3.1, these were fitted first with the polynomial function (*polyfit*) in Matlab [67] [ref]. The results in Figure 3.1 below show the curve fitting for polynomials of different orders. The 6^{th} order polynomial over-fits the data, whereas the 3^{rd} and 5^{th} order increases the RSS over large distances, which we know to be impossible as RSS values from antennas decreases monotonically with distance. It was decided that the best polynomial fit was the 4^{th} order polynomial, due to the fact that it not only fits the all points closely without visible overfiting, but it also plateaus at the correct value close to the origin. In addition, the sudden drop off after 2 m models the fact that RFID communication begins to be erratic after 2 m (sometime tags will respond, but sometimes they will not, with an increasing probability of not responding the further the tag is beyond the 2 m mark). This choice is to be compared with another curve fitting solution.

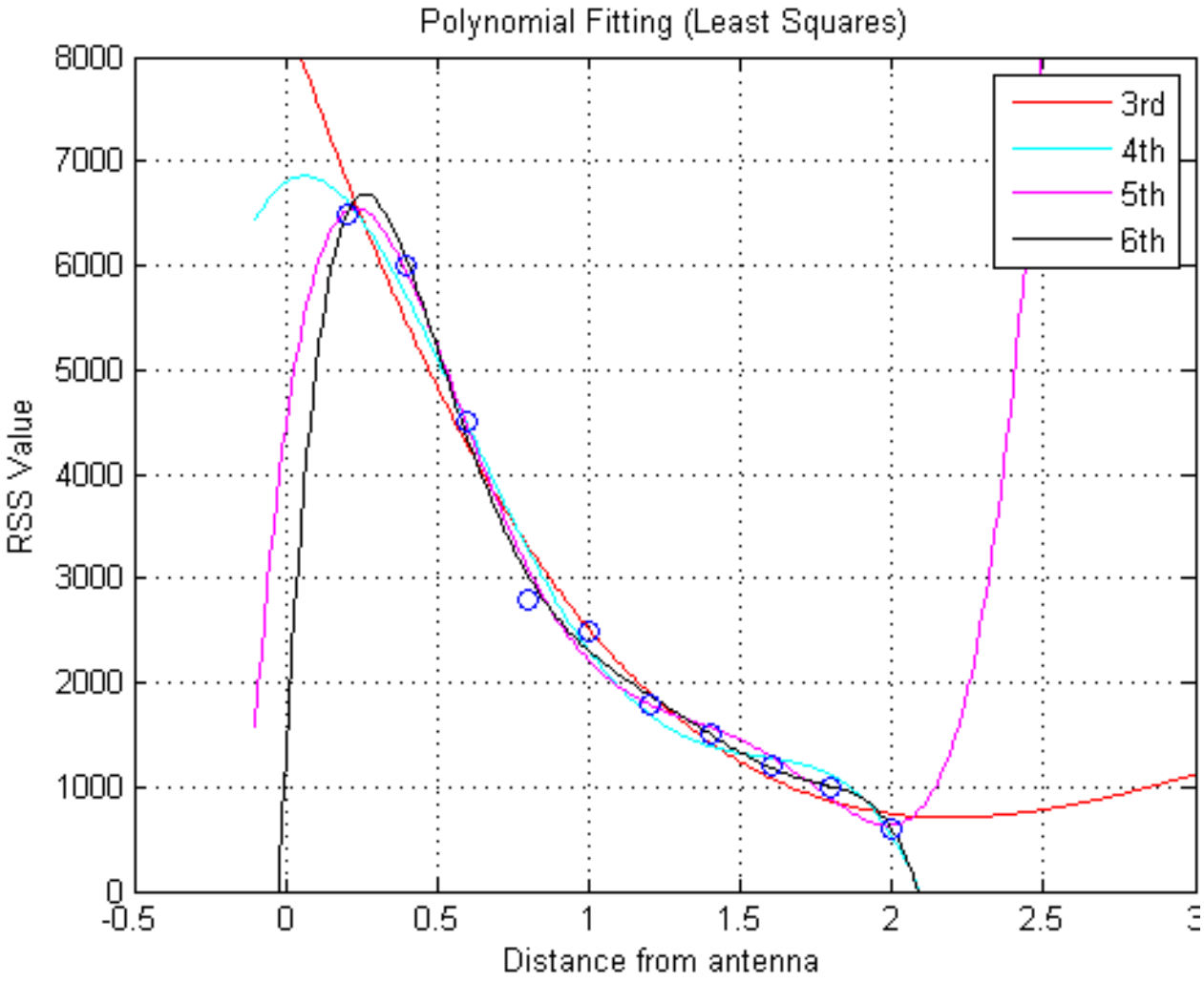

Figure 3.1: Polynomial fitting showing 3rd, 4th, 5th and 6th degree polynomials fitting experimental data

We then used exponential functions to fit the data, based on the exponentially decreasing energy distribution of an antenna at increasing distances. Since Matlab has no built in exponential fitting function, we decided to make use of the matrix division operator "**A\B**" which gives the least squares solution to the system of equations **A*X = B**. First we created a matrix containing exponentials of different orders, and then used the least square solution finder built into Matlab to find the best combination of exponentials which would minimise the error. The exponential curves of orders 3 to 6 are shown in Figure 3.2. We can see from the curves that the 6th order suffers from over-fitting. The 4th diverges from the data points at the ends of the dataset, and as a result does not agree with antenna physics (the closer the tag, the higher the RSS must be), as the RSS does not

peak at 0 m. The 5th order also suffers from the same problem of not peaking at 0 m. The 6th order, suffers from over-fitting (seen between 0 m and 0.5 m), and was thus not considered any further. Therefore it was decided that the best exponential fit is the 3rd order exponential.

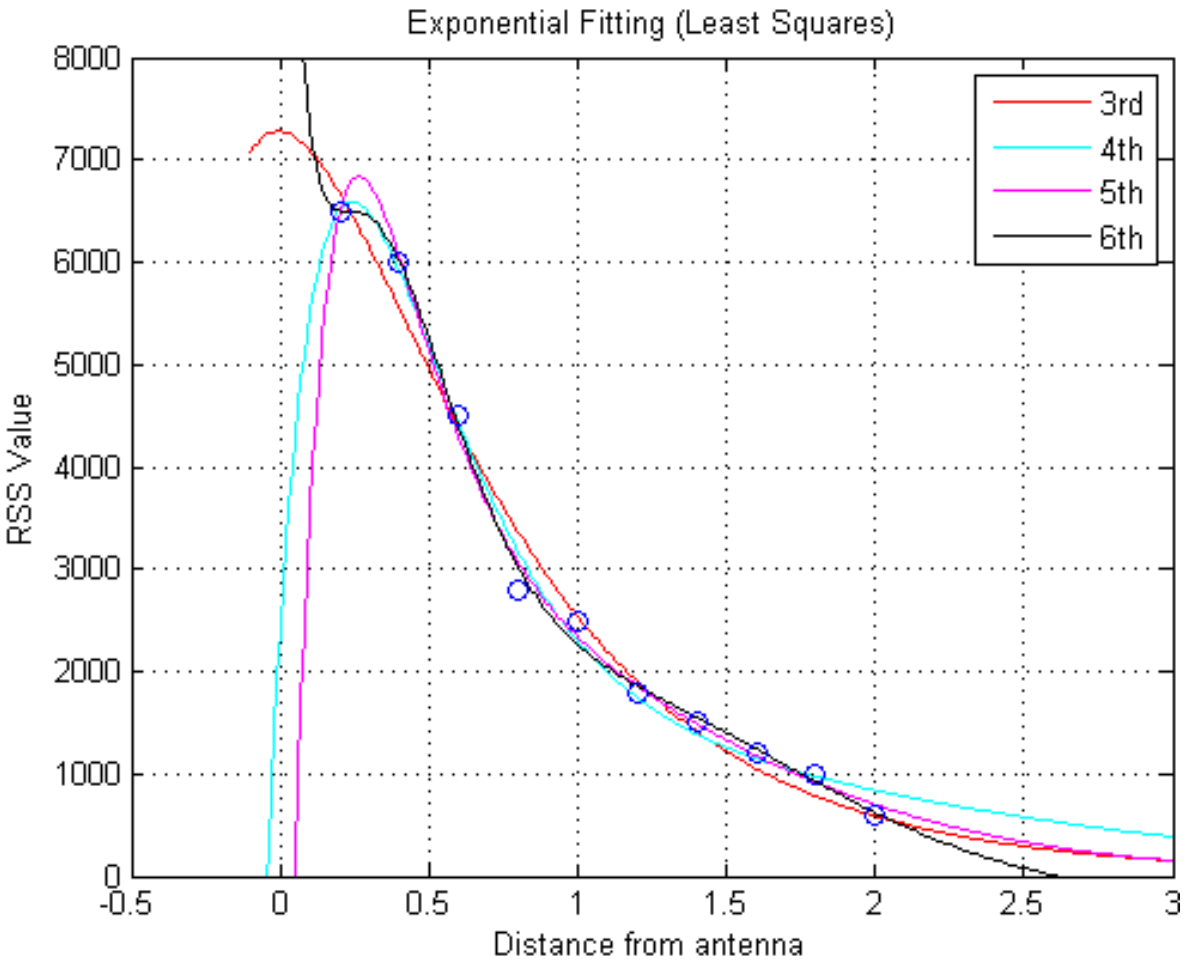

*Figure 3.2: Exponential fitting showing 3rd, 4th, 5th and 6th order exponentials vs distance (**m**)*

These two chosen fitting curves (polynomial and exponential) were compared for both minimising the error between the data-points and the fitted curve as well as each functions agreement with antenna physics. The error comparisons of the chosen curves are shown in Figure 3.3 below, for our test, we are only interested in modelling distances between 0.2m and 2 m, beyond which the RFID tags response is unreliable (erratic).

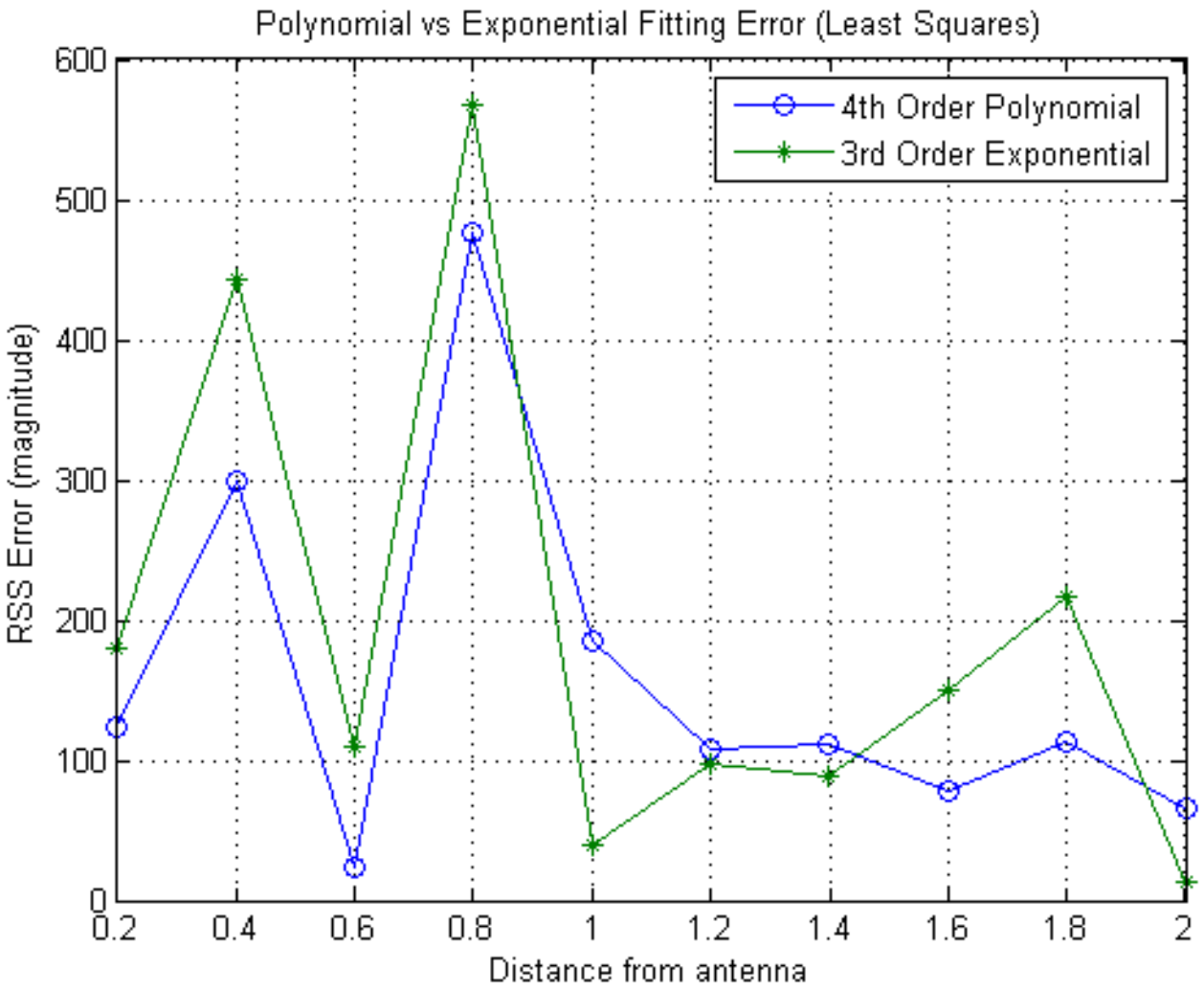

*Figure 3.3: Polynomial and exponential fitting error (magnitude) for RFID tag response vs distance in **m***

The accumulated error for polynomial and exponential fits was 1584 and 1907 RSS units respectively. The polynomial fit is better when considering only the total fitting error. But when comparing the two solutions taking into account all the factors, the issue experienced with using polynomial functions to fit this data was, polynomials are unbounded functions and as a result either tend to -∞ or + ∞. We also know that for physical systems the energy is bounded. This made polynomials unsuitable for modelling

the RFID sensor response. For these reasons it was decided to use the 3rd order exponential function to model the RFID sensor response. The coefficients of the exponential approximation are shown in Table 3.2.

Table 3.2: Exponential least squares approximation coefficients

1st Order Coefficient (c_1)	2nd Order Coefficient (c_2)	3rd Order Coefficient (c_3)
2235	17144	-12101

Thus the function for approximating the RSS value for a tag at distance ***d*** is given by:

$$RSS(d) = \left(c_1 e^{-d} + c_2 e^{-2d} + c_3 e^{-3d}\right) \quad (3.19)$$

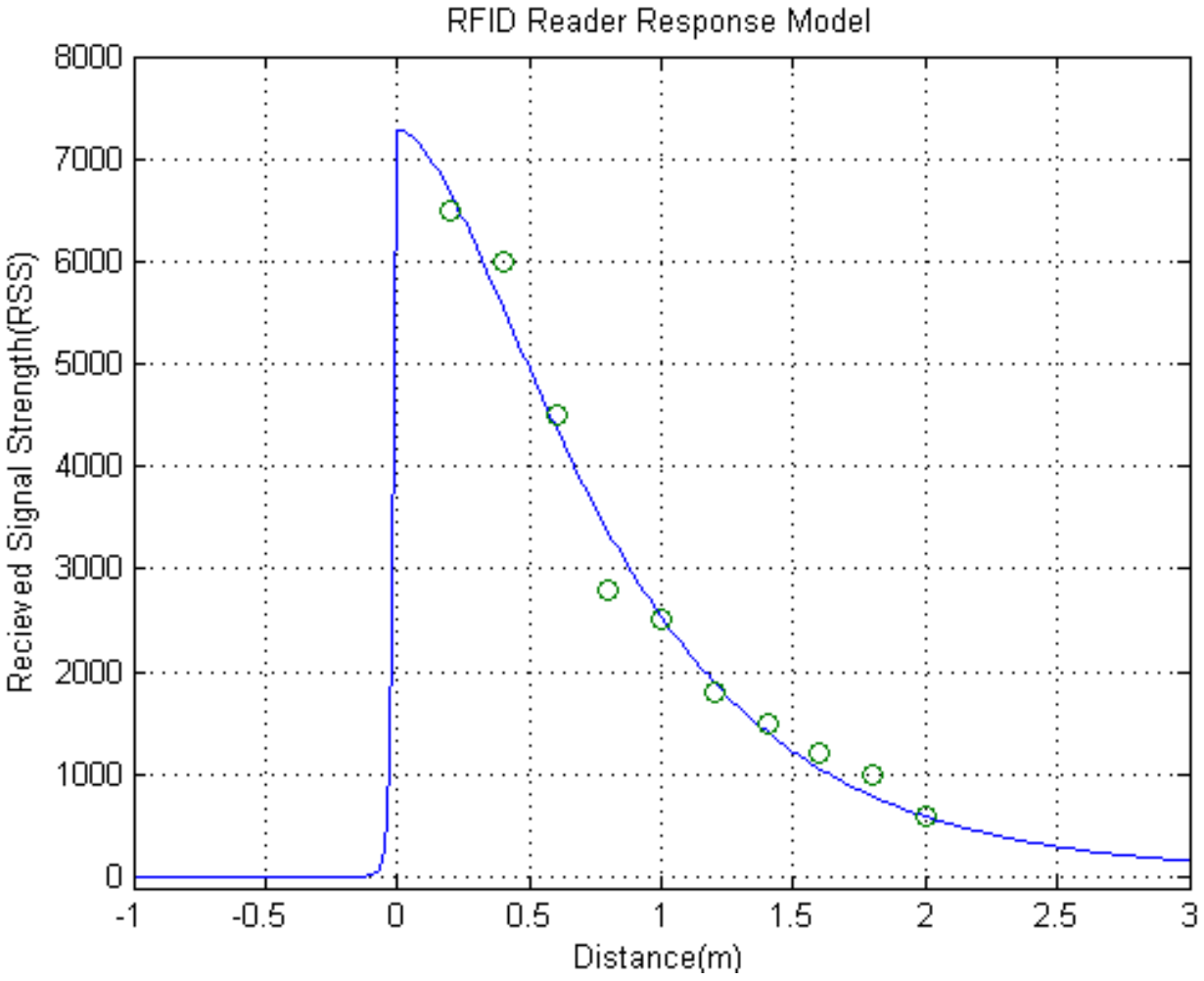

Figure 3.4: Final 2D tag RSS response curve using exponential approximation

Figure 3.4 shows the final 2D sensor model, derived from experiments using a 3rd order exponential function fit from (3.19). As seen from the graph there is a small chance that the RFID tag reader can read tags that are behind the antenna, this occurs if the tag is sufficiently close to the antenna (confirmed experimentally), hence the non-zero RSS for negative distances in Figure 3.4. To simulate this effect, equation (3.19) was mirrored and attenuated about the y-axis to give RSS values for negative distances (i.e. behind antenna). Two different tag types were tested; a square 3 cm by 3 cm tag and a rectangular 2 cm by 6 cm. Different tags gave different RSS values at similar distances, with the rectangular tags giving the highest RSS for a given distance. The shape of their response curves remained the same albeit scaled. The experimental results can be found in the CD accompanying this project.

The model was further extended to a 3 Dimensional reader response surface, allowing for the simulation of tag responses in a planar environment. This response was obtained by assuming the RSS value drops in accordance with the angle (***θ***, theta) of deviation from the normal angle(**0**$^{\circ}$ i.e. directly in front of the antenna). We can confirm this assumption by observing the antenna beam pattern from the antenna's datasheet [68], this beam pattern is shown in Figure 3.5,

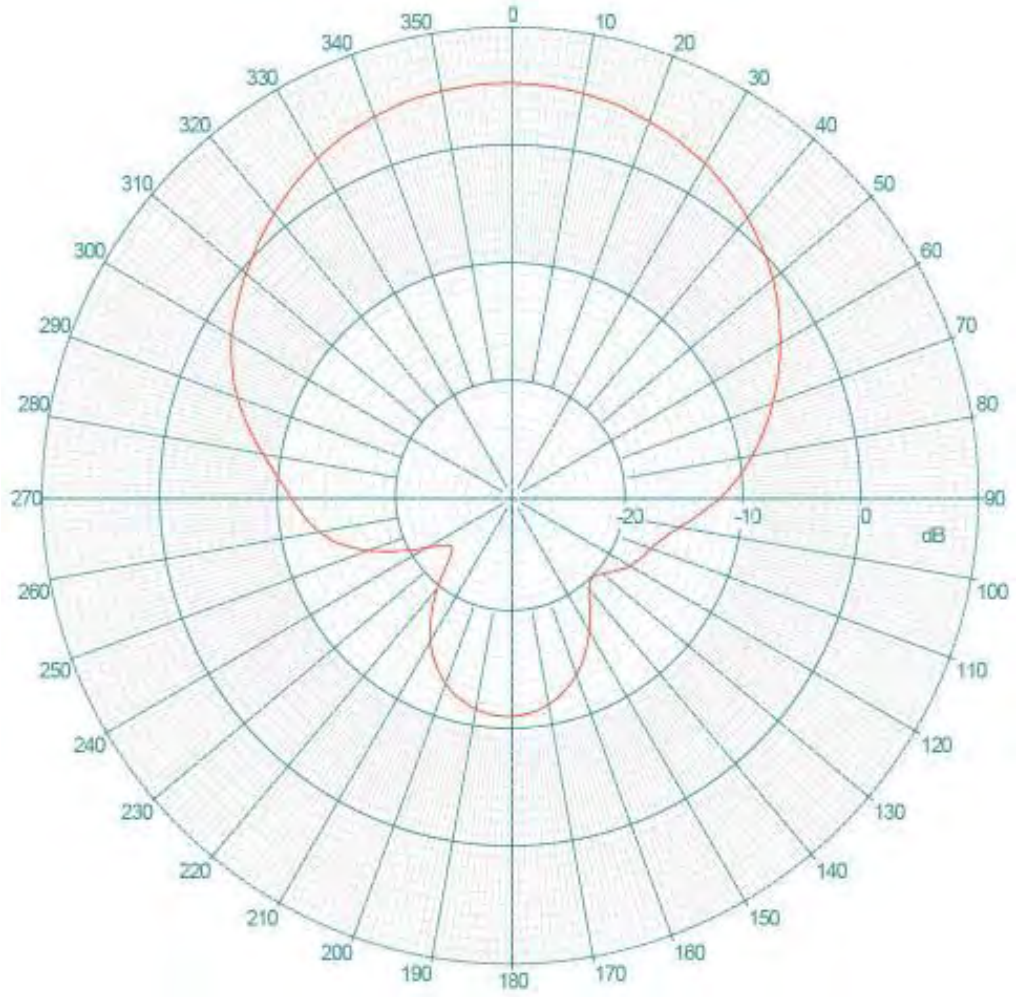

Figure 3.5: Alien Technology antenna ALR-8696-C beam pattern (from datasheet [68])

If we approximate the radiation pattern of the antenna with a centroid around a unit circle, then this function ***t(θ)*** can then be used to scale a version of the above tag response curve rotated about the origin, to give tag responses at various distances and angles from the antenna. ***t(θ)*** could be modelled as the intersection of the line ***t*** with the unit circle with an offset centre, as shown in Figure 3.6 below. The method for obtaining this function ***t*** is as follows:

We want the intersection between the line:

$$x(t) = tcos(\theta) \quad and \quad y(t) = tsin(\theta) \tag{3.20}$$

And the circle:

$$\left(x - \frac{1}{2}\right)^2 + y^2 = (\frac{1}{2})^2 \tag{3.21}$$

Substituting (3.20) into (3.21) we get:

$$t = 0 \quad or \quad t = \cos(\theta)$$

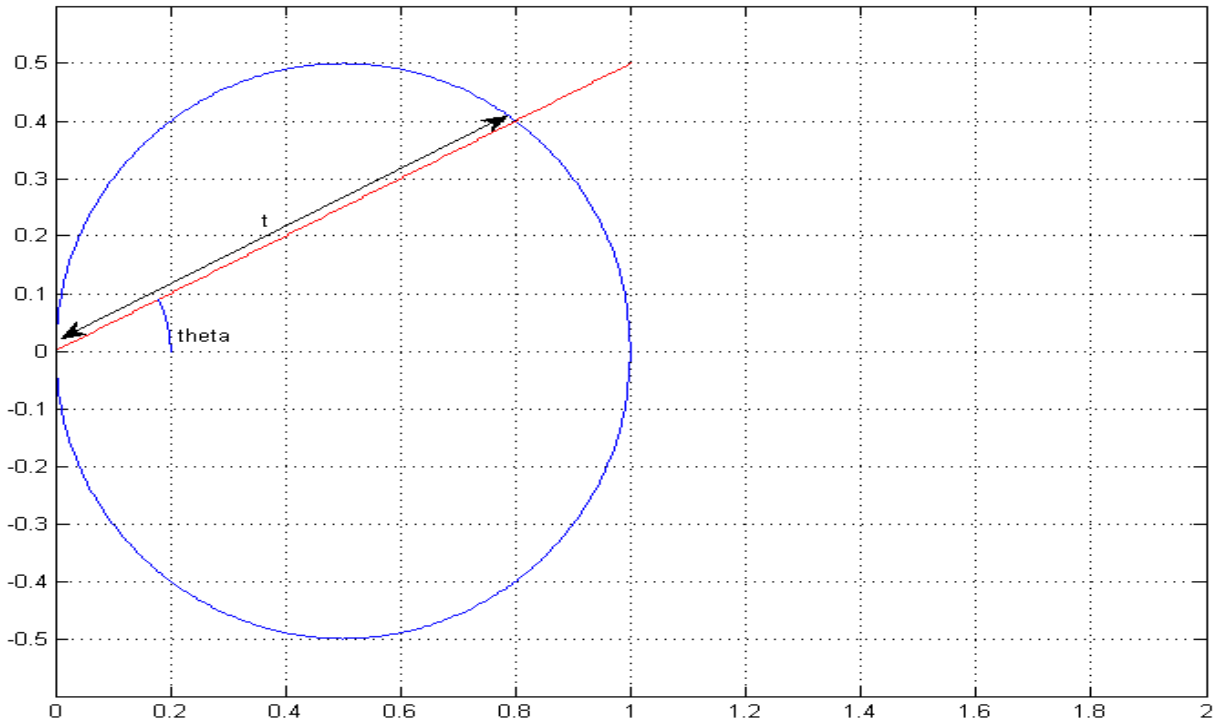

Figure 3.6: Modelling 3D response model by rotation of 2D model

The resulting equation gives us the RFID tag response RSS for a tag at distance ***d*** and angle ***θ***:

d>=0 $RSS(d,\theta) = \left(c_1 e^{-d} + c_2 e^{-2d} + c_3 e^{-3d}\right) \times |\cos(\theta)|$ *(3.22)*

d<0 $RSS(d,\theta) = \left(c_1 e^{-ad} + c_2 e^{-2ad} + c_3 e^{-3ad}\right) \times |\cos(\theta)|$ *(3.23)*

Where ***a*** is an attenuation factor of 50.

The resulting planar RSS response surface is shown below in Figure 3.7 and Figure 3.8 also showing the pioneer robot platform in a top down view. The resulting sensor model is in close agreement with the energy distribution of the radio signal shown in the antenna datasheet (Alien Technology: ALR-8696-C circular polarised antenna) [68].

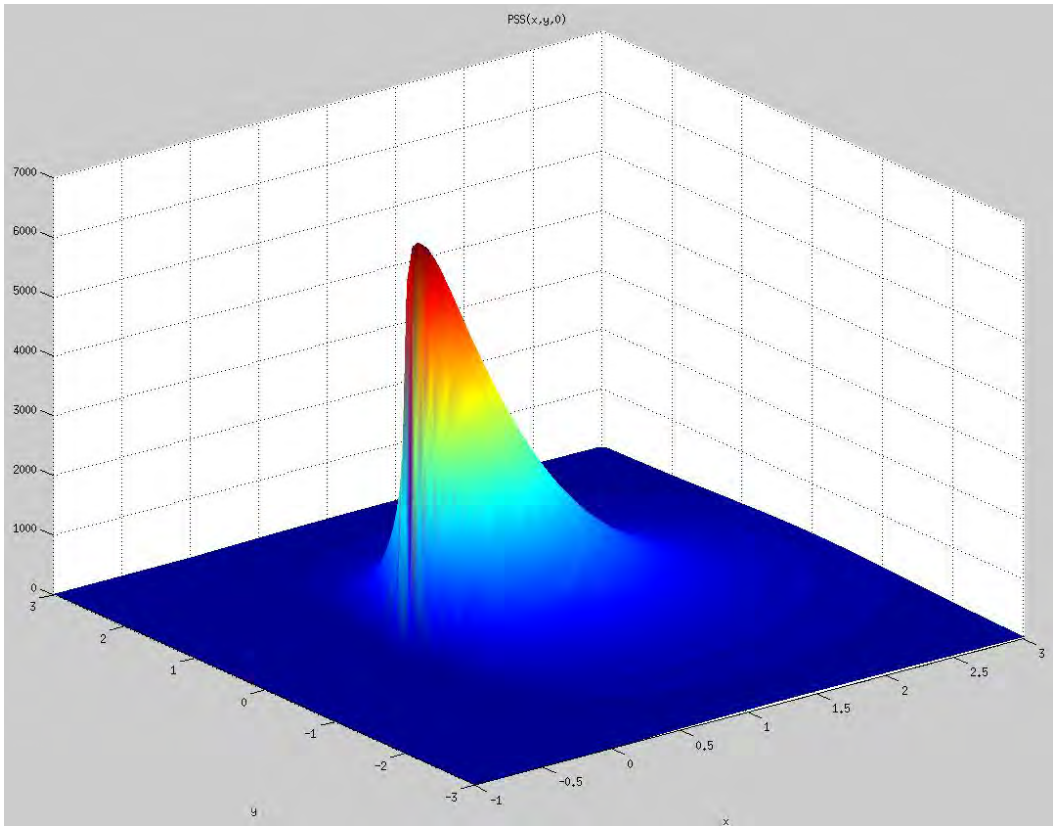

Figure 3.7: Three dimensional tag response when vehicle is at (0; 0) in the x-y plane

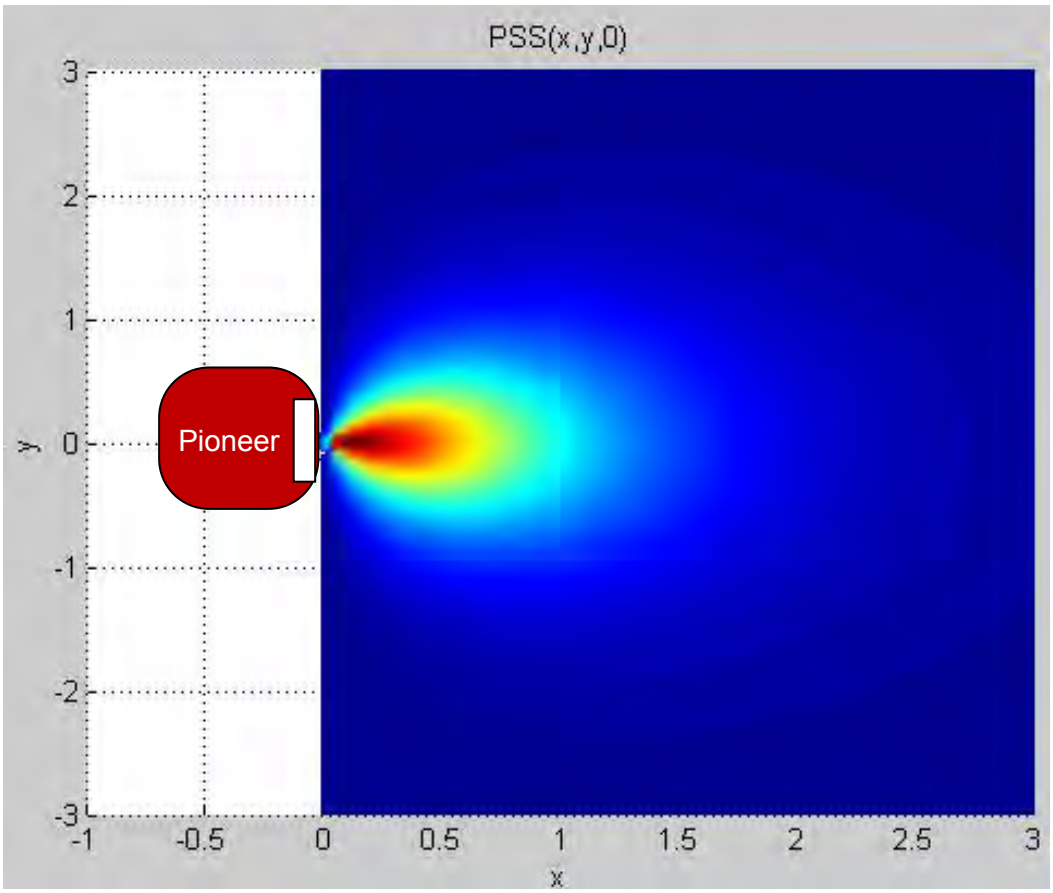

Figure 3.8: Top down view of 3D tag response when vehicle is at (0; 0) in the x-y plane

3.1.2. The Case for Distance and Orientation

In theory the case for judging distance and orientation can be made when the there are at least 3 visible tags, since there are no two points in state space (displacement and orientation) which would lead to identical RSS values. Any change in location or orientation will change the position of the tags relative to the robot in a unique way, and hence the RSS values of the snapshot will be sufficiently different for the algorithm to recognise that the robot is in a different state, provided the movement is significant (greater than 0.4 m).

3.2. Simulation Environment Design

The simulator was designed to model randomly distributed tag responses in a planar environment. The tags are represented by circles and the robot is represented by a blue cross. These tags are randomly distributed throughout the environment. The idea is to make the simulation as realistic as possible by ensuring that the robot motion and control commands are as close to the physical platform implementation as possible. This will not only ensure the accuracy of the simulation, but also simplify the implementation experiment stage of the project.

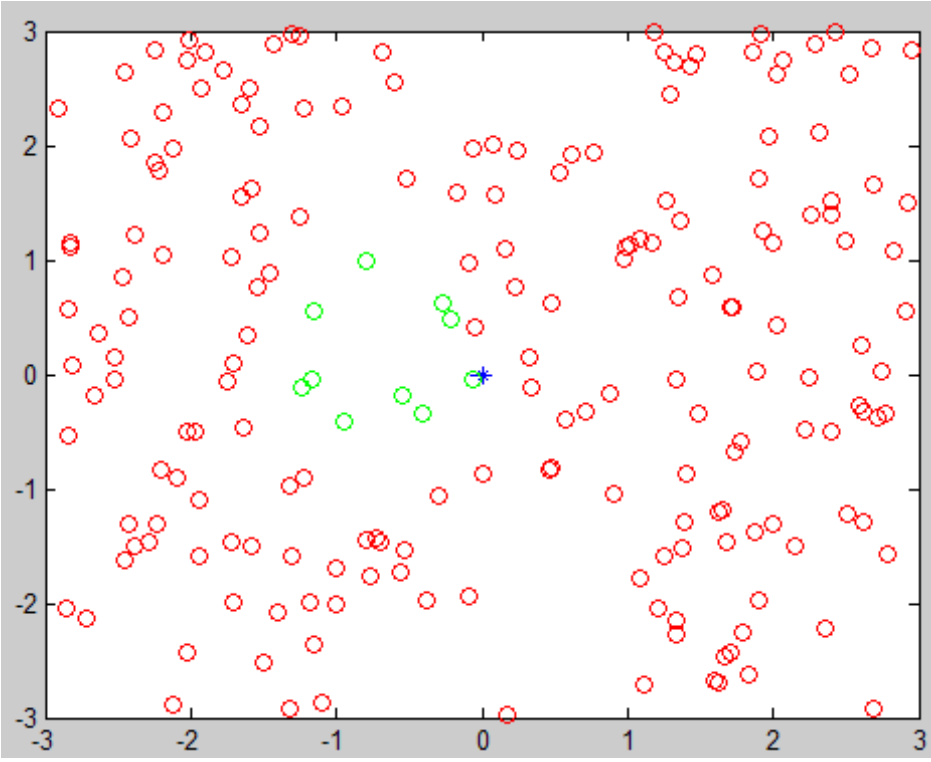

Figure 3.9: Simulation Environment in Matlab (axis labels in [m])

The simulation environment is shown in

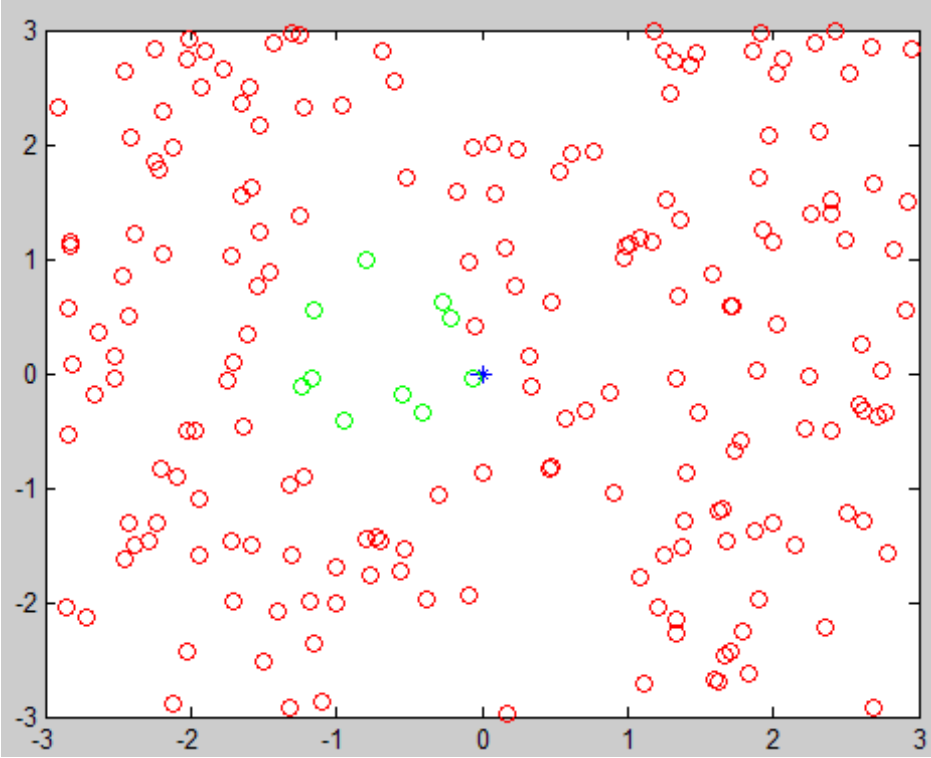

Figure 3.9. Red tags are dormant (out of read range) while green tags are active (responding/within read range). The dimensions of the space correspond to the size of lab environment we will be testing in.

3.2.1. Snapshot Capturing

To compute the RSS of readable tags in the robot environment, it was necessary transform the tag locations to the robots reference frame as shown below in Figure 3.10.

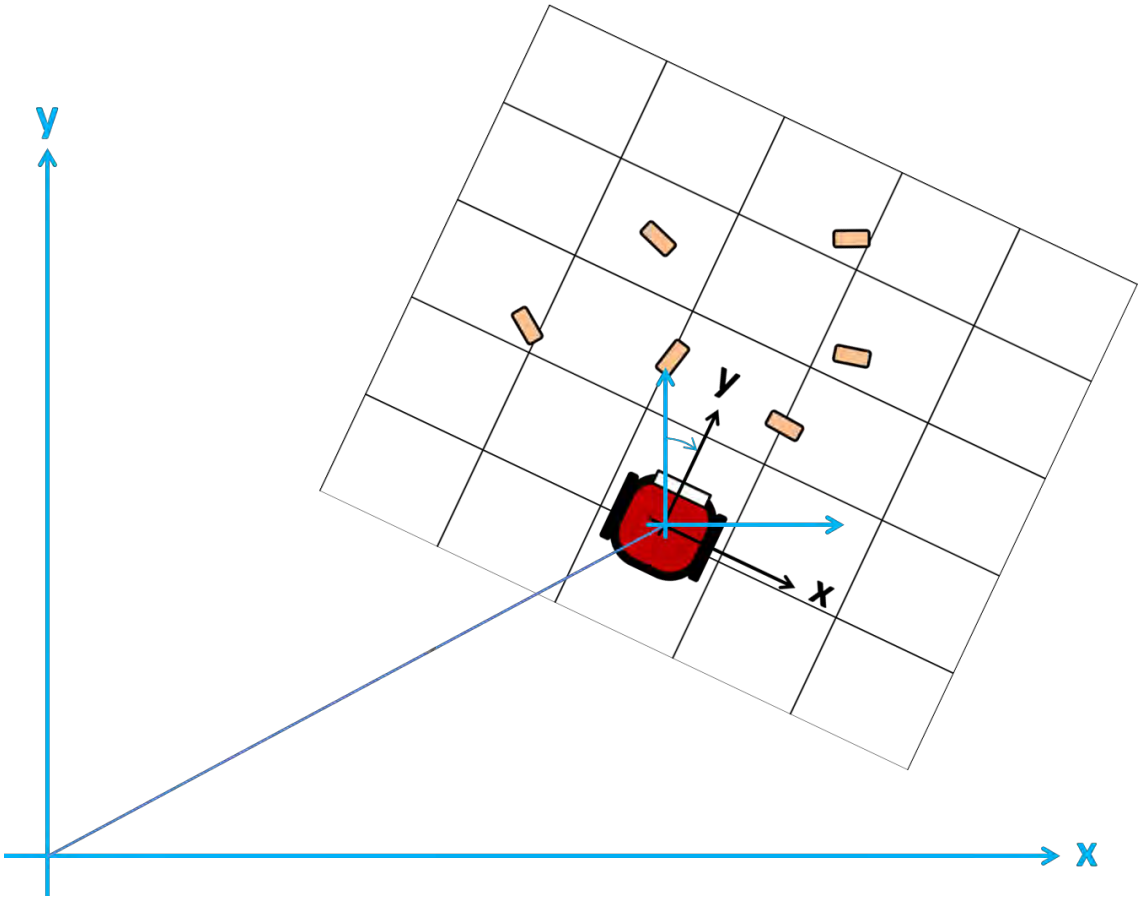

Figure 3.10: Global reference frame to robot reference frame, to compute tag RSS values, as these calculations are performed in a fixed reference frame

Tag values below a threshold of 1000 RSS units were ignored. This was because of an observation made during the sensor modelling, which is that RSS values below 1000 correspond to tags on the verge of the readable range, and thus reply inconsistently and lead to noisy snapshot measurements. A snapshot is stored as a 2D matrix consisting of a tag ID and its corresponding RSS value. A sample snapshot is in Table 3.3. Each snapshot is then added to an observation matrix, which is simply a tiling of these snapshot matrices. Sorting snapshots in accordance with tag IDs was needed to simplify comparisons and speed up clustering.

Table 3.3: Example of a RFID snapshot

Tag ID	ABCD	EFGH	IJKL	MNOP	QRST	UVWX
RSS value	2500	5830	1500	4062	3140	4879

3.3. Exploration, Localisation and Navigation

The localisation and navigation algorithm was comprised of three steps:

1. Exploration

2. Clustering of observations
3. Building a state transition matrix

The flowchart in Figure 3.11 illustrates the overall algorithm starting from exploration to navigation. The *"Log on to reader"* step does not apply to the simulation part of the project as it did not involve the physical implementation hardware.

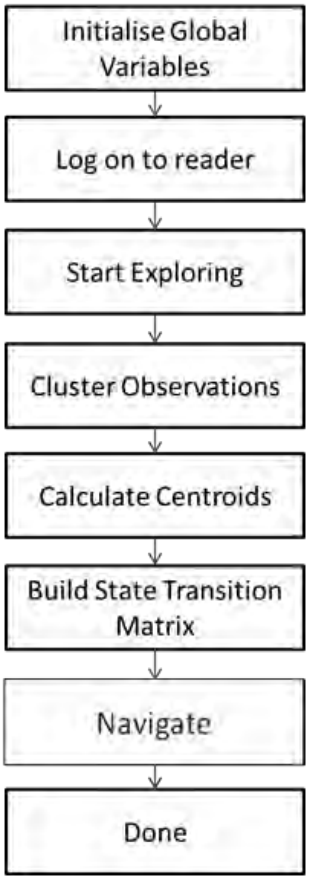

Figure 3.11: Overall Experiment Script

3.3.1. Exploration

Actions, that the vehicle can execute, were numbered from 1 to 5, in place of movement forward (F), arc left (AL), arc right (AR), rotate left (RL), rotate right (RR). These action names and their proximate motions are shown graphically in Figure 3.12.

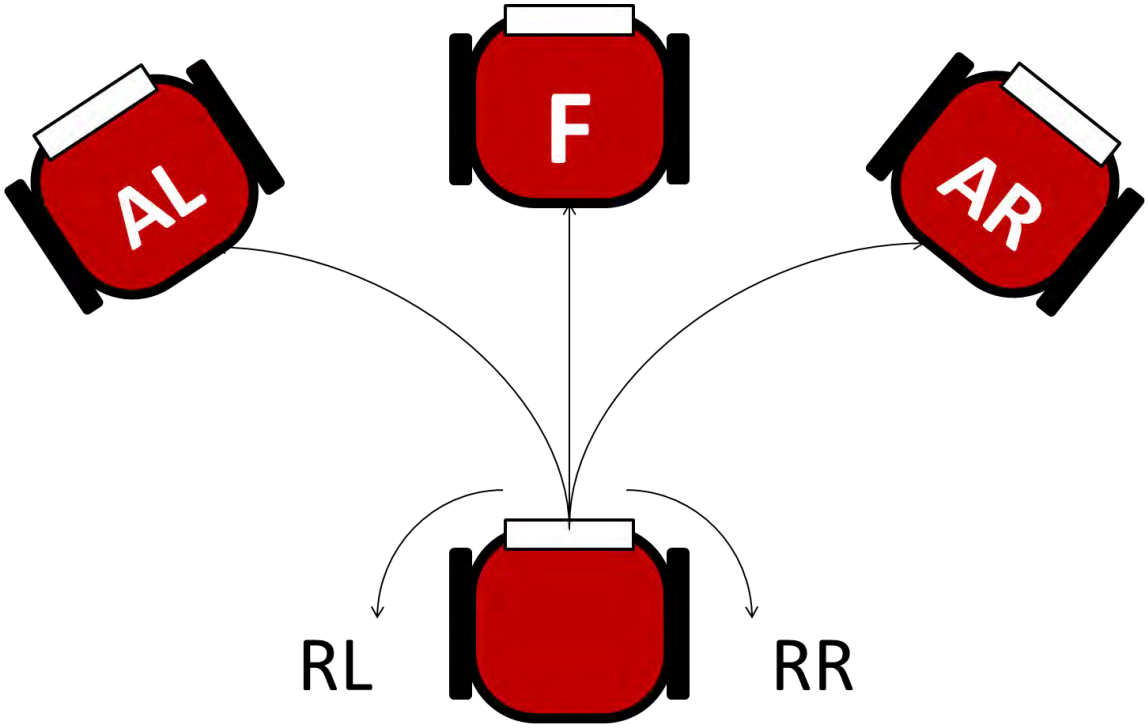

Figure 3.12: Robot action labels and resulting motion used in implementation

Since the vehicle commands used in all the algorithms had to be similar to those to be used on the pioneer platform, action commands we implemented by converting the action numbers (#) to translational and rotational velocities. Table 4 shows the values of these velocities for each action.

Table 4: Actions and their corresponding velocities

Action (#)	**Translational Velocity (m/s)**	**Rotational Velocity (rad/s)**
Forward (1)	$2.5 + \eta_t$	η_r
Arc Left (2)	$2.5 + \eta_t$	$\pi/2 + \eta_r$
Arc Right (3)	$2.5 + \eta_t$	$-\pi/2 + \eta_r$
Rotate Left (4)	$0.25 + \eta_t$	$\pi/2 + \eta_r$
Rotate Right (5)	$0.25 + \eta_t$	$-\pi/2 + \eta_r$

Where η_t and η_r correspond to the translational [-1,1] and rotational noise [$-\pi/5$,$\pi/5$] respectively.

During exploration, the vehicle first takes a snapshot, then chooses each action at random after taking a snapshot. This random choice is not uniformly distributed, but weighted such that the forward and arc actions are more likely during the early stages of exploration and rotations are least likely. This will allow the robot to map out larger areas more quickly.

Additionally the algorithm increases the probability of executing a rotation inversely to the number of tags observed, such that when there are no observable tags, the probability of executing a rotation is 1. Value iteration is not used during the initial exploration, as there are no states, or state transition matrices.

The algorithm used for exploring the RFID environment is shown in Figure 3.13. This algorithm is used to obtain observation (snapshots) of tags within the read range. This rudimentary exploration was later extended to explore frontiers by biasing the forward action (increasing the probability of choosing the forward action) when unknown tags are detected.

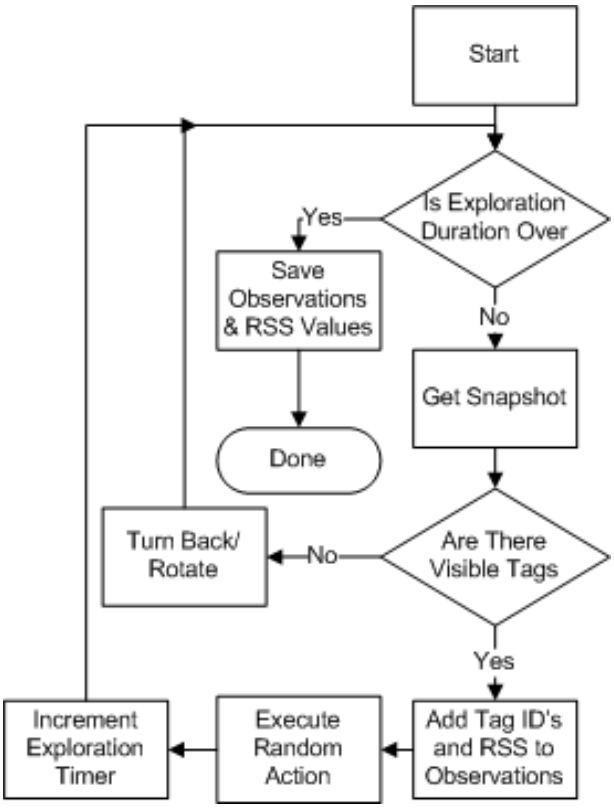

Figure 3.13: RFID exploration algorithm

Given sufficient (more than 500 steps per 25 square metres) simulation time the algorithm is able to explore the entire area multiple times.

3.3.2. Clustering

The clustering algorithms used were k-means, k-medoids and a custom clustering algorithm. After clustering is performed, empty clusters are discarded. The custom clustering is similar to k-medoids except the initial choice of centroids is stationary. When using k-means, to increase the accuracy one can simply increase the number of clusters

(up until the number of observations), knowing that empty clusters will be discarded. Similarly with k-medoids with the exception that k-medoids does no produce empty clusters due to the restriction that a cluster centroid must be an observation. The reasoning behind this being that since it is a much simpler algorithm it should run faster. During experiments a modified hierarchical clustering was also tested. The value of k was chosen empirically, based on the different algorithms performances listed in the tables below. We found the best value (yielding good clustering accuracy, below 0.6 m, whilst minimising the number of clusters) to be between one half and two thirds the number of observations.

Table 3.5: Comparison of clustering algorithms for 1625 observations and 405 States

	k-means	custom clustering	k-medoids
Number of Non-empty Clusters	319	316	358
Ave Cluster Separation x (m)	1.009772	1.040637	0.878805
Ave Cluster Separation y (m)	0.967746	0.996498	0.898314
Ave Cluster Separation d (m)	1.487936	1.536722	1.359275
Ave Cluster Separation θ (rad)	3.130666	2.934081	2.761911
	107.626335	3.218349	0.295258

Table 3.6: Comparison of clustering algorithms for 1625 observations and 813 States

	k-means	custom clustering	k-medoids
Number of Non-empty Clusters	449	550	662
Ave Cluster Separation x (m)	0.846350	0.769972	0.648700
Ave Cluster Separation y (m)	0.813128	0.659460	0.559164
Ave Cluster Separation d (m)	1.302560	1.144415	0.996750
Ave Cluster Separation θ (rad)	2.504633	1.971098	1.770322
Computation Time (seconds)	231.718135	6.645083	0.383748

Table 3.7: Comparison of clustering algorithms for 1625 observations and 1083 states

	k-means	custom clustering	k-medoids
Number of Non-empty Clusters	463	615	785
Ave Cluster Separation x (m)	0.829449	0.714975	0. 510887
Ave Cluster Separation y (m)	0.805798	0.670508	0. 685297
Ave Cluster Separation d (m)	1.277713	1.131585	1.005238
Ave Cluster Separation θ (rad)	2.553838	1.872474	1. 467408
Computation Time (seconds)	380.937944	8.554062	0.376200

Table 3.8: of clustering algorithms for 1625 observations and 1625 states

	k-means	custom clustering	k-medoids
Number of Non-empty Clusters	427	1040	1024
Ave Cluster Separation x (m)	0.851935	0.518023	0.501029
Ave Cluster Separation y (m)	0.821903	0.442401	0.434716
Ave Cluster Separation d (m)	1.304289	0.840413	0.826336
Ave Cluster Separation θ (rad)	2.704753	1.215803	1.197497
Computation Time (seconds)	1142.754865	13.590391	0.530620

Below we compare the three clustering techniques graphically; we measure their performance in terms of the average cluster spread (which will influence the localisation accuracy) in Figure 3.14.

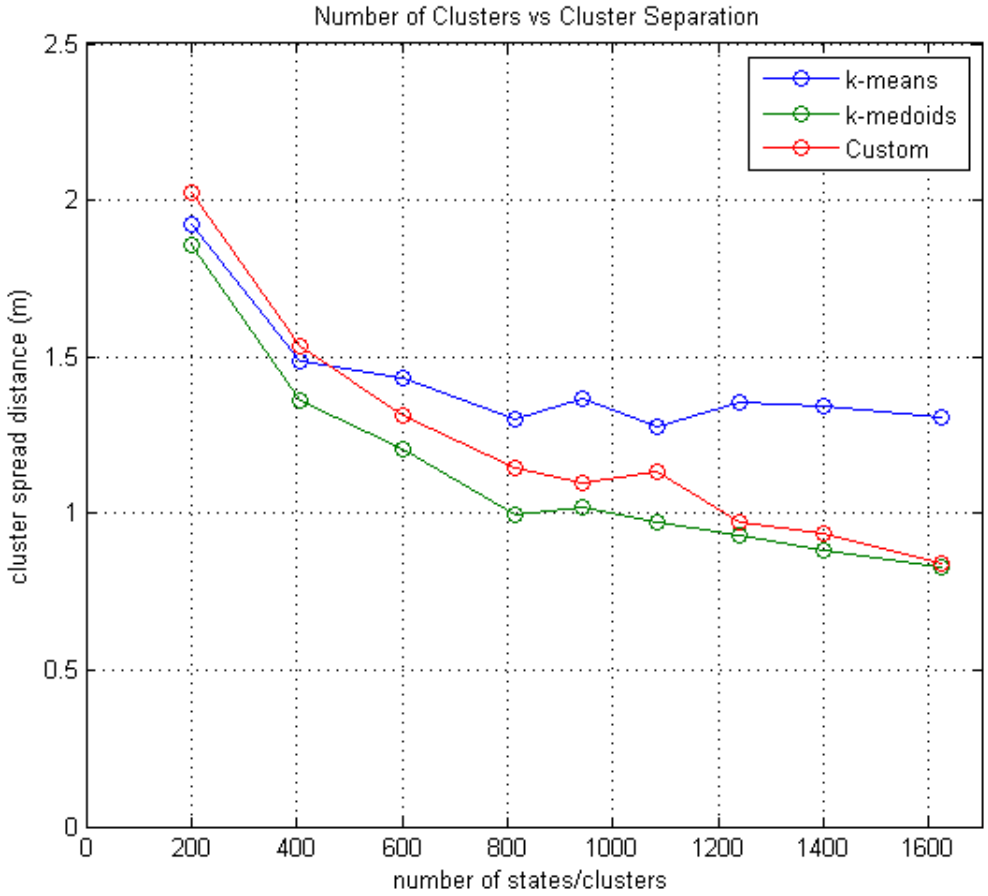

Figure 3.14: Comparison of Clustering Algorithms

From Figure 3.14 we see that the cluster spread is comparable between all three algorithms for number of clusters/states below 400. However this number of clusters is not accurate (1.5 m cluster spread is equivalent to 0.7 m accuracy) for our purposes of navigation. Beyond this k-medoids produces better results than all the other algorithms in terms of accuracy of clustering.

Based on the above reasoning, k-medoids provided the best results and was also the quickest to compute. Therefore we decided to use k-medoids for all subsequent clustering for simulations.

Below shows a sample of clustered observations, clusters are shown by colour. As we can see from the red and purple lines, orientation is preserved.

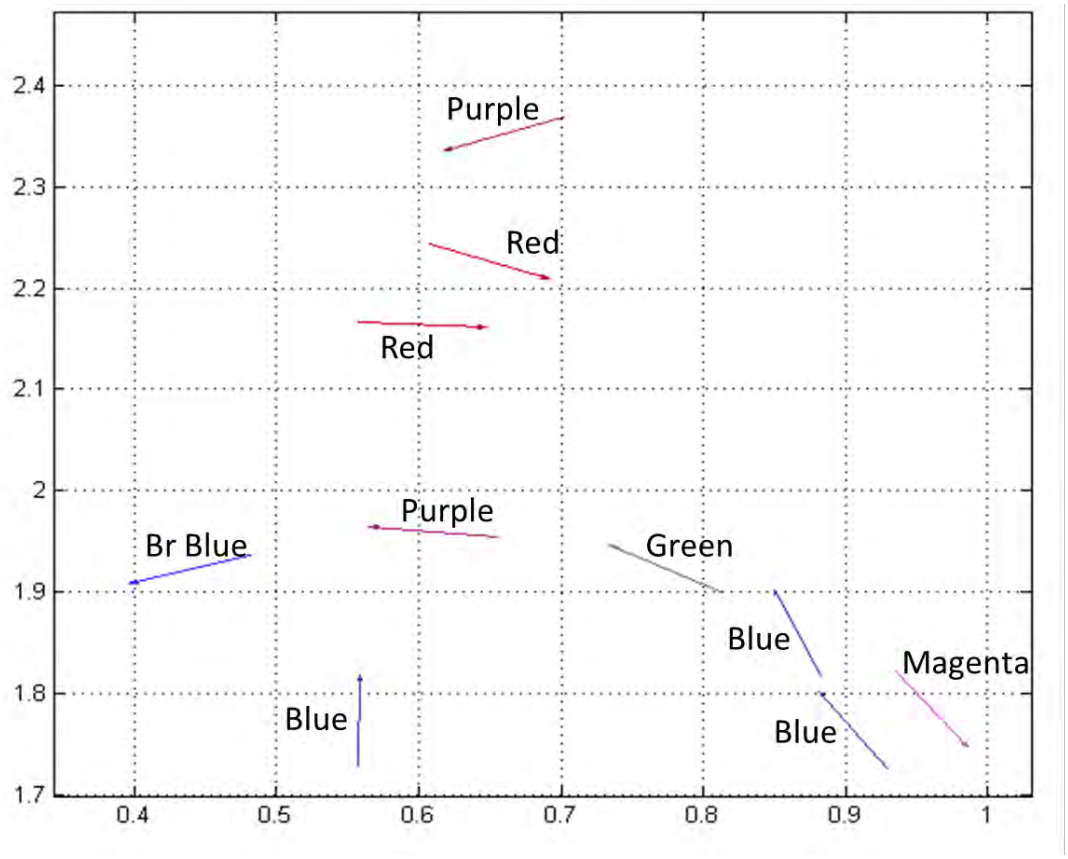

Figure 3.15: Typical clustering output for various vehicle poses

3.3.3. Localisation Method

The localisation algorithm used in the simulations seeks to minimise the normalised difference/distance between the current RFID snapshot RSS values and those of the recorded cluster/state centroids. This normalised distance metric is computed subtracting the RSS values of the current snapshot from that of each of the cluster centroids for corresponding tag IDs. These values are then summed up and compared to the minimum of total RSS sum of either the centroid or snapshots. The proportion is then given a percentage RSS match score. Secondly the number of matching tag ID's between each centroid and the snapshot is given a percentage match. These scores are than summed up with a bias of 60% ID score match plus 40% RSS score match, and the centroid with the highest score is selected as the closest or likely robot pose, thus localising the robot.

This localisation algorithm was tested using the initial observations, and attempted to match each observation to its allocated cluster. The accuracy of the localisation scheme is shown in the *results* section. Additionally another localisation method was tested, which compared only the RSS values for matching IDs.

3.3.4. State Composition

The state vector is a 2 by max matrix, max representing the maximum number of tags which can be read in one snapshot. This max value is preportional to the tag density, and represents the maximum number of tags which can appear in the antenna's readable range (in our case, a circle of radius 1.25 m in front of the antenna as seen in FIGZasa). This matrix consists of a list of tag IDs (10 digit hexadecimal numbers) in the first column and a RSS value varying from 0 to approx 60000 unit in the second column. This state matrix (snapshot) is then tiled into a 2 by max by nObs matrix of observations for storage and subsequent clustering. This state matrix is shown in FIZAasa. Once the observations in the observation matrix have been clustered, the cluster centroid will represent a robot state (i.e when the robot is observing a snapshot closely matching a certain centroid, it will be considered to be in the state represented by that centroid). Essentially a state is a snapshot of the RFID environment, which, after clustering, is represented by the cluster centroids.

3.3.5. Building state transitions matrix and navigating

The state Transition Matrix I represents a measure of the probability of transitioning from one state to another for a given action, in the form of a 3 dimensional [Sx(S+1)xA] (#states by #states+1 by #actions) sparse probability matrix. The clusters from the clustering algorithm were used as states for the state transition matrix. The additional state in the second dimension holds the probability of transitioning to an empty state. The vehicle uses a similar algorithm to that in Figure 3.13 to record the transition probabilities. It does this by exploring the environment, choosing set actions at random and executing them, it then increments the probability of transitioning from the previous state, to the current state, given the chosen action. It attempts to record at least 2 transitions for each action in every state. This ensures an accurate representation of the state transition probabilities. A section of the state transition matrix is shown in Figure 3.16 for the forward action. On the y-axis is the present state, on the x-axis is the state which can be transitioned to and the colour indicates the probability of making this transition (red being probability close to 1 and blue probability close to 0) given the forward action is chosen. Additionally an extra column is added as the probability of transitioning to outside the read range.

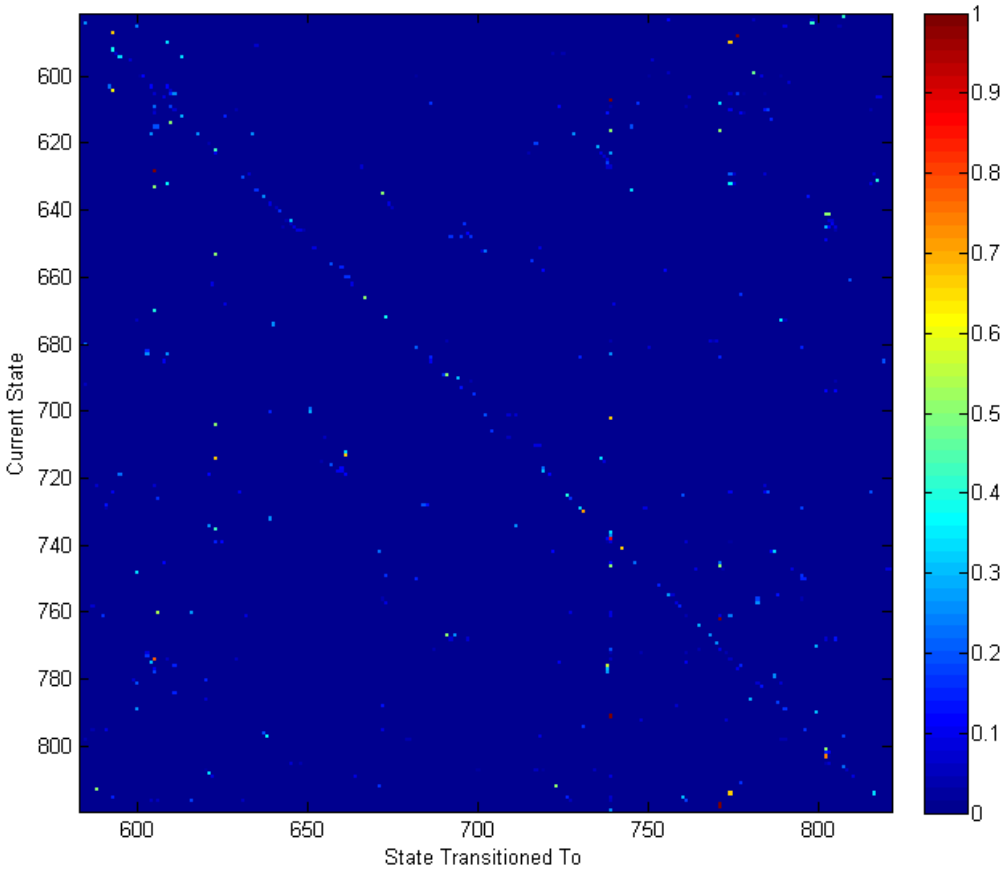

Figure 3.16: Section of State Transition matrix

When the TM has been acquired, the vehicle is now equipped to navigate using a value iteration algorithm. A goal location is selected from one of the states (these could be linked to any area of interest, possibly from images taken by the vehicle). A reward matrix is created with the goal location having a desired positive value and undesirable states (ones that are close to the edge of the environment, i.e. last column) are given negative values. The vehicle then uses value iteration to get an action policy (action to execute given the vehicle is in a certain state) that will allow it to reach the goal whilst avoiding the edges of the environment (where there are no RFID tags). Executing these actions, allows the vehicle to navigate to the desired goal autonomously.

4. Simulation Results

4.1. Localisation Results

These results were obtained in a simulated area of 5 by 5 metres, 800 tags and a simulation time of 1000 exploration time steps. The localisation of the vehicle averaged an error of 0.34 m over 20 simulations/trial runs. The average of the maximum localisation error for all 20 trials was 1.1 m in the same simulations. The table below shows these localisation results from different (randomly chosen) starting positions, to illustrate the performance of the algorithm over varying distances.

Table 4.9: Numerical localisation results over 20 trial runs

Test #	Max Loc Error (m)	RMS Loc Error (m)	Initial Distance to Goal (m)	Max Distance to Goal (m)	Time to Goal (st)	Comments
1	0.263108	0.113088	1.451262	1.451262	28	
2	1.373005	0.408296	2.017578	2.017578	44	
3	0.277988	0.115034	2.612348	2.613312	45	
4	0.266809	0.127836	1.618905	1.618905	21	
5	1.209967	0.339866	1.158275	1.158275	20	
6	2.573254	0.962738	5.180108	5.184237	85	
7	1.209967	0.339866	1.158275	1.158275	20	
8	0.770608	0.289507	2.255162	2.255162	77	
9	0.396196	0.175624	2.562783	2.562783	30	
10	0.84493	0.253016	2.035301	2.035301	182	Circling
11	2.651533	0.997474	2.890733	3.259291	43	
12	0.564744	0.213986	3.679879	3.701556	57	
13	0.318143	0.180671	1.854034	1.856366	37	
14	0.948004	0.246589	3.370003	3.370003	89	
15	1.656564	0.412161	5.620122	5.620122	71	
16	0.832765	0.18374	4.616417	4.616417	51	
17	3.670778	0.858748	6.051785	6.775494	488	Limit Cycling
18	0.355889	0.167153	1.706876	1.935891	48	
19	0.407162	0.143514	2.070111	2.170551	41	
20	0.663023	0.229172	2.984635	2.984635	114	

Ave	1.0627218	0.33790395				

A sample run of the localisation error is shown in Figure 4.1 below and a run with extended simulation time is shown in Figure 4.2.

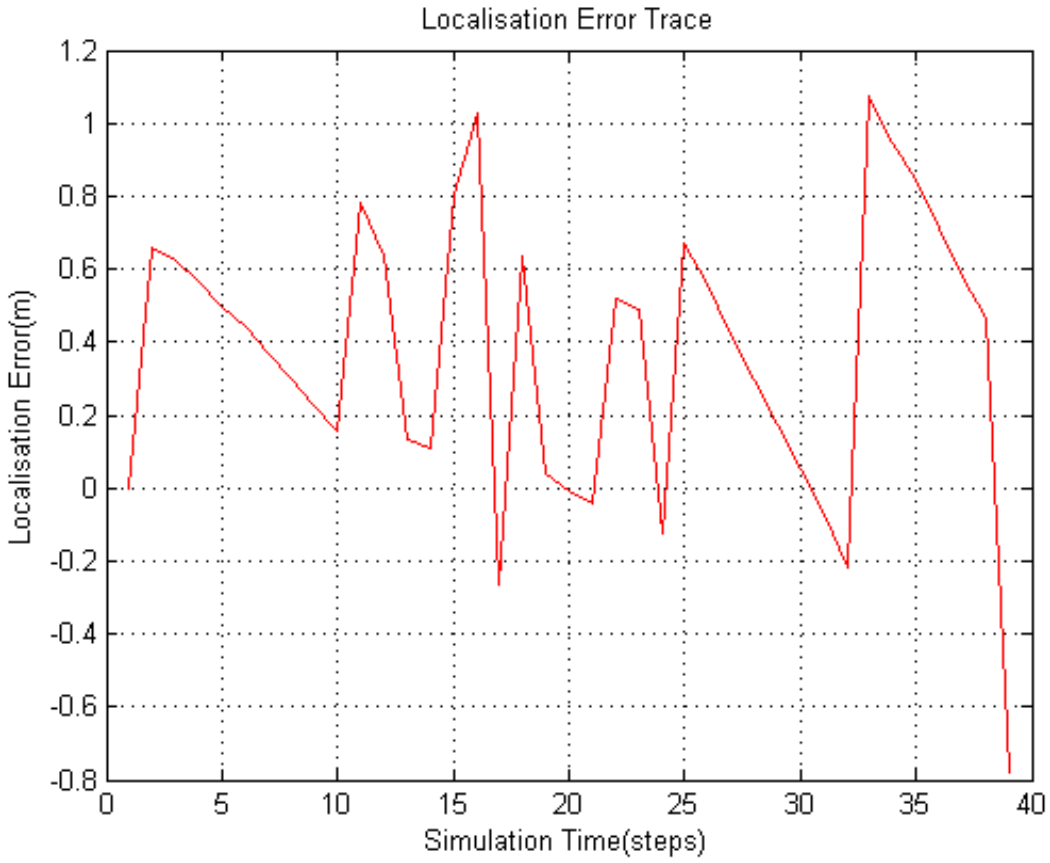

Figure 4.1: Localisation error simulation results

We can see clearly from Figure 4.2 below the cycling between states which occurs on the border between states. Stricter clustering (using a higher number of clusters), although allowing more precision (due to higher density of centroids, meaning the distance to the closest cluster is reduced), can lead to worse accuracy manifested as this fluctuation between alternating states. Stricter clustering in this instance means using a higher number of clusters in the case of kmeans/kmedoids (leading to smaller clusters and ideally more accurate localisation). Having smaller clusters means the centroids are closer together, the action policy therefore may give similar actions or actions which may be in opposition due to slight variations due to noise in the snapshots. This can cause a situation, in the case where the goal is directly behind the robot , where the policy gives an

action rotate left, upon executing that action and ending up in a different state very close to the previous state, the action policy then gives an action rotate right. Thus the vehicle begins to cycle between these two state action pairs.

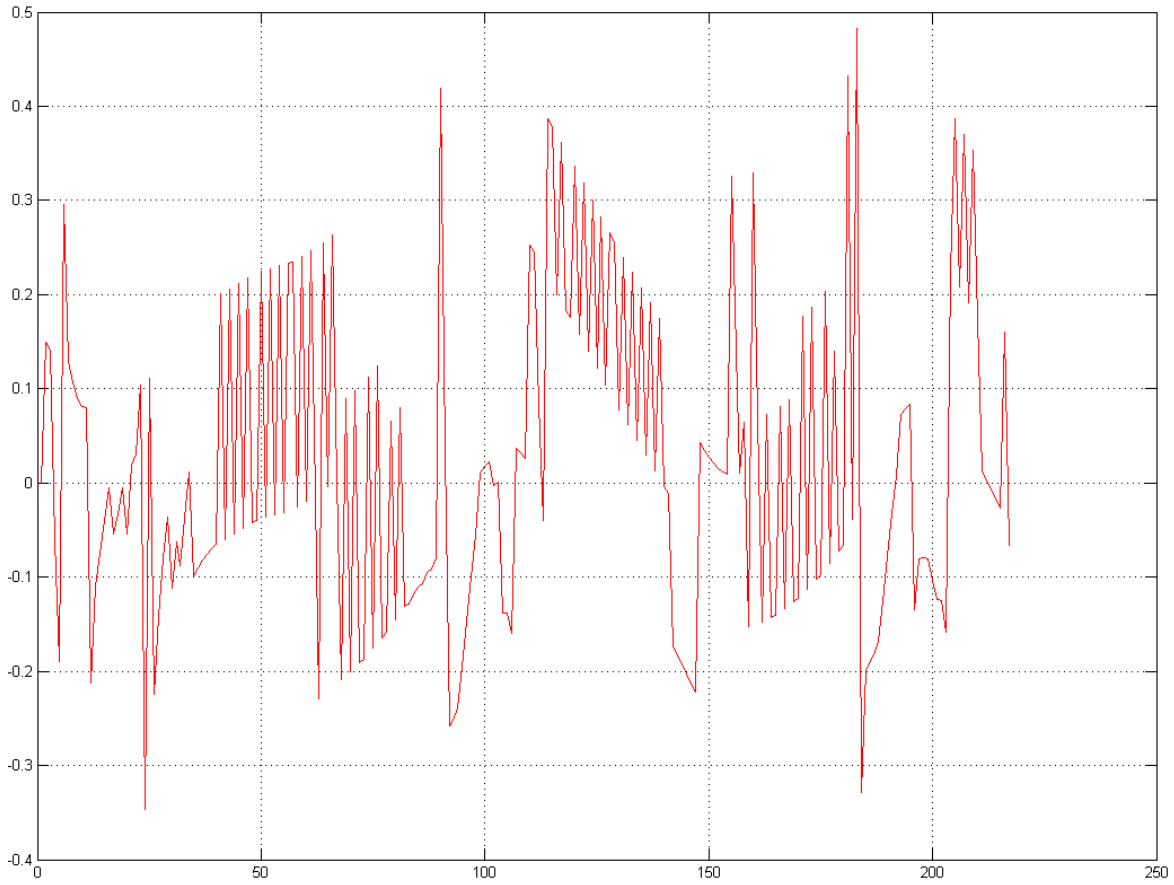

Figure 4.2: Localisation error over long simulation time (y-axis: localisation error [m], x-axis: simulation time)

Figure 4.3 below shows the vehicle position (in blue) in the simulation environment, the purple is the closest cluster that the vehicle has localised itself in and the green is the goal state/cluster. As in previous simulations the tags are the red circles.

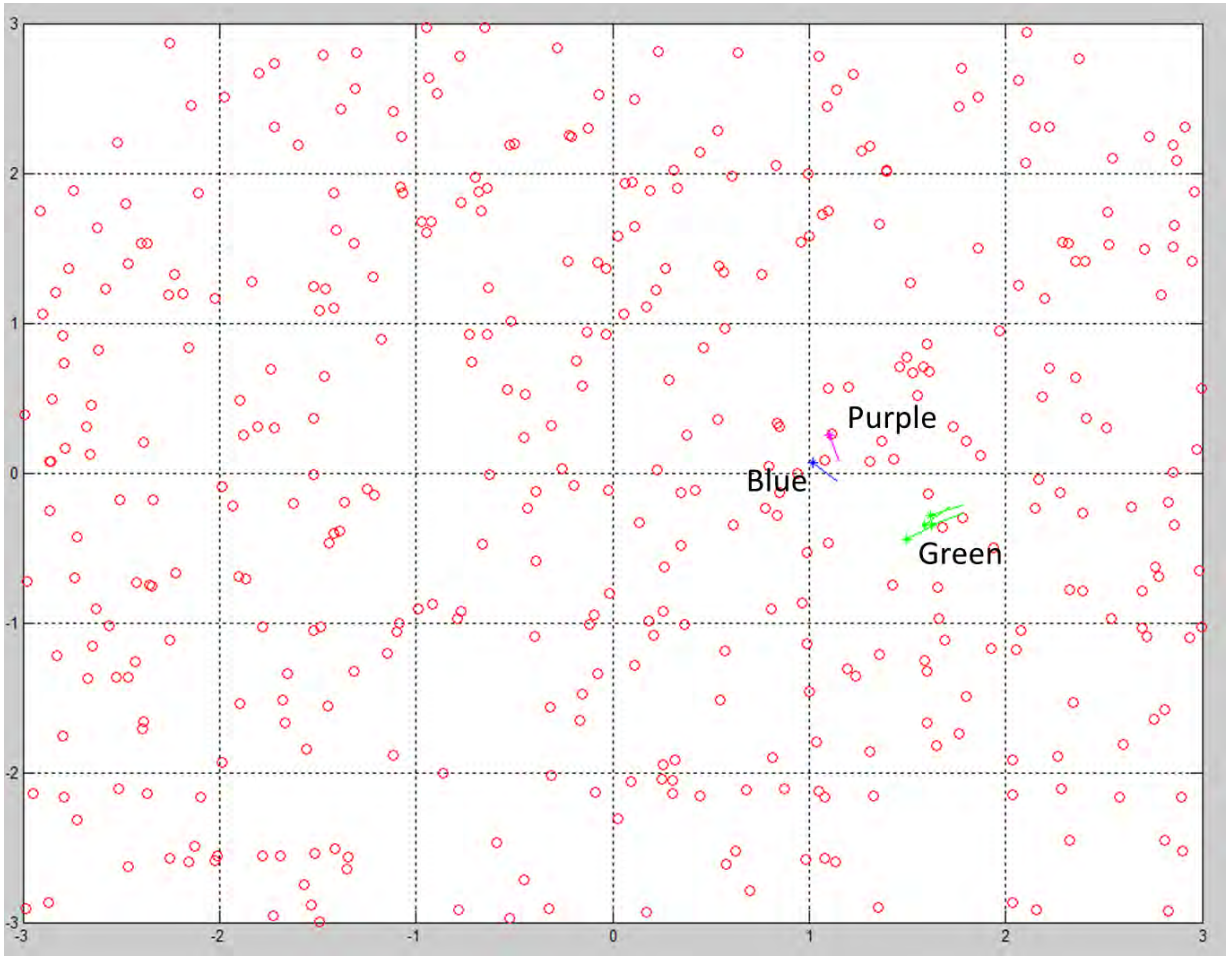

Figure 4.3: Simulation environment showing localisation with precision below 0.2 m

When performing simulations over longer time periods, which would produce a lot of observations close together, we noticed that clustering would occasionally assign circular clusters. These clusters, although giving accurate co-ordinate localisation, cause ambiguity in the robots pose orientation and hence poor navigation. Figure 4.4 shows this result.

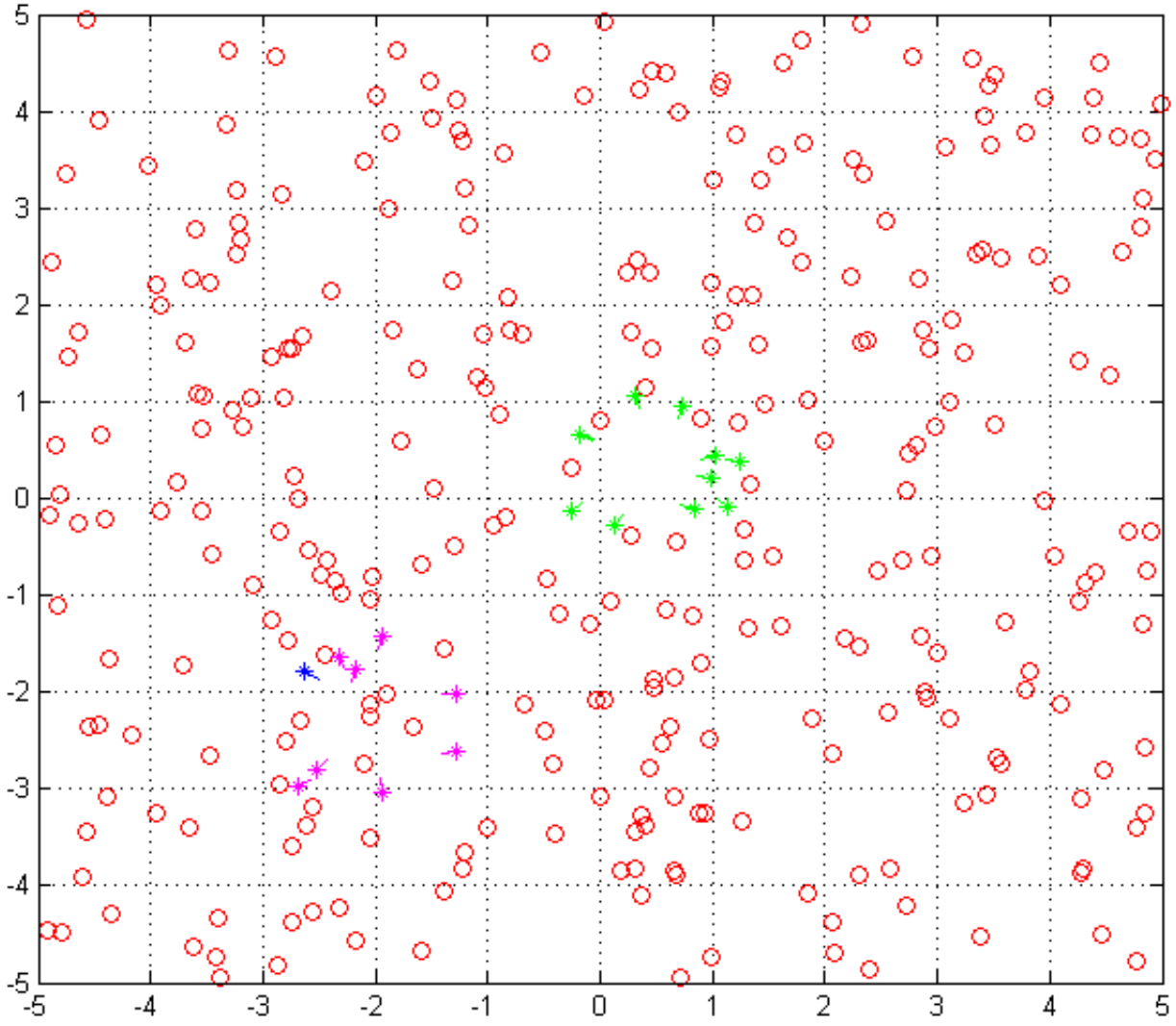

Figure 4.4: Simulation showing how clustering can lead to circular clusters and ambiguous orientation

On the border of the environment localisation tends to be very poor, since the clustering seems to cluster observations with small RSS values together (which are most observations on the outskirts), leading to these clusters having centroids which are close to the centre. Figure 4.5 below shows an example of this bad clustering, as above the robot is in blue, the cluster in purple and the goal in green.

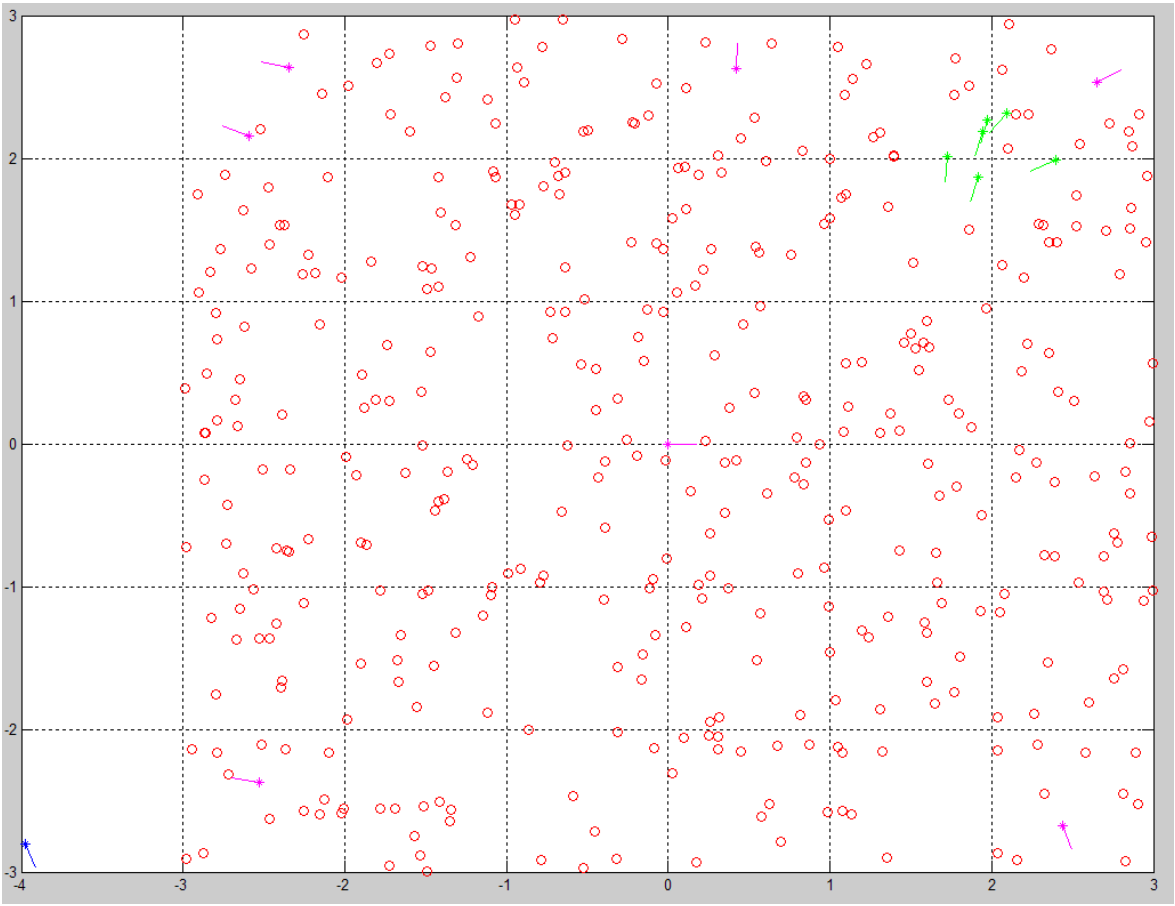

Figure 4.5: Simulation example of bad clustering leading to poor localisation

Another observation of the simulation environment was that the determinism of the simulation lead to limit cycles wherein the vehicle would cycle between two opposing actions. This happens when the goal is either directly in front or behind the robot, and the localisation is not aligned with the navigation policy (the robot chooses to turn left when it is on the right and chooses to turn right when it is on the left, due to localisation alternating between opposite states incorrectly). This can also happen when the navigation policy leads to the vehicle to circle the goal without actually reaching it. We call this limit cycling and it is caused by the limited number of actions and a sparsely populated state transition matrix. It was observed that adding noise to the robot motion eventually eliminates limit

cycles. This is because limit cycles are the result of being stuck in an infinite action-decision loop; adding noise can change the results of actions or influence the decision made or both. Since introducing noise was observed to eliminate limit cycles, it was introduced in the simulations, but it will not be necessary to include noise introduction in the experimental application, as the real world already has noise. Figure 4.6 shows an example of limit cycling using the localisation error.

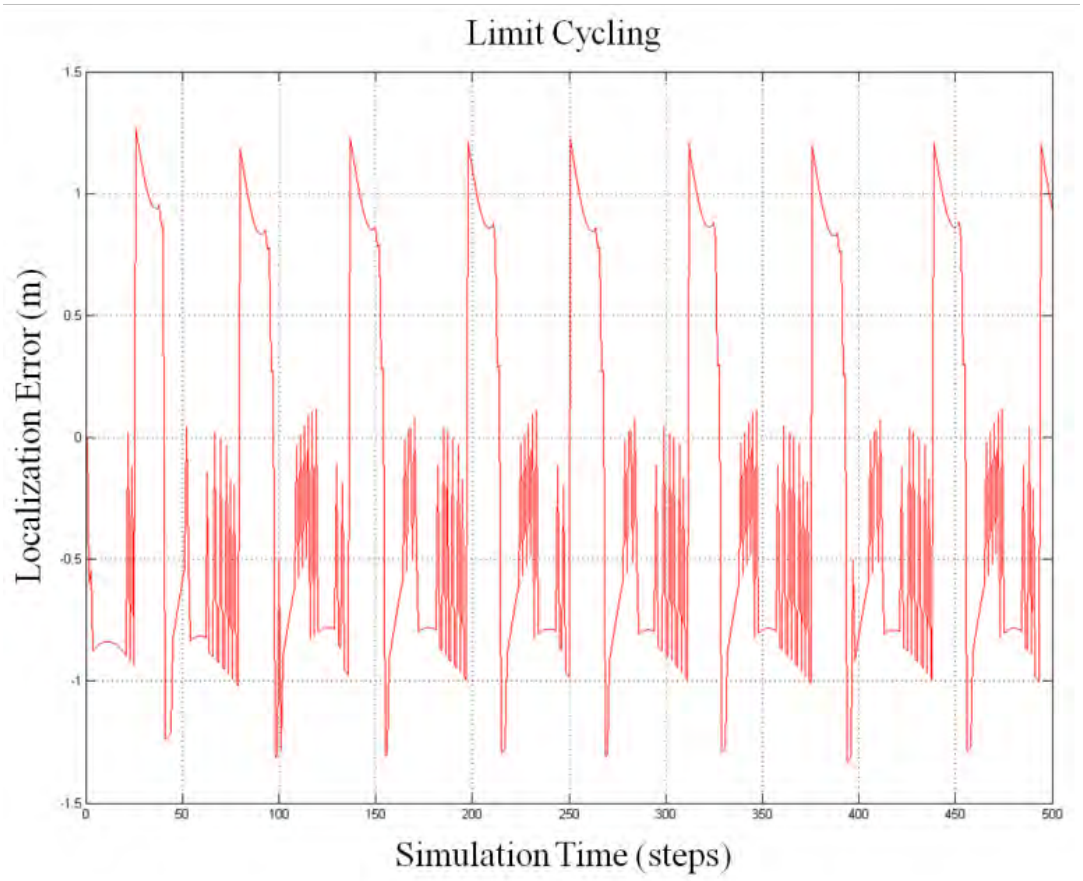

Figure 4.6: Limit cycling due determinism of the simulator

4.2. Navigation Results

Figure 4.7 below shows a sample of the navigation results. As we can see the displacement error decays consistently over time.

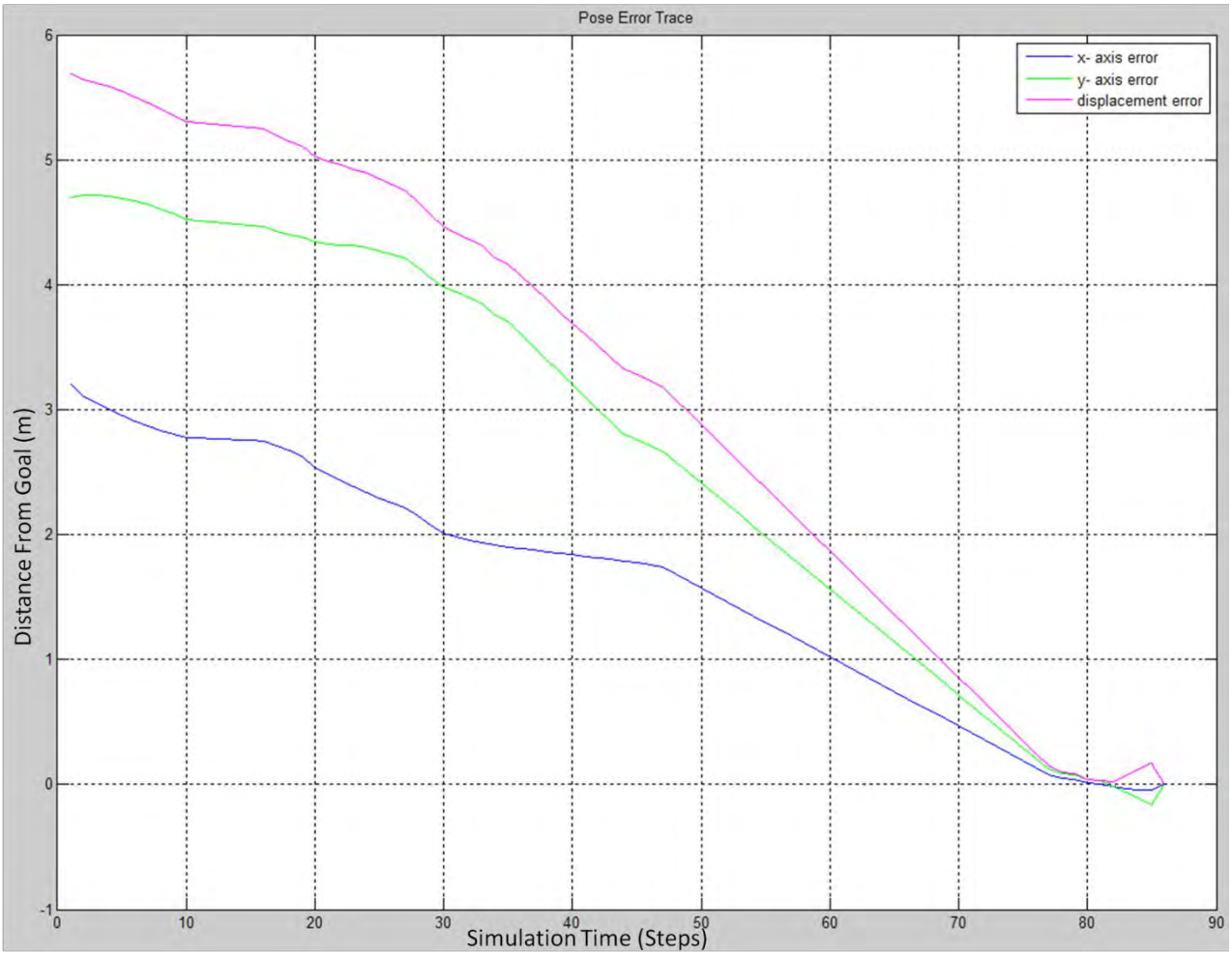

Figure 4.7: Navigation showing error from goal state over simulation time

Another error trace is shown in Figure 4.8; in this case the robot starts facing away from the goal, which can be seen by the initial increase in error as the vehicle turns around. In addition the orientation adjustment can be seen when the error increases slightly closer to the goal.

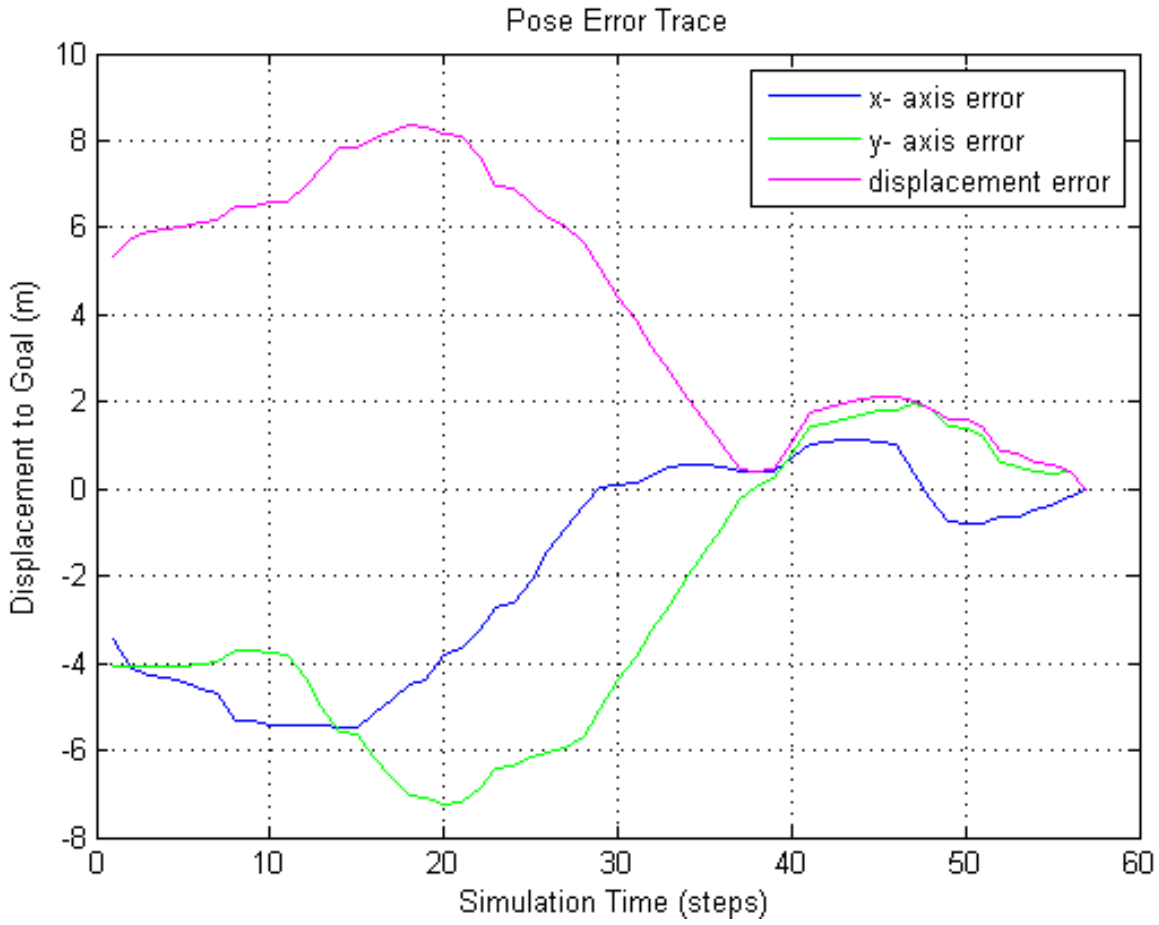

Figure 4.8: Distance to goal over simulation time steps

The reason for the robot not following a straight path may be due to the limited actions available to the robot, the limited time for exploration as well as the spread of the clusters i.e. there exist paths which may not have been explored yet, and states which have not been visited along the straight path. This leads to the robot path more likely to be along explored states. The non-visiting of states can be attributed to two reasons, firstly time/steps for explorations being limited and secondly, clustering can lead to some states being larger than others thus overshadowing smaller states, hence the vehicle may never register some clusters even though it may be the same cluster location

It is worth noting that when the vehicle is on the outskirts of the tag area (very low RSS), the poor localisation results in poor navigation results. The worst case results in the

displacement error not decaying to zero. Additionally, the clustering algorithm can lead to poor clusters depending on the starting centroids used. The clustering tends to put all the observations with low RSS (usually outskirts) in the same cluster, despite the different tag identities. This leads to observations at the centre of the environment being part of the cluster with observations on the outskirts. This is because the centroid will be in the centre of the environment. As seen in Figure 4.9, poor clustering can result in limit cycling and this can lead to bad navigation, where the robot cannot navigate to the goal, and as a result the distance to goal does not decrease to zero over time, Figure 4.9 illustrates this result.

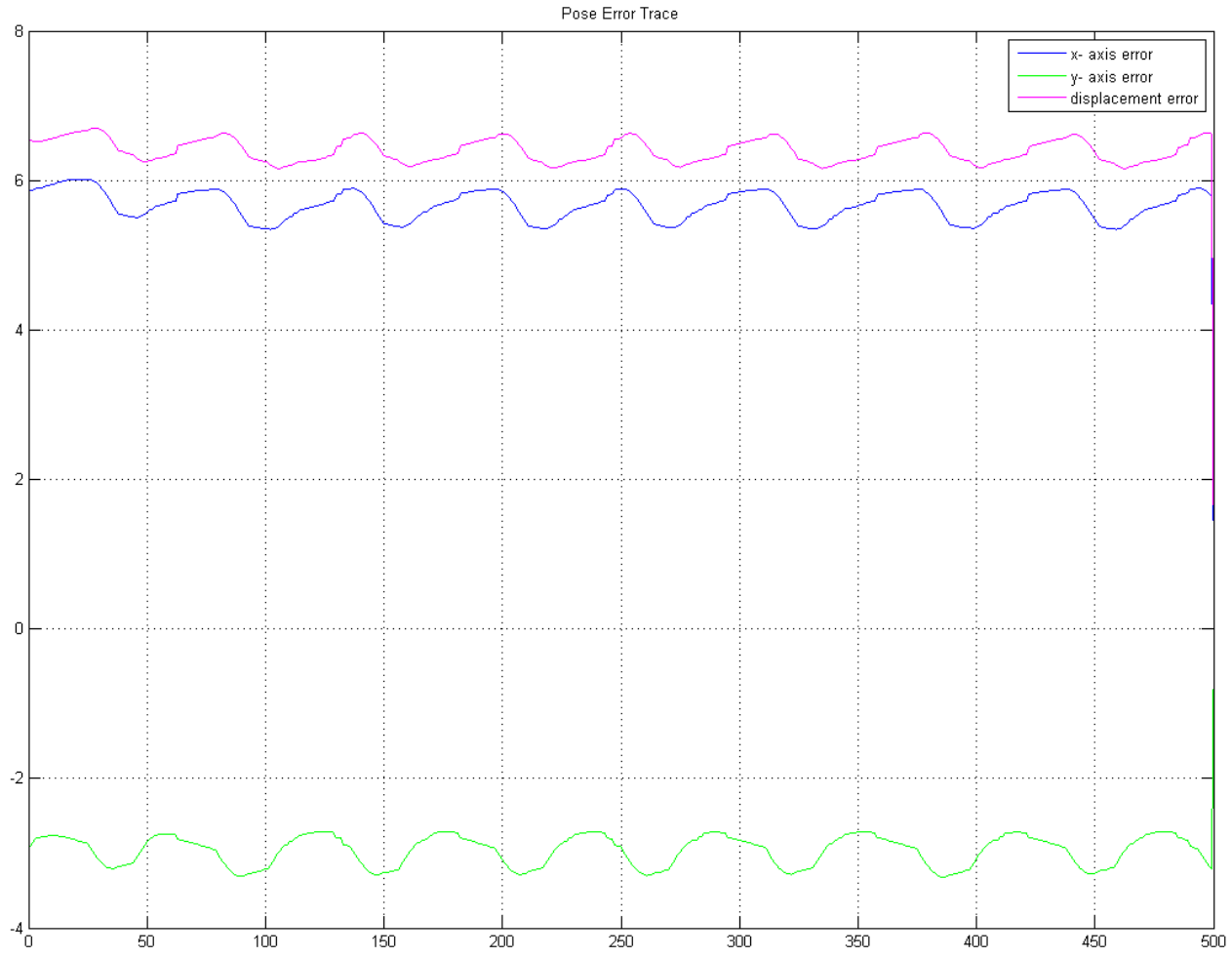

Figure 4.9: Limit cycling leading to poor navigation on the outskirts of the environment

5. Experiments and Setup

For the second part of this work, it was desired to test the results of the simulation on the physical robot platform. For this stage, a *Pioneer P3-DX (Adept Mobile Robots)* was used; the platform was controlled using Robot Operating System (ROS: Electric 2011) [69] software running on a mini-notebook (dell inspiron) machine. The algorithms for the experiments were also written in Matlab 2011b, which in turn used MEX-functions to control the pioneer platform over a wireless connection.

5.1. Experimental Setup

Experiments were conducted in a lab of 5 m by 5 m with a line grid overlaid on the floor as the ground truth for the location of the robot. The tags were dropped at random locations and orientations in an approximately uniform distribution throughout the environment, their locations were recorded for ground truth. This setup is shown in Figure 5.1.

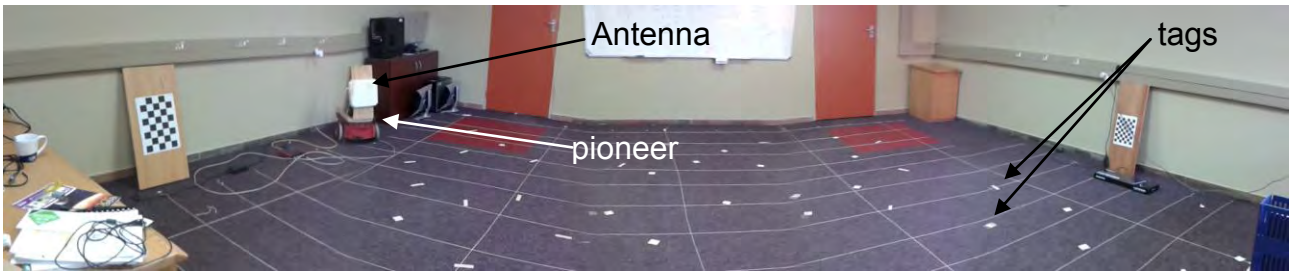

Figure 5.1: Experimental environment setup

The notebook, which is connected to the pioneer via a serial connection, sends control commands from ROS to the pioneer hardware platform. These can be motion commands, such as translational and rotational velocity or sensor queries, such as odometry readings. This notebook was also running the mex-Server, which receives Matlab commands over Wi-Fi (from a client computer) and translates them into ROS commands, which in turn are sent to the platform. The connectivity setup for the experiments is shown in Figure 5.2 below.

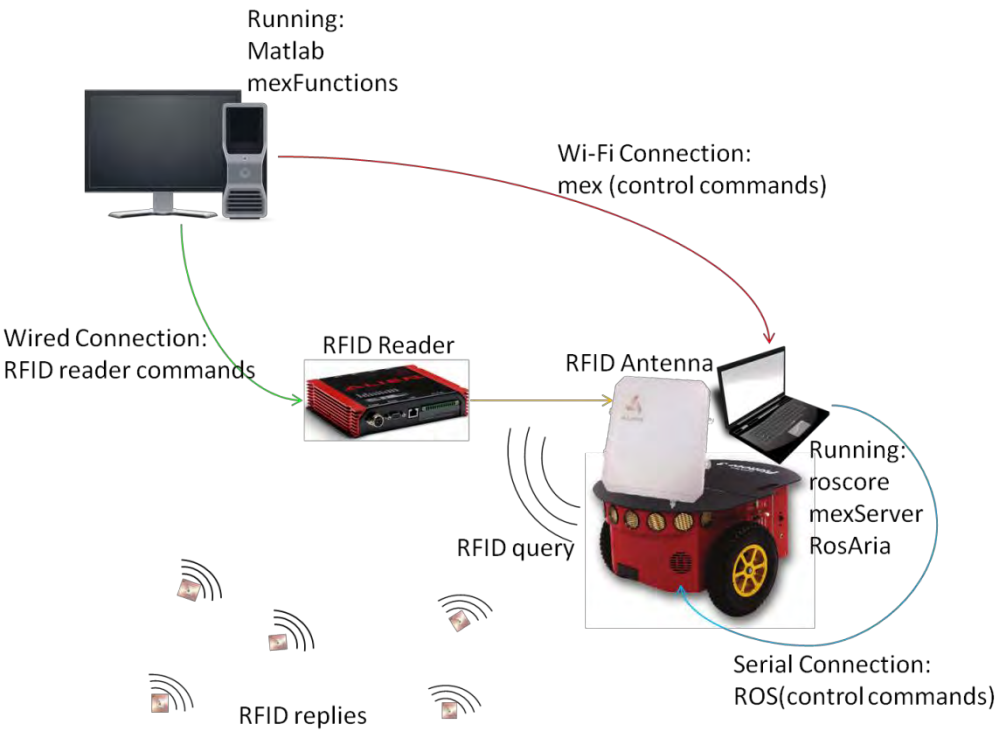

Figure 5.2: Experiment Connection Setup

During experiments, it turned out that the format of the tag ID (a 24-digit hexadecimal number), was too big for Matlab to distinguish between different tag IDs due to rounding errors. It was also observed that tags positions were subject to random shuffling and inconsistency during different observations. This made it difficult to use the clustering algorithms used in the simulator (k-mean and k-medoids), without sorting and using positional look-up tables (to place identical tags in the same row), as they require the data to be aligned. It was decided that implementing these measures would be computationally expensive especially for larger areas. To address issue of tag IDs, we chose to convert and process the tag ID as a 24-char string. We then used a modified form of a hierarchical clustering algorithm to cluster observations, because of its simplicity in implementation and

its robustness to unaligned data. The custom clustering was done using a thresholding technique to find the closet observations globally. The custom clustering algorithm is shown in Figure 5.3 and is based on a linearly increasing threshold for cluster matches. It is similar to hierarchical clustering, except observations are only grouped together if their separation metric is below this moving threshold.

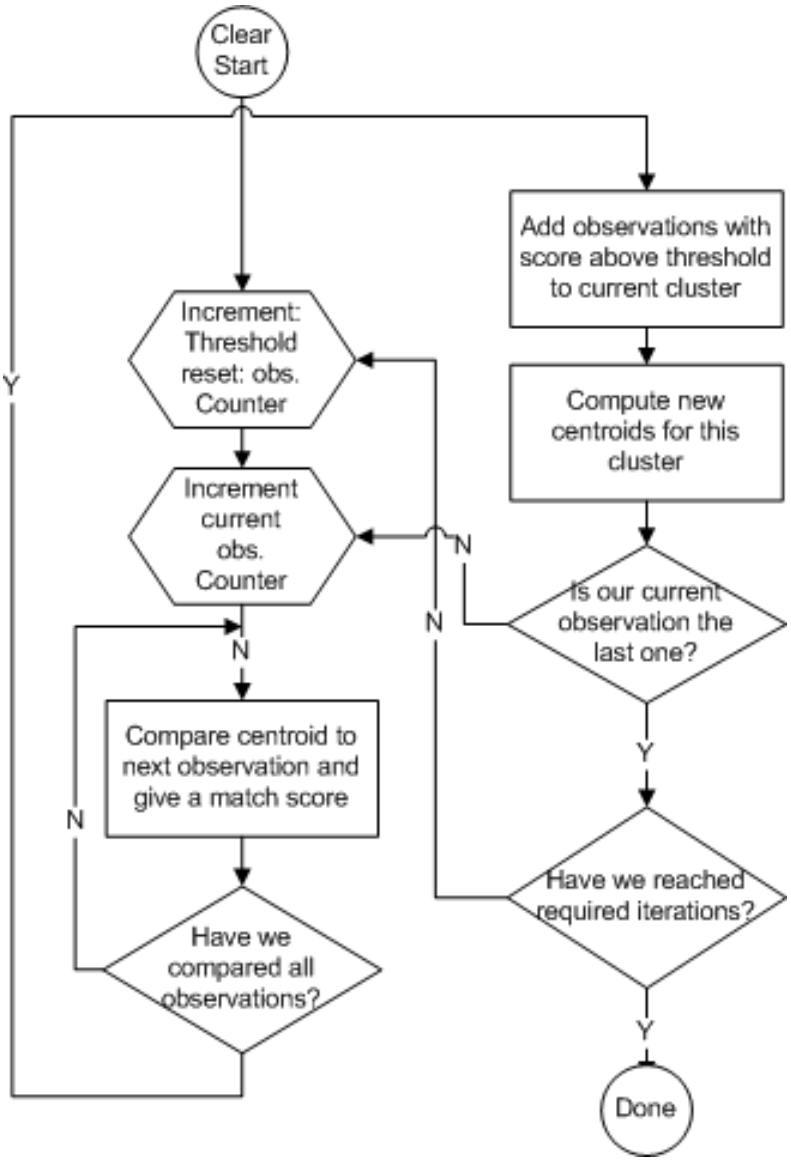

Figure 5.3: Custom clustering algorithm

5.2. Localisation Testing

Figure 5.4 below illustrates how the accuracy of the localisation was measured. Since the cluster centroids are internal to the robot, and are topological rather than metric, it was decided that the accuracy would be measured experimentally by the diameter of each cluster.

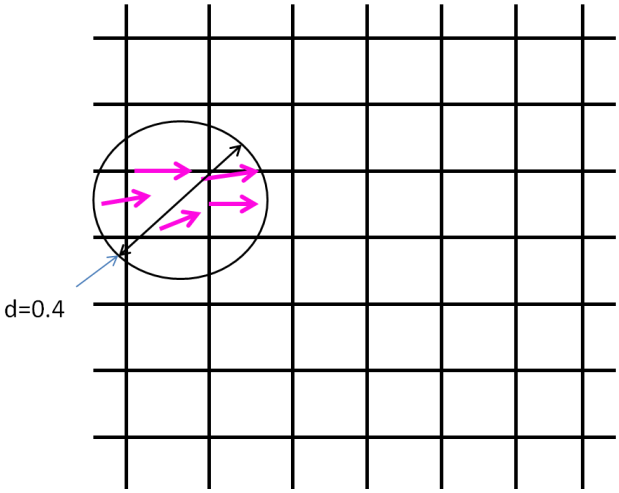

Figure 5.4: How the localisation accuracy was measured

Localisation tests were conducted by obtaining the ground truth of the robot in the environment. The robot was then placed at random locations in the environment and issued a localisation command. The resulting cluster was recorded, then the robot was moved a set distance (0.1 m) and the process was repeated. We record the localisation error as the distance the vehicle must move until it registers a change in state. It is worth noting that, on the border between two or more states, the robots localisation may cycle between these bordering states. This fluctuation is measured as a percentage of incorrect state localisations (assuming the most recent state with above 90% certainty is the correct state) versus the total number of localisations performed. We call this percentage the localisation certainty.

The above method was also used to assess the rotational localisation accuracy, although for our navigation purposes, we only required an angular accuracy of 45° due to the fact that our action commands rotated the vehicle by 45° and no less.

6. Experimental Results

This section looks at the results from testing our simulation algorithms on the pioneer platform in a lab environment. During the practical implementation of the exploration algorithm, it was found that the tag orientations' effect on the RSS value was negligible when compared to the random fluctuations of RSS values between sequential queries. This simplified our tag placement, since we could ignore tag orientation.

6.1. Localisation Results

The localisation of the vehicle was shown to be able to distinguish different clusters from distances of over 0.43 m over 13 random locations and four angles in rotations of π/2 (90°) at each location with 100% accuracy. Below this distance, the accuracy declines rapidly leading to spurious localisation of 30% consistency at 0.2 m. The straight line localisation accuracy (which is the measured localisation accuracy when moving in a straight line in varying steps/separation distances) is shown in Table 6.10 for various distances, and was measured by the number of incorrect vs. correct localisations.

Table 6.10: Experimental Localisation Results

Separation distance/precision (m)	Localisation consistency/repeatability
0.4	90%
0.3	50%
0.2	30%

In addition to the co-ordinate consistency, the orientation consistency was also tested. These results are shown in Table 6.11 below, and were acquired by placing the pioneer at intervals and changing the orientation. We then issued localisation commands and recorded the most frequent state as the *correct state* and all others as the *error states*. It was found that there was ever only one error state (as can be seen from the table). The co-ordinates measured x from left to right, y from bottom to top of the lab area and 0 along the positive y axis (This is shown in Figure 6.1).

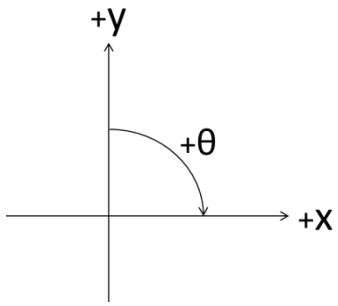

Figure 6.1: Co-ordinate reference frame

The highlights show when the vehicle is on the border between two states, thus leading to a decrease in % consistency.

Table 6.11: Orientation Consistency Test Results

Theta (deg)	x (m)	y (m)	Correct state	Error state	Consistency %
0^0	0.8	0.40	31		0
0^0	0.8	0.80	157	18	30
0^0	0.8	1.20	18		0
0^0	0.8	1.60	18		0
0^0	0.8	2.00	307		0
0^0	0.8	2.40	340	8	20
0^0	0.8	2.80	8	21	25
0^0	0.8	3.20	21		0
0^0	0.8	3.60	473		0
45^0	0.8	0.40	32	82	10
45^0	0.8	0.80	82	18	45
45^0	0.8	1.20	319	8	10
45^0	0.8	1.60	8		0
45^0	0.8	2.00	8		0
45^0	0.8	2.40	56	8	30
45^0	0.8	2.80	21		0
45^0	0.8	3.20	124		0
45^0	0.8	3.60	105	160	25
90^0	0.8	0.40	82		0
90^0	0.8	0.80	119		0

90^0	0.8	1.20	8		0
90^0	0.8	1.60	92		0
90^0	0.8	2.00	36		0
90^0	0.8	2.40	306		0
90^0	0.8	2.80	139		0
90^0	0.8	3.20	124		0
90^0	0.8	3.60	0		0
270^0	0.8	0.40	0		0
270^0	0.8	0.80	0		0
270^0	0.8	1.20	31		0
270^0	0.8	1.60	82		0
270^0	0.8	2.00	31		0
270^0	0.8	2.40	222	31	30
270^0	0.8	2.80	31	18	40
270^0	0.8	3.20	8		0
270^0	0.8	3.60	8		0

7. Discussion and Conclusions

From our simulations, we verified the consistency of our RSS calculations by comparing them to the beam pattern of the *Alien Antenna* data sheet. We can conclude that localisation below 0.4 m precision is achievable with over 90% consistency. For our navigation simulations, looking at the state transition matrix we can see that the diagonal shows that for small movements of the forward action there is a finite probability of staying in the same state, this is influenced both by the localisation consistency and the displacement of each action. Based on the achieved consistency, we recommend using an action displacement of at least 0.4 m for each translational action. This will reduce the amount of localisation noise, as it ensures that the robot has completely transitioned to another cluster.

From our experimental results we can confirm that localisation consistency below 0.4 m (linear) precision is achievable with at 90% consistency (i.e. 90% of the time). This is comparable to the accuracies achived by Schneegans et al [31] and Vorst et al [32], although their consistency was not mentioned. Additionally orientation precision of 45º is also achievable. This was precise and accurate enough for navigation, since our actions did not contain rotations of less than 45º or translations of less than 0.4 m.

Localisation consistency noise/errors were caused by the fluctuating number of visible tags from the RFID reader; this caused the localisation algorithm to cycle between states which are close to each other.

It was also noted that, strict clustering can lead to better precision, but this method of increasing precision is also more susceptible to these fluctuations, hence consistency deteriorates; strict clustering although achieving better localisation, leads to erroneous autonomous navigation because of this.

We can conclude that localisation using only RFID tags is both feasible and practical. This means it may be practical to apply it in environments without GPS or stationary landmarks, such as underground.

Navigation was proved to be feasible from the simulation results, where successful navigation from distances of up to 8 m was achieved. These simulation results need to be confirmed experimentally to verify their accuracy and practicality.

8. Future Work

Due to time constraints it was not possible to fully test navigation using only RFID tags, although it was implemented on the pioneer platform. Future work should look at how navigation would be tested and judged both for end goal accuracy (closeness to goal in metres and orientation in degrees) and navigation efficiency (i.e. time/steps to goal). Additionally particle filters should be implemented for the localisation, which we believe would greatly improve the localisation, due to the possibility that this would limit the cycling between states that are close to each other. It follows that increasing the localisation consistency would also improve navigation as well.

Lastly the experimental results should be confirmed in an underground environment, by distributing RFID tags and attempting to localise and navigate in this practical environment.

9. Bibliography

[1] Micheal W. George, "U.S. Geological Survey, Mineral Commodity Summaries," January 2011. [Online]. https://www.cia.gov/library/publications/the-world-factbook/geos/sf.html

[2] Mining Weekly. (2009, February) Mining Weekly.com. [Online]. http://www.miningweekly.com/article/worlds-new-deepest-mine-safe-cheap-anglogold-2009-02-09

[3] G. R. Adams and A. J. Jager, "Petroscopic observations of rock fracturing ahead of stope faces in deep-level gold mines," vol. 80, no. 06, 1980.

[4] Steven Schultz. (1999, December) Princeton University Website. [Online]. http://www.princeton.edu/pr/pwb/99/1213/microbe.shtml

[5] Nick Wadhams. (2011, February) Wired Magazine March 2011. [Online]. http://www.wired.com/magazine/2011/02/st_ultradeepmines/

[6] James S. Monroe and Reed Wicander, "The Changing Earth: Exploring Geology and Evolution," 1997.

[7] Dineo Matomela. (2011, April) Business Report. [Online]. http://www.iol.co.za/business/rising-sa-mine-deaths-need-urgent-attention-1.1055349

[8] Department of Mineral Resources, "Annual Report 2010/11," Pretoria, 2011.

[9] Christian Forster, Deon Sabatta, Roland Siegwart, and Davide Scaramuzza, "RFID-based hybrid metric-topological SLAM for GPS-denied environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, Ka, 2013, pp. 5228-5234.

[10] Bryan Adams, Cynthia Breazeal, Rodney A. Brooks, and Brian Scassellati, "Humanoid Robots: A new kind of tool," *IEEE Intelligent Systems*, vol. 15, no. 4, pp. 25-31 , 2000.

[11] CSIR. (2011, July) Council for Scientific and Industrial Research (CSIR). [Online]. http://www.csir.co.za/mias/mining.html

[12] Sisa James, Robyn A. Verrinder, Deon Sabatta, and Ali Shahdi, "Localisation and Mapping in GPS-denied Environments using RFID Tags," in *Robotics and Mechatronics Conference of South Africa*, Pretoria, 2012.

[13] Mining Weekly. (2011, August) Mining Weekly.com. [Online]. http://www.miningweekly.com/article/num-calls-for-more-compassion-from-the-industry-when-it-comes-to-mine-fatalities-2011-08-05

[14] CSIR. (2011, January) Council for Scientific Industrial Research (CSIR). [Online]. wwww.csir.co.za/profile_of_csir.html

[15] Rainer Kummerle, Dirk Hahnel, Dmitri Dolgov, Sebastian Thrun, and Wolfram Burgard, "Autonomous Driving in a Multi-level Parking Structure," in *Robotics and Automation, IEEE International Conference on*, Kobe, 2009.

[16] MIT and Olin College, "DARPA Grand Challenge," 2007.

[17] Hugh Durrant-Whyte and Tim Bailey, "Simultaneous localization and mapping: part I," *Robotics & Automation Magazine, IEEE*, vol. 13, no. 2, 2006.

[18] John Markoff. (2010, October) New York Times Reprints. [Online]. http://bngumassd.org/neatstuff/selfdrive%20cars.pdf

[19] Sebastian Thrun. (2012) Udacity. [Online]. http://www.udacity.com/overview/Course/cs373/CourseRev/apr2012

[20] US Goverment. (2012, July) GPS.gov. [Online]. http://www.gps.gov/systems/gps/performance/accuracy/

[21] C. Baker et al., "A Campaign in Autonomous Mine Mapping," in *Robotics and Automation, IEEE International Conference on*, vol. 2, Los Angeles, 2004.

[22] Hayato Kondo and Tamaki Ura, "Navigation of an AUV for investigation of underwater structures," *Control Engineering Practice*, vol. 12, no. 12, pp. 1551–1559, December 2004.

[23] Vassilis Varveropoulos, "Robot Localization and Map Construction Using Sonar Data," The Rossum Project, 2000.

[24] Engineering ToolBox. [Online]. http://www.engineeringtoolbox.com/speed-sound-d_82.html

[25] Wen-jing Zeng, Lei Wan, Tie-dong Zhang, and Shu-ling Huang, "Simultaneous localization and mapping of autonomous underwater vehicle using looking forward sonar," *Journal of Shanghai Jiaotong University (Science)*, vol. 17, no. 1, pp. 91-97, Feb. 2012.

[26] Elliot S. Duff, Peter I. Corke Jonathan M. Roberts, "Reactive navigation and opportunistic localization for autonomous underground mining vehicles," *Information Sciences*, vol. 145, no. 1-2, pp. 127-146, August 2002, http://www.sciencedirect.com/science/article/pii/S002002550200227X.

[27] Sangdo Park and Hongchul Lee, "Self-Recognition of Vehicle Position Using UHF Passive RFID Tags," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 1, pp.

226 - 234, January 2013.

[28] Y. Raoui et al., "RFID-based topological and metrical self-localization in a structured environment," in *International Conference on Advanced Robotics*, Munich, 2009, pp. 1 - 6.

[29] D. Hahnel, W. Burgard, D. Fox, K. Fishkin, and M. Philipose, "Mapping and Localization with RFID Technology," in *International Conference on Robotics and Automation*, vol. 1, Barcelona, 2004, pp. 1015 - 1020.

[30] Charles C. Kemp, and Matthew S. Reynolds Travis Deyle, "Probabilistic UHF RFID tag pose estimation with multiple antennas and a multipath RF propagation model," in *International Conference on Intelligent Robots and Systems*, Nice, 2008, pp. 1379 - 1384.

[31] Sebastian Schneegans, Philipp Vorst, and Andreas Zell, "Using RFID Snapshots for Mobile Robot Self-Localization," EMCR, Tubingen, 2007.

[32] Philipp Vorst, Sebastian Schneegans, Bin Yang, and Andreas Zell, "Self-Localization with RFID Snapshots in Densely Tagged Environments," in *Intelligent Robots and Systems, International Conference on*, Nice, 2008.

[33] Xiaoqin Zhang, "SVD based kalman particle filter for robust visual tracking," in *Pattern Recognition, International Conference on*, Beijing, 2008.

[34] Greg Welch and Gary Bishop, "An Introduction to the Kalman Filter," Chapel Hill, 27599-3175, 2006.

[35] Lindsay Kleeman, "Understanding and Applying Kalman Filtering," in *Proceedings of the Second Workshop on Perceptive Systems*, Perth, 1996.

[36] Daniel Seliger. (2012) State Estimation: Particle Filter. Presentation.

[37] Hans-Peter Kriegel, Peer Kroger, Jorg Sander, and Arthur Zimek, "Density-based clustering," *WIREs: Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 231–240, April 2011.

[38] Trevor Hastie, Robert Tibshirani, and Jerome Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Stanford, California: Springer Series in Statistics, 2008.

[39] Leonard Kaufman and Peter J. Rousseeuw, *Finding groups in data: An Introduction to Cluster Analysis*.: John Wiley & Sons, 2009, vol. 344.

[40] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Knowledge*

Discovery and Data Mining, vol. 96, Portland, 1996, pp. 226-231.

[41] R. Sibson, "SLINK: An optimally efficient algorithm for the single-link cluster method," *The Computer Journal*, vol. 16, no. 1, pp. 30-34, 1973.

[42] Zhexue Huang, "Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values," *Data Mining and Knowledge Discovery*, vol. 2, no. 3, pp. 283-304, September 1998.

[43] Tian Zhang, Raghu Ramakrishnan, and Miron Livny, "An Efficient Data Clustering Method for Very Large Databases," in *International Conference on Management of Data*, New York, 1996, pp. 103-114.

[44] R.T. Ng and Jiawei Han, "CLARANS: A Method for Clustering Objects for Spatial Data Mining," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 14, no. 5, pp. 1003 - 1016, September 2002.

[45] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar, "Cluster Analysis," in *Introduction to Data Mining*.: University of Minnesota, 2006.

[46] Vladimir Estivill-Castro and Jianhua Yang, *PRICAI 2000 Topics in Artificial Intelligence*. Callagahn, Australia: Springer Berlin Heidelberg, 2000, vol. 1886, Fast and Robust General Purpose Clustering Algorithms.

[47] Mathworks. (2013) Mathworks. [Online]. http://www.mathworks.de/de/help/stats/kmeans.html

[48] A. Jain, M. Murty, and P Flynn, "Data Clustering: A Review," *ACM Computing Surveys (CSUR)*, vol. 21, no. 3, pp. 264-323, September 1999.

[49] Junjie Wu, *Advances in K-means Clustering: A Data Mining Thinking*., 2012.

[50] D. Wishart, "Mode analysis: a generalization of nearest," *London and New York Academic Press*, 1969.

[51] Narendra Sharma, Aman Bajpai, and Ratnesh Litoriya, "Comparison the Various Clustering Algorithms of Weka Tools," *International Journal of Emerging Technology and Advanced Engineering*, vol. 2, no. 5, pp. 73-80, May 2012.

[52] Joey Kippen, "Clustering: Knowledge From Understanding," Colorado State University, Fort Collins, Review 2009.

[53] P. Svestka and M. H. Overmars, "Probabilistic Path Planning," in *Robot Motion Planning and Control*, Jean-Paul Laumond, Ed., 1998, ch. 5.

[54] Dave Ferguson, Maxim Likhachev, and Anthony Stentz, "A Guide to Heuristic-based

Path Planning," in *International Conference on Automated Planning and Scheduling*, Monterey, 2005, pp. 9-18.

[55] Steven M. LaValle and James J. Kuffner, "Rapidly-Exploring Random Trees: Progress and Prospects," 2000.

[56] Daniel Delling, "Engineering and Augmenting," Universit¨at Fridericiana zu Karlsruhe , Karlsruhe , Dissertation 2009.

[57] S. Skiena, *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*, 3rd ed.: Addison-Wesley, 1990.

[58] Thomas H. Connen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms*, 2nd ed. London, England: McGraw-Hill Book Company.

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

[60] Charles M. Grinstead and J. Laurie Snell, "Markov Chains: Introduction," in *Introduction to Probability*.: American Mathematical Society, 1998, ch. 11.1, pp. 405-407.

[61] Martin L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, USA: John Wiley and Sons, 2005.

[62] Manuela Veloso, "Reinforcement Learning: Value and Policy Iteration," Carnegie Mellon University: Computer Science Department, Pittsburgh, Lecture Slides 2001.

[63] Elena Pashenkova, Irina Rish, and Rina Dechter, "Value iteration and policy iteration algorithms for Markov decision problem," University of California: Department of Information and Computer Science, Irvine, Paper 1996.

[64] Verena Heidrich-Meisner and Christian Igel, "Evolution Strategies for Direct Policy Search," in *Parallel Problem Solving from Nature - PPSN X*.: Springer Berlin Heidelberg, 2008, pp. 428-437.

[65] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, US: The MIT Press, 1999.

[66] Machine Learning Experiences in Artificial Intelligence. (2009) Value Iteration, Policy Iteration, and Q-Learning. Document. [Online]. http://uhaweb.hartford.edu/compsci/ccli/projects/QLearning.pdf

[67] The MathWorks Inc., *MATLAB 2011b*. Natick, Massachusetts, United States: The MathWorks Inc., 2011.

[68] ALIEN Technology. (2000) Low VSWR/Axial Ratio Antenna ALR-8696-C. Data Sheet.

[69] Morgan Quigley et al., "ROS: an open-source Robot Operating System," in *ICRA workshop on open source software*, 2009, p. 5.

[70] Earle B. Amey, "U.S. GEOLOGICAL SURVEY MINERALS YEARBOOK," 2002.

[71] (2007, Nov.) DARPA Grand Challenge. [Online]. http://www.darpagrandchallenge.com/

[72] James Norris. (2004, September) Markov Chains: Descrete Time Markov Chains. Cambridge University Press: Document. [Online]. http://www.statslab.cam.ac.uk/~james/Markov/

[73] Hongpeng Chi, Kai Zhan, and Boqiang Shi, "Automatic guidance of underground mining vehicles using laser sensors," *Tunnelling and Underground Space Technology*, vol. 27, no. 1, pp. 142-148, January 2012, http://www.sciencedirect.com/science/article/pii/S0886779811001155.

10. Appendix A: Code

Simulation Script

```
% RFID tag Localisation Simulator
% This simualtion places tags in a grid with various orientations
% (note that you can modifiy this to place tags in customised positions and orientations)
close all
clear all
clc
%rng(100)
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%Simulation Parameters
nt = 250;              % Number of Tags
n = 6;                % Grid size
ts = 500;             % Simulation Duration
dt = 0.25;              % Simulation interval
num_actions=5;           % L R F TL TR
veh_state = [0,0,pi];% Initial Vehicle state [x y theta]
thresholdRSS = 500;  % Set the threshold for tag detection
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

vs =0.2;             % Quiver Length
v = 0;               % Vehicle Initial Velocity
w = 0;               % Vehicle Initial Rotational Velocity

% Place tags in grid
fprintf('Initializing Tag Locations \n')

tagx = n*rand(1,nt)-n/2;%[(n/nt):(n/nt):n]-n/2;
tagy = n*rand(1,nt)-n/2;%[(n/nt):(n/nt):n];
tago = pi*rand(1,nt); %ones(1,nt);

tag_list = [tagx;tagy;tago];        % Matrix containing all the tags in the grid

% Initialise Entropy Trace
tplot=[];
iplot=[];

% Start Simulation
fprintf('Running Simulation\n')

%%
% Explore
%veh_state = [rand(1),rand(1),2*pi*rand(1)]; % Place Vehicle
[observations,observationsV] = explore(ts,dt,tag_list,thresholdRSS,veh_state);

%%
% Cluster The States Using k-means
fprintf('Clustering...\n')
num_States = round(length(observationsV)/4)*2-1;
%[IDX, cluster_Centroids] = k_means(observations, num_States);
%[IDX,cluster_Centroids,num_States] = k_hedge(observations',0.25);
[IDX,~, cluster_Centroids] = kmedoids(observations',num_States);
%IDX = 1:size(observations,1);
%cluster_Centroids = (observations);

% Clean Clusters
[IDX, cluster_Centroids,num_States] = clean_clusters(IDX,cluster_Centroids);

nnum = 0;
for j = 1:1:num_States
    num = 0;
    for i=1:length(observationsV)

        if IDX(i) == j % Check if IDX contains j
```

```
%quiver(observationsV(i,1),observationsV(i,2),vs*cos(observationsV(i,3)),vs*sin(observati
onsV(i,3)),'r*');
            %hold on;
            num = num + 1;

        end

    end

    if num ~= 0
        nnum = nnum + 1;
    end

end
fprintf('Valid Clusters: %4.2f %% full\n',100*nnum/num_States)
%%
% Visualise State Clusters
%fprintf('Visualising State Clusters\n')
%k_viz(observationsV,IDX,tag_list)
%%
% Build State Transition Probabilities
fprintf('Building State Transition Probability\n')
steps = length(observations)*50;
veh_state = [rand(1),rand(1),2*pi*rand(1)]; % Place Vehicle
[state_transition] = build_transition(veh_state,tag_list,thresholdRSS, cluster_Centroids,
steps,dt, num_actions);

%fprintf('state transition matrix is %f4.2 %% full\n',1-
100*sum(sum(isnan(state_transition(:,:,1))))/(num_States*num_States))

veh_state = 0.8*[n*rand(1)-n/2, n*rand(1)-n/2, 2*pi*rand(1)];
%%
save 'functional_workspace'
fprintf('Running Value Iterations...\n')
% Choose a random goal state
goal_state = 0.8*[n*rand(1)-n/2, n*rand(1)-n/2, 2*pi*rand(1)];
ss = rssCalc(goal_state,tag_list);
goal_state = localise( cluster_Centroids,ss );
% Initialise Value Iteration Values
reward=num_States;
discount=0.1;
default = -10;
%[ state_Value, nav_policy ] = value_iteration(af,al,ar, goal_state, reward,default,
cost); %Not working properly yet
R = default.*ones(num_States,num_States,num_actions); % Reward Matrix
R(:,goal_state,:) = reward;
[state_Value, nav_policy, ~, ~] = mdp_value_iteration(state_transition, R, discount);
% Navigate
fprintf('Starting Navigation...\n')
%figure
dt=0.05;

% M = getframe(gcf);      % Get current frame

for nn=1:500

    ss = rssCalc(veh_state,tag_list);           % Compute Current RSS for visible tags
    ss(ss<thresholdRSS)=0;                      % Zero non detections

    visible = sum(abs(ss));

    plot(tagx,tagy,'ro')
    grid on
    hold on
    if visible  % if there are visible tags
        quiver(veh_state(1),veh_state(2),vs*cos(veh_state(3)),vs*sin(veh_state(3)),'b*');
    else
        quiver(veh_state(1),veh_state(2),vs*cos(veh_state(3)),vs*sin(veh_state(3)),'k*');
```

```
    end

    % Mark Goal State
    for i=1:size(observationsV,1)
        if(IDX(i)==goal_state)

quiver(observationsV(i,1),observationsV(i,2),vs*cos(observationsV(i,3)),vs*sin(observatio
nsV(i,3)),'g*');
        end
    end

    % Find the nearest state
    if visible
        df = num_States.*ones(1,num_States);
        for i = 1:num_States
            df(i) = norm(cluster_Centroids(i,:)-ss);          % Find Distance to Cluster
Centroids
        end
        [~,minIndex] = min(df);                % Find Closest Cluster Centroid

        % Mark nearest Cluster
        for i=1:size(observationsV,1)
            if IDX(i) == minIndex

quiver(observationsV(i,1),observationsV(i,2),vs*cos(observationsV(i,3)),vs*sin(observatio
nsV(i,3)),'m*');
            end
        end

        %nav_policy(minIndex)

        if minIndex==goal_state
            fprintf('Found Goal!!!!!!!!!!!!!!!! :-D\n')
            [v w] = act('S');

quiver(veh_state(1),veh_state(2),vs*cos(veh_state(3)),vs*sin(veh_state(3)),'g*');
            break
        else
            [v w] = act(nav_policy(minIndex));
        end
    else % If ~visible
        [v w] = act('b'); % Turn around
    end

    veh_state = driveBot(veh_state,v,w,dt);

%     % Grab a frame
%     M(nn) = getframe;

    hold off
    pause(0.1)

end
% movie2avi(M, 'movieNav', 'fps', 30);
% grids=[-n/2,n/2,-n/2,n/2];
% polViz( nav_policy,grids,dt,cluster_Centroids,tag_list )

fprintf('Finished\n')
```

Calculate RSS Values Function

```
function [ ss ] = rssCalc( veh_state, tag_list )
% RSSCALC This function takes in a vehicle state and a list of tags, and returns the
RSS(recieved signal strength) for each tag
%
%   usage: rssCalc( veh_state, tag_list )
%
%   veh_state: vector [x y phi] describing the state of the vehicle in
%              cartesian co-ordinates. phi is the orientation measured
%              positive counter clockwise from the positive x-axis
%
%   tag_list: 3 x n matrix [x1 x2 .. xn   tag characteristics for 'n' tags
%                          y1 y2 .. yn   with co-ordinates x,y
%                          a1 a2 .. an]  orientation of normal vector of the tag in the
global referecnce frame
%
%   ss:       vector [ss1 ss2 .. ssn]recieved signal strength for
%             corresponding tags

vx = veh_state(1);              % Robot/Vehicle x position (global ref frame)
vy = veh_state(2);              % Robot/Vehicle y position (global ref frame)
vo = veh_state(3);              % Robot/Vehicle orientation (global ref frame)
to=tag_list(3,:);               % Tag normal orientation (global ref frame)

% Change to vehicle reference frame (place vehicle at the origin)
tx = tag_list(1,:)-vx;  % Translate tags in the x direction to vehicle reference frame
ty = tag_list(2,:)-vy;  % Translate tags in the y direction to vehicle reference frame
tagpos = [tx;ty];

% Rotate frame to face align x direction (this is required for the PSS function)
relpos = (tagpos'*[cos(vo) -sin(vo);sin(vo) cos(vo)])';    % Rotate tag positions

% Reassign new tag co-ordinates in robot reference frame
txv=(relpos(1,:));      % Tag x co-ordinates after rotation
tyv=(relpos(2,:));      % Tag y co-ordinates after rotation

% Calculate tag normal rel to vehicle
to = atan2(sin(to-vo),cos(to-vo));      % Rotate tag angle (gives tag normal orientation
in vehicle reference frame)
tav=atan2(tyv,txv);                     % Work out angle to the tag relative to the
vehicle (vehicle referenct frame)
tov = atan2(sin(tav-to),cos(tav-to));   % Compute tag normal orientation relative to the
vehicle

%Compute RSS in the Vehicle reference frame
ss=PSS(txv,tyv,tov);

end


function rss = PSS(x,y,ta)
%This function accepts RFID tag location(cartesian co-ordinates) & its orientation &
returns the recieved signal strength
%
%   Usage function rss = PSS(x,y,ta)
%
%   x: the tags' position along the x axis
%   y: the tags' position along the y axis
%   ta: the orientation of the tag relative to the vehicle/robot

n = length(x);
rss = zeros(1,n);
a = [2235 17144 -12101];        % Coeficients of polynomial approximation of order 3
d = sqrt(x.*x + y.*y);          % Distance to tags

ct = abs(cos(atan2(y,x)));      % Compute angle of the tag rel to robot
c = 50;                         % Set attenuation value behind the antenna
```

```
% For each tag calculate the rss using best-fit function based on experimental data
for t=1:n
    if (x(t)>=0)    % If the tag is in front of the antenna
        rss(t) = [exp(-d(t)) exp(-2*d(t)) exp(-3*d(t))]*a'*ct(t);%*abs((cos(ta(t))+1)/2);
    else            % If the tag is behind the antenna (increase attentuation)
        rss(t) = [exp(-c*d(t)) exp(-2*c*d(t)) exp(-
3*c*d(t))]*a'*ct(t);%*abs((cos(ta(t))+1)/2);
    end
end
```

Localise Algorithm Function

```
function [ state ] = localise( cluster_Centroids,ss )
%LOCALISE localise( cluster_Centroids,ss )
%   Detailed explanation goes here

    [num_States,~] = size(cluster_Centroids);
    df = num_States.*ones(1,num_States);
    for i = 1:num_States
        df(i) = norm(cluster_Centroids(i,:)-ss);          % Find Distance to Cluster
Centroids
    end
    [~,state] = min(df);                % Find Closest Cluster Centroid/State

end
```

Explore Function

```
function [observations,observationsV] = explore(ts,dt,tag_list,thresholdRSS,veh_state)
%EXPLORE Summary of this function goes here

%   Detailed explanation goes here


% Initialise Algorithm Variables
index =1;
vs =0.1;            % Quiver Length
numObs = round(ts/dt);
observations = zeros(size(tag_list,2),numObs);
observationsV = zeros(numObs,length(veh_state));
%action = 'f';
tf = round(rand(1));
watchdog = 0;
pv=0.1;


% Plot Grid
%figure

fprintf('Start Exploring\n')
for t=1:dt:ts % Walk around observing states

    % Plot tags
    %hold off
%     plot(tag_list(1,:),tag_list(2,:),'ro')
%     hold on
%     grid on

    if(mod(t,10)==1)
        UpdateProgress(100*t/ts);
    end
    %veh_state = [2*rand(1)-1,2*rand(1)-1,2*pi*rand(1)];

    ss = rssCalc(veh_state,tag_list);           % Compute Current RSS for visible tags
```

```
    ss(ss<thresholdRSS)=0;                          % Zero non detections

    %>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
    % Planning Algorithm Goes Here
    visible = sum(abs(ss));

    if visible  % if there are visible tags
        observations(:,index)=ss;
        observationsV(index,:) = veh_state;

%quiver(veh_state(1),veh_state(2),vs*cos(veh_state(3)),vs*sin(veh_state(3)),'b*');
        for i = 1:length(tag_list)
            if ss(i)
                plot(tag_list(1,i),tag_list(2,i),'go')
            end
        end
    else

%quiver(veh_state(1),veh_state(2),vs*cos(veh_state(3)),vs*sin(veh_state(3)),'k*');
    end

    %Explore using randomly selected actions
    action=round(3*rand(1)+0.5);
    % Choose Action
    [v w] = act(action);

    if ~visible
        action = 'b';
        v=0;
        if tf
            w=5;
        else
            w=-5;
        end
        if watchdog>10
            v=pv;
        end
    %    fprintf('Turning Around\n')
    end
    % Check for looping
    if action~='b'
        watchdog=0;
        pv=0.5/10;
        tf=round(rand(1));
    else
        watchdog=watchdog+1;
        pv=pv+0.1/10;
    end
    % Execute Move
    veh_state = driveBot(veh_state,v,w,dt);
    index = index+1;
    % Algorithm ends here
    %<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

    %pause(0.1)

end
% Remove Empty Observations
fprintf('\nRemoving Empty Observations...\n')
empty = zeros(size(tag_list,2),1);
for i=numObs:1
    question = sum(observations(:,i)==empty);
    if question==size(tag_list,2)
        observations(:,i)=[];
    end
end
observations = observations';
```

```
fprintf('Finished Exploring :-)\n')
end
```

Build Transition Function

```
function [ state_transition ] = build_transition(veh_state,tag_list,thresholdRSS,
cluster_Centroids,steps,dt,num_actions)
%function [ af al ar ] = build_transition(veh_state, cluster_Centroids )
%Makes the robot wander around building a state transition probability matrix for each
action
%   [veh_state] = initial vehicle state (position and orientation)
%   [cluster_Centroids] = centrods of the various state
%   [num_State] =  total number of states
%   [steps] = number of itaration to take to build probability matrix (in the order of
num_States*100)
%   function returns [af al ar] which correponds to the probability
%   of transtioning from on state to another by executing the relevant
%   action

[num_States,~] = size(cluster_Centroids);
%state_transition = sparse(zeros(num_States,num_States,num_actions));
state_transition = zeros(num_States,num_States,num_actions);
%action = 1;
%n=6;
inc = 1;
tf = randi(2,1)-1;
watchdog=0;
pv=0.1;

% % First Find which state we are in..?
% ss = rssCalc(veh_state,tag_list);           % Compute Current RSS for visible tags
% df = ones(1,num_States);
% for i = 1:num_States
%     df(i) = norm(cluster_Centroids(i,:)-ss);          % Find Distance to Cluster
Centroids
% end
% [~,minIndex] = min(df);                % Find Closest Cluster Centroid

%veh_state = [n*rand(1)-n/2, n*rand(1)-n/2, 2*pi*rand(1)];

for nn=1:steps % Wander around Building State transision probability matrix

    % Progress Bar
    if(mod(nn,100)==1)
        UpdateProgress(100*nn/steps);
    end
% %
PLACE><PLACE><PLACE><PLACE><PLACE><PLACE><PLACE><PLACE><PLACE><PLACE><PLACE><PLACE
>
%     veh_state = [n*rand(1)-n/2, n*rand(1)-n/2, 2*pi*rand(1)];
% for direction=0:2*pi/10:2*pi
% veh_state(3)=veh_state(3)+direction;
%
LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE>
    ss = rssCalc(veh_state,tag_list);           % Compute Current RSS for visible tags
    ss(ss<thresholdRSS)=0;                      % Zero non detections
    visible = sum(abs(ss));

    if visible

        %pminIndex=minIndex;

        % Find the current/nearest state
        for i = 1:num_States
            df(i) = norm(cluster_Centroids(i,:)-ss);          % Find Distance to Cluster
Centroids
        end
```

```
        [~,pminIndex] = min(df);                % Find Closest Cluster Centroid Prev
Index

%
LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE>

%            for k=1:5
            %Make a Move>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                    %Explore using randomly selected actions
                    action=randi(num_actions);
                    %action=k;
                    [v w] = act(action);
                    % Execute Move
                    veh_state = driveBot(veh_state,v,w,dt);
            %Make a Move>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

                %
LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE>
222222222222
                    ss = rssCalc(veh_state,tag_list);           % Compute Current RSS for
visible tags
                    ss(ss<thresholdRSS)=0;                      % Zero non detections
                    visible = sum(abs(ss));

                    if visible

                        % Find the current/nearest state
                        for i = 1:num_States
                            df(i) = norm(cluster_Centroids(i,:)-ss);          % Find
Distance to Cluster Centroids
                        end
                        [~,minIndex] = min(df);                % Find Closest Cluster
Centroid Current Index

                        if(pminIndex==minIndex)                % If its in the same state
                            inc = inc/4;
                        else
                            inc = 1;
                        end


state_transition(pminIndex,minIndex,action)=state_transition(pminIndex,minIndex,action)+i
nc; % Increment Transition
                    %veh_state = 0.9*[n*rand(1)-n/2, n*rand(1)-n/2, 2*pi*rand(1)];
                    end
%            end
            %
LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE>
222222222222

     else % ie Outside tag range, then turn around
            action = 'b';
            v=0;
            if tf
                w=5;
            else
                w=-5;
            end
            if watchdog>10
                v=pv;
            end
        % Execute Move
        veh_state = driveBot(veh_state,v,w,dt);
     end

    % Check for looping
    if action~='b'
        watchdog=0;
        pv=0.5;
```

```
        tf=round(rand(1));
    else
        watchdog=watchdog+1;
        pv=pv+0.1;
    end

%end
end

%save('before_norm.mat');
fprintf('\n...Normalising Probability Matrices\n')

% Apply Laplacian Smoothing
%state_transition=state_transition+1;
% Normalising
state_transition = state_transition ./ repmat(sum(state_transition,2),1,num_States);

%[state_transition]=normalise(state_transition);

end
```

Experiment Code

```
% Sisa James
% CSIR 2013
% This is the final script for my experiments and includes running the
% entire experiment

clear
clc

%SETUP AND INITIALISATION
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
% Define Global Variables
global reader,
global ServerAddr;
%global state_transition;

% Control Variables
nActions = 3;
dt=2;      % Time Step
v=0.15;       % Maximum Velocity
n=2;         % Number of scans per observations (avarage reading over n scans)

% Specify Mex Server
ServerAddr='10.42.43.1';

% Create TCP/IP connection to reader
% Specify server machine and port number.
%reader = tcpip_open('146.64.165.29',23);
reader = tcpip('146.64.165.29', 23);

% Set size of receiving buffer, if needed.
set(reader, 'InputBufferSize', 8000);
% set line terminator
%set(reader,'terminator','CR/LF')

% Open connection to the server.
fopen(reader);
% Login to the RFID Reader
fprintf(reader,'alien');
fprintf(reader,'password');
% Set the RFID reader settings
fprintf(reader,'PersistTime = 0');
fprintf(reader,'NetworkTimeout = 65535');
fprintf(reader,'TagListCustomFormat = IDK %i RSSI %m TIME %t;');
fprintf(reader,'TagListFormat=Custom');
readerCommand(reader,['time=',datestr(now,'yyyy/mm/dd hh:MM:ss')])
```

```
%%
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
if exist('idObs.mat','file')
    fprintf('Removing previous Variables from memory...\n')
    delete 'state_transition.mat';
    delete 'idObs.mat';
    delete 'rssObs.mat';
end
%%
% Explore
steps=100;   % Number of Steps in exploration
user = 'y';
while strcmp(user,'y')||strcmp(user,'Y')
[idString,observations] = exploreReal(reader,steps,v,dt,n,nActions);

user = input('Continue Exploring: y/n: ','s');
end
fprintf('Finished Exploring\n')

%%
% Cluster
%shuffle observations
[nTags, nObs] = size(observations);
for o_index=1:nObs
    swap_index = randi(nObs);

    tempID = idString(:,:,o_index);
    tempRSS = observations(:,o_index);

    idString(:,:,o_index) = idString(:,:,swap_index);
    observations(:,o_index) = observations(:,swap_index);

    idString(:,:,swap_index) = tempID;
    observations(:,swap_index) = tempRSS;
end
%run clustering algorithm
fprintf('Clustering..............\t')
[IDX, nClusters] = preprocess( observations, idString, 0.8);
fprintf('Done\n')
%
fprintf('Calculating Centroids....\t')
[centroidID, centroidRSS] = calCentroids( IDX, idString,observations);

if exist('state_transition.mat','file')
    fprintf('Removing previous state_trantision from memory...\n')
    delete 'state_transition.mat';
end

fprintf('Done\n')
save('matlab');

unreachable = [0 0 0 0 0];

%%
[idScan,rss,tags] = scanNetTags(reader,n);
fprintf('Localising..............\t')
state = localise(centroidID, centroidRSS, idScan, rss)
fprintf('Done\n')
%%
clc
nActions=5;
unreachable_old = unreachable;
fprintf('Building State Transition Matrix\n')
if exist('state_transition.mat','file')
    fprintf('loading state transition matrix from memory...\n')
    load('state_transition.mat');
elseif exist('state_transition','var')
    fprintf('continue from state transition matrix...\n')
else
```

```
    fprintf('Creating state transition matrix...\n')
    % Create a laplacian smoothed transition probability matrix
    state_transition = ones(nClusters+1,nClusters+1,nActions);
end
old_st = state_transition;
tSteps = length(observations)*1.5;
state_transition = build_transition_real(reader,centroidID,
centroidRSS,tSteps,dt,nActions,0.15,state_transition);
%clear R;
save('state_transition');

% Exploit Action Symmetry
%load('state_transition.mat');
%state_transition = sym_exp(state_transition,0.7);
%clc

%%
%save('state_transition');
figure(3)
subplot(2,2,1)
imagesc(state_transition(:,:,1))
subplot(2,2,2)
imagesc(state_transition(:,:,2))
subplot(2,2,3)
imagesc(state_transition(:,:,3))
subplot(2,2,4)
imagesc(state_transition(:,:,4))
%

for i=1:5
    fprintf('Old Unreachable States = %i of %i\n',unreachable_old(i),nClusters)
end

for i=1:5
    unreachable(i) = sum(sum(state_transition(:,:,i),1)<nClusters+2);
    fprintf('Unreachable States = %i of %i\n',unreachable(i),nClusters)
end

%%

% Choose a goal state
load('state_transition.mat','state_transition')
goal_state = 166;%round(rand*nClusters);%localise( cluster_Centroids,ss );

fprintf('\n...Normalising Probability Matrices\n')
% Normalising
st = normalise( state_transition);

% Initialise Value Iteration Values
reward=1;
discount=0.1;
default = 0.5;
%[ state_Value, nav_policy ] = value_iteration(af,al,ar, goal_state, reward,default,
cost); %Not working properly yet
R = default.*ones(nClusters+1,nClusters+1,nActions); % Reward Matrix
R(:,nClusters+1,:) = -0.5;%reward;
R(nClusters+1,:,:) = -0.5;%reward;
R(goal_state,:,:) = reward;
R(:,goal_state,:) = reward;
%[state_transition_n]=normalize(state_transition);
fprintf('Running Value Iterations...\n')
[state_Value, nav_policy, it, ~] = mdp_value_iteration(st, R, discount);

%figure(2),grid on,hold on,plot(1:nClusters+1,state_Value,'-r')
%%
% Navigate
fprintf('Starting Navigation...\n')
[success, state_transition] = navigate(goal_state, nav_policy,reader,centroidID,
centroidRSS, state_transition );
```

```
if ~success
    fprintf('Goal Not Reached :-(\n')
end
save state_transition;
save('matlab');

%%
for rows=1:8
start = getPose();
poseData = zeros(72,5)-1;
for i = rows*(9+4+9):rows*(9+4+9)+9
    moveRobot(0.1,0,4); % move 0.4m
    poseData(i,1:3) = (getPose()-start)';
    [idScan,rss,tags] = scanNetTags(reader,n);
    poseData(i,5)=tags;
    fprintf('Localising..............\t')
    poseData(i,4) = localise(centroidID, centroidRSS, idScan, rss);
end
%
pause(1)
%getPose()
fprintf('Turning..............\t')
for j=1:4
    %set_angle(-j*pi/(4),0.005)
    moveRobot(0,-pi/(4*2),2)
    poseData(i+j,1:3) = (getPose()-start)';
    [idScan,rss,tags] = scanNetTags(reader,n);
    poseData(i+j,5)=tags;
    fprintf('Localising..............\t')
    poseData(i+j,4) = localise(centroidID, centroidRSS, idScan, rss);
end
%
for k = 1:10
    moveRobot(0.1,0,4); % move 0.4m
    poseData(i+j+k,1:3) = (getPose()-start)';
    [idScan,rss,tags] = scanNetTags(reader,n);
    poseData(i+j+k,5)=tags;
    fprintf('Localising..............\t')
    poseData(i+j+k,4) = localise(centroidID, centroidRSS, idScan, rss);
end
save('matlab')
moveRobot(0,pi/(4*2),2*2)
moveRobot(0.1,0,4); % move 0.4m
moveRobot(0,pi/(4*2),2*2)
end
poseData


%%
% Disconnect and clean up the server connection.
%
fclose(reader);
%delete(reader);
%clear reader
```

Explore Algorithm Function Implementation

```
function [idObs,rssObs] = exploreReal(alien,steps,vMax,dt,scans,nActions)
%[observations,observationsV] =
exploreReal(readerObject,steps,vMax,dt,scansPerObservation)
%   Detailed explanation goes here

if nargin<6
    nActions=5;
end

mxTags = 20;
```

```
tf=round(rand);
watchdog=0;
pv=0.05;
curr=1;


if exist('idObs.mat','file')
    fprintf('Loading previous observations')
    load('idObs.mat')
    idObs = cat(3, idObs, char(zeros(mxTags,24,steps)));
    load('rssObs.mat');
    curr = size(rssObs,2);
    rssObs = [rssObs zeros(mxTags,steps)];
    size(rssObs,2);
else
    idObs = char(zeros(mxTags,24,steps));
    rssObs = zeros(mxTags,steps);
end
fprintf('Start Exploring\n')
for n=curr:curr+steps % Walk around observing states

    % Update Progess Bar (Not necessary for correct execution)
    if(mod(n,5)==1)
        save idObs;
        save rssObs;
        fprintf('\n')
        UpdateProgress(100*n/steps);
        fprintf('\n')
    end

    % Read Visible Tags
    [id,obs,tags] = scanNetTags(alien,scans);
    % If you can see at least 3 Tags, then you can successfully localise (x,y,theta)
    visible = (tags>3) && (sum(obs)>3500); % tags is (int)

    if visible  % if there are visible tags

%         id
%         size(id)
%         size(obs)
%         tags
%         size(idObs(1:tags,:,n))
        idObs(1:tags,:,n) = id;
        rssObs(1:tags,n)=obs;

        %Explore using randomly selected actions
        action=round(nActions*rand(1)+0.5);
        % Choose Action
        [v w] = actReal(action, vMax,1);
        % Execute Move
        moveRobot(v,w,dt);
        %pause(dt+0.6)      % Wait for bot to complete action + latency

        watchdog=0;
        pv=0.05;
        tf=round(rand(1));

    else % ie no visible tags (then turn around)

        v=0;
        if tf
            w=pi/4;
        else
            w=-pi/4;
        end
        if watchdog>10
            v=pv;
            pv=pv+0.1/10;
        end
```

```
        watchdog=watchdog+1;
        fprintf('Turning Around\n')
        % Execute Move
        moveRobot(v,w,dt);
        %pause(dt+0.25)      % Wait for bot to complete action + latency
    end

end

% % Remove Empty Observations
% fprintf('\nRemoving Empty Observations...\n')
% empty = zeros(mxTags,1);
% for i=steps:1
%     question = sum(rssObs(:,i)==empty);
%     if question==size(tag_list,2)
%         rssObs(:,i)=[];
%     end
% end
% rssObs = rssObs';

fprintf('Finished Exploring :-)\n')
end
```

Preprocess Algorithm (Custom Clustering) Implementation

```
function [ index,centroids] = preprocess( obs, idString, nObs, nTags, match)
%PREPROCESS Summary of this function goes here
%   Detailed explanation goes here

if nargin<4
    match=0.70;
end

index = 1:nObs;
IDmatch=0;
RSSdiff=0;
centroids = -1.*ones(2,nObs,nTags);
%centCount=0;
count=0;

for observation1=1:nObs-1

    for observation2=observation1+1:nObs

        if index(observation2)>observation1 && index(observation1)>=observation1 % ie not
already clustered

            for iTg=1:nTags

                count=count+1;
                for jTg=1:nTags

                    %Check ID match
                    if
strcmp(idString(iTg,:,observation1),idString(jTg,:,observation2,:))
%obs(1,observation1,iTg) == obs(1,observation2,jTg)

                        IDmatch=IDmatch+1;

                        RSSdiff = RSSdiff + abs(obs(2,observation1,iTg)-
obs(2,observation2,jTg));

                        %break

                    end % if ID match

                end % for j
```

```
            end % for i

            % Compute Similarity Score
%             observation1
%             observation2

            IDsc = IDmatch/sum(obs(1,observation1,:)~=0)

            RSSdiff/sum(obs(2,observation1,:))

            RSSsc = 1-RSSdiff/sum(obs(2,observation1,:))

            score = 0.5*(IDsc + RSSsc)
            match

            % Check if they match
            if 0.5*(IDsc + RSSsc) >=match % ie they match

                index(observation2)=index(observation1);
                fprintf('changed')
                %centroids(1,);
            else
                fprintf('Not changed')
            end

            pause

            IDmatch=0;
            RSSdiff=0;

        end % if index

    end % for observation2

end % for observation1


end % Function End
```

Calculate Centroids Function (calCentroids)

```
function [ centroidID, centroidRSS ] = calCentroids( IDX, idObs,rssObs, match)
%CALCENTRIODS Computes the cluster centroids
%   [ centroidID, centroidRSS ] = calCentroids( IDX, idString,observations)
%
%   See also preprocess, readData.

if nargin<4
    match=0.5;
end

[nTags, ~] = size(rssObs);
nClusters = max(IDX);

centTags = round(nTags*(2-match));
centroidID = char(zeros(centTags,25,nClusters));
centroidRSS = zeros(centTags,nClusters);

% Find All Unique IDs in each cluster
for cluster=1:nClusters

    cTag=1; % reset centroid tag stack
    clusterIDs = idObs(:,:,IDX==cluster);

    for obsInClust=1:size(clusterIDs,3)

        tag=1;
        % while tag is not empty & while we havent checked all the tags
```

```
        while tag<=nTags && ((clusterIDs(tag,1,obsInClust)-0) > 42) %&&
((centroidID(cTag,1,cluster)-0)>42)
            unique=1;
            for i=1:cTag
                % if cluster ID already exists in centroidID
                if strcmp(centroidID(i,1:24,cluster),clusterIDs(tag,:,obsInClust))
                    unique=0;
                    centroidID(i,25,cluster) = centroidID(i,25,cluster)+1;
%                     centroidID(i,25,cluster)*1
%                     i,cluster
                    break;
                end
            end

            if unique
                centroidID(cTag,1:24,cluster) = clusterIDs(tag,:,obsInClust);
                cTag=cTag+1; % increment centroid tag stack
            end % if unique

            tag=tag+1; % increment tag location along the entire cluster
        end % while idString

    end % for Each Observation in the Cluster

    % Remove Outliers
    for t = 1:centTags
        if centroidID(t,25,cluster)<0.6*size(clusterIDs,3)
            % Remove tag ID (and consequently the RSS) from centroid
%             0.8*size(clusterIDs,3)
%             centroidID(t,25,cluster)*1
%             pause
            for ix=t:centTags-1
                centroidID(ix,:,cluster) = centroidID(ix+1,:,cluster);
            end
        end
    end
end % for all clusters

%centroidID(t,25,:)*1

% Delete Occurance Column
centroidID(:,25,:)=[];

% Find Average RSS for each centroid
for cluster=1:nClusters

    clusterIDs = idObs(:,:,IDX==cluster);
    clusterRSS = rssObs(:,IDX==cluster);

    tag=1;
    % For all tags in centroid
    while tag<=nTags && (centroidID(tag,1,cluster)-0)>42

        centCount=0;

        for obsInClust=1:size(clusterIDs,3)

            cTag=1;
            while cTag<=nTags && (clusterIDs(cTag,1,obsInClust)-0)>42

                % if centroidID matches Observation in Cluster
                if strcmp(centroidID(tag,:,cluster),clusterIDs(cTag,:,obsInClust))


centroidRSS(tag,cluster)=centroidRSS(tag,cluster)+clusterRSS(cTag,obsInClust);

%                     cluster
%                     tag,cluster
%                     centroidID(tag,:,cluster)
```

```
%                       cTag,obsInClust
%                       clusterIDs(cTag,:,obsInClust)
%                       centroidRSS(tag,cluster)
%                       pause

                    centCount=centCount+1;

                end

                cTag=cTag+1;
            end


        end % for Each Observation in the Cluster
%         cluster
%         clusterRSS
%         clusterIDs
%         centroidRSS(tag,cluster)
%         centCount
        centroidRSS(tag,cluster)=centroidRSS(tag,cluster)./centCount;
        tag=tag+1;

    end % while: all tags in Centroid

end % for all clusters


end % Function End
```

Scan Visible Tags Function (ScanNetTags)

```
function [idReturn, rssReturn, maxTags] = scanNetTags(t,n)
%[IDstring, RSS, nTags] = scanNetTags(readerObj, numOfScansToAverage)
%   Detailed explanation goes here

%n=1;
RSS = zeros(1,1); % ID&RSS
IDstring = char(zeros(1,24,n)); % # Tags, 24 Char ID
%ID eg. E200 9033 1317 0087 0490 E133
maxTags=0;
maxScan=1;

% fclose(t);
% fopen(t);

if strcmp(t.status,'closed')
    fprintf('Attempting to re-establish t communications...')
    set(t, 'InputBufferSize', 5000);
    fopen(t);
    fprintf(t, 'alien');
    fprintf(t, 'password');
    % Set the RFID t settings
    fprintf(t,'PersistTime = -1');
    fprintf(t,'NetworkTimeout = 65535');
    fprintf(t,'TagListCustomFormat = IDK %i RSSI %m TIME %t;');
    fprintf(t,'TagListFormat=Custom');
    readerCommand(t,['time=',datestr(now,'yyyy/mm/dd hh:MM:ss')])
end

%clearReaderBuffer(t)
flushinput(t)

for scan=1:n


    % Send Read tags command
    fprintf(t, 't');
```

```
    % Wait for tag Data
    dataLine = fgetl(t);
    count=1;
    while sum(find(dataLine=='K'))<=0 && sum(find(dataLine=='>'))<=0 && count<3
        dataLine = fgetl(t);
        count = count+1;
    end

    tagCounter=0;
    % While there is still tag data read all of it and store it
    while ischar(dataLine) && sum(find(dataLine=='K'))>0

        tagCounter = tagCounter + 1; % increment # of tags in scan
        if tagCounter>maxTags
            maxTags = tagCounter;
            maxScan = scan;
        end

        % Find Delimiters
        delimIDx = find(dataLine == 'K');
        delimRSSx = find(dataLine == 'T');

        % Extract tagID
        id = dataLine(delimIDx(1)+2:delimIDx(1)+2+28);
        id(isspace(id))=[]; % Remove Spaces
        % Store tag-ID
        IDstring(tagCounter,:,scan)=id;

        % Extract RSS
        rssString = dataLine(delimIDx(1)+2+28+6:delimRSSx(1)-1);
        RSS(tagCounter,scan) = str2double(rssString); % Store Tag RSS

        % Get the next line
        %dataLine = fscanf(t);
        dataLine = fgetl(t);
        %dataLine = char(fread(t));
        %[m,~,~] = fread(t);
        %dataLine = char(m)';
        % reply = [reply word];

    end

end

% averageScan
idReturn=IDstring(:,:,maxScan);
rssReturn = RSS(:,maxScan);

for t=1:maxTags
    match=1;
    sc=1;
    while sc<=n && sc~=maxScan
        totalTags=size(IDstring(:,:,sc),1);
        for tagN=1:totalTags
            if strcmp(idReturn(t,:),IDstring(tagN,:,sc))
                rssReturn(t)=rssReturn(t)+RSS(tagN,sc);
                match=match+1;
                break
            end
        end
        sc=sc+1;
    end
    rssReturn(t)=rssReturn(t)/match;
end

rssReturn(rssReturn<1500)=0;
fprintf('Visible Tags: %i\n',maxTags);
```

```
end
```

Build Transition Algorithm Function in Implementation

```
function [ state_transition, ntr ] = build_transition_real(alien,centroidID,
centroidRSS,steps,dt,nActions,vMax,state_transition)
%BUILD TRANSITION [ state_transition, ntr ] = build_transition_real(alien,centroidID,
centroidRSS,steps,dt,nActions)
%Makes the robot wander around building a state transition probability matrix for each
action
%   alien = alien reader tcpi/ip object
%   centroidID = ID char matrix of centroids (numTags-by-tagIDcharLength(24)-by-
numStates)
%   centroidRSS = RSS double matrix of RSS values (numTags-by-numStates)
%   steps = number of iterations to build probability matrix (in the order of
num_States*10)
%
%   function returns
%   state_transition which correponds to the probability
%   of transtioning from one state to another by executing the relevant
%   action(dim-3) size is (numStates-by-numStates-by-numActions)
%   [ntr] = number of new transitions (an indicator of entropy/unexplored space)

num_States = size(centroidRSS,2);

if nargin<7
    vMax = 0.15;
    state_transition = zeros(num_States+1,num_States+1,nActions);
elseif nargin<8
    state_transition = zeros(num_States+1,num_States+1,nActions);
end

%state_transition = sparse(zeros(num_States,num_States,nActions));
%state_transition = zeros(num_States,num_States,nActions);
inc = 1;
scans=3;
tf=round(rand);
watchdog=0;
pv=0.05;
ntr=0;

%figure(3)
for nn=1:steps % Wander around Building State transision probability matrix

    % Progress Bar
    if(mod(nn,5)==1)
        UpdateProgress(100*nn/steps);
        save state_transition;
    end

    % Read Visible Tags
    [id,obs,tags] = scanNetTags(alien,scans);
    % If you can see at least 3 Tags, then you can successfully localise (x,y,theta)
    visible = (tags>2) && (sum(obs)>3500); % tags is (int)

    if visible

        % LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE>
        pState = localise(centroidID, centroidRSS, id, obs);
        %pState=cState;

        % Choose Random Action
        action=actGen();
        %action=round(rand*nActions+0.5);
        if round(rand+0.35) % 85% of the time choose the least occured action
            [~,action]=min(sum(state_transition(pState,:,:),2));
        end
        [v w] = actReal(action,vMax);
```

```
        % Execute Move
        % MOVE><MOVE><MOVE><MOVE><MOVE><MOVE><MOVE><MOVE><MOVE><MOVE><MOVE>
        moveRobot(v,w,dt);
        %pause(dt+0.6)      % Wait for bot to complete action + latency
        % Reset Watchdog
        watchdog=0;
        pv=0.05;
        tf=round(rand(1));

        % Read Visible Tags
        [id,obs,tags] = scanNetTags(alien,scans);
        % If you can see at least 3 Tags, then you can successfully localise (x,y,theta)
        visible = (tags>2) && (sum(obs)>3500); % tags is (int)

        if visible

            % LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE><LOCALISE>
            cState = localise(centroidID, centroidRSS, id, obs);
            if cState==pState
                inc=inc/2;
            else
                inc=1;
                if ~state_transition(pState,cState,action)
                    ntr = ntr+1;
                    fprintf('New Transision: %f \tState: %f to State: %f \tAction:
%f',ntr,pState,cState,action)
                end
            end
            % Update State Transition Matrix

state_transition(pState,cState,action)=state_transition(pState,cState,action)+inc; %
Increment Transition

        else

state_transition(pState,num_States+1,action)=state_transition(pState,num_States+1,action)
+inc; % Increment Transition To Empty
        end

    else % ie no visible tags (then turn around)

        v=0;
        if tf
            w=pi/4;
        else
            w=-pi/4;
        end
        if watchdog>5
            v=pv;
            pv=pv+0.01;
        end
        fprintf('Turning Around\n')
        % Execute Move
        moveRobot(v,w,dt);
        %pause(dt+0.6)      % Wait for bot to complete action + latency
        % Increment Watchdog
        watchdog=watchdog+1;
    end
end

%save('before_norm.mat');
% fprintf('\n...Normalising Probability Matrices\n')
% % Apply Laplacian Smoothing
% state_transition=state_transition+1;
% % Normalising
% state_transition = state_transition ./ repmat(sum(state_transition,2),1,num_States);

%[state_transition]=normalise(state_transition);
ntr
```

```
end
```

Navigation Algorithm Implementation Function

```
function [ found,state_transition ] = navigate(goal_state, nav_policy,reader,centroidID,
centroidRSS, state_transition )
%[found, state_transition] = navigate(goal_state, nav_policy, reader, state_transition )
%   Detailed explanation goes here
%Attempts to Navigate to a desired goal state implemented on the robot

nStates = length(nav_policy);

if nargin < 4
    state_transition = ones(nState,nState,max(nav_policy));
end

tf=0;
pv=0.05;
watchdog=0;

found = 0;
cState = 0;
pState = 0;
inc=1;
dt=2;
%tcount=0;
%action=1;

for nn=1:50

    [id,obs,tags] = scanNetTags(reader,3);

    visible = (tags>2) && (sum(obs)>3500); % tags is (int)

    % Find the nearest state
    if visible

        pState = cState;
        % Localise the vehicle
        cState = localise(centroidID, centroidRSS, id, obs);

        if cState==goal_state
            found=1;
            fprintf('Found Goal!!!!!!!!!!!!!!!! :-D\n')
            break
        else
            action=nav_policy(cState);
            [v w] = actReal(action);

            if pState % ie Transition is valid
                if cState==pState
                    inc=inc/2;
                else
                    inc=1;
                    if ~state_transition(pState,cState,action) % ie transition is 0
                        fprintf('New Transision! State: %f to State: %f \tAction:
%f\n',pState,cState,action)
                    end
                end
                % Update State Transition Matrix

state_transition(pState,cState,action)=state_transition(pState,cState,action)+inc; %
Increment Transition
                fprintf('State: %f to State: %f \tAction: %f\n',pState,cState,action)
            end

        end
```

```
        % Execute Move
        moveRobot(v,w,dt);
        watchdog=0;
        pv=0.05;
        tf=round(rand(1));
        %pause(dt+0.5)      % Wait for bot to complete action + latency

    else % If ~visible

        if ~watchdog && pState % Increment only once
            cState = 0;
            % Increment Transition To Empty

state_transition(pState,nStates,action)=state_transition(pState,nStates,action)+inc;
        end

        action = 'b';
        v=0;
        if tf
            w=pi/4;
        else
            w=-pi/4;
        end
        if watchdog>10
            v=pv;
        end
        fprintf('Turning Around\n')
        % Execute Move
        moveRobot(v,w,dt);
        % Incriment Counter
        watchdog=watchdog+1;
        pv=pv+0.1/10;
        %pause(dt+0.5)      % Wait for bot to complete action + latency
    end
    % Check for looping
if mod(nn,10)
    save('state_transition');
end

end

save('state_transition');
end
```

Localisation Algorithm Function Implementation

```
function [ state ] = localise( idCentroids,rssCentroids,scanID, scanRSS )
%LOCALISE [ state ] = localise( idCentroids,rssCentroids,scanID, scanRSS )
%   Detailed explanation goes here

[nTags,clusters] = size(rssCentroids);
[snTags,~]=size(scanRSS);
IDscore = zeros(1,clusters);
rssDiff = zeros(1,clusters);

% Start Scoring

% for each centroid
for cCount=1:clusters

    idMatch=0;

    maxTags = 1;
    sTag=1;
    % for each scan tag
    while sTag<=snTags && (scanID(sTag,1)-0)>42

        cTag=1;
```

```
        % for each tag in Centroid/Cluster
        while cTag<=nTags && (idCentroids(cTag,1,cCount)-0)>42

            % if tag IDs match
            if strcmp(idCentroids(cTag,:,cCount),scanID(sTag,:))
                idMatch = idMatch+1;
                rssDiff(cCount) = rssDiff(cCount) + abs(rssCentroids(cTag,cCount)-
scanRSS(sTag));
            end % if tagID match

            cTag=cTag+1;

            if cTag>maxTags
                maxTags = cTag;
            end
        end % for each tag in Centroid/Cluster

        sTag=sTag+1;
        if sTag>maxTags
            maxTags = sTag;
        end
    end % for each tag in the scan

    IDscore(cCount) = (idMatch)/maxTags;%./snTags) + (rssDiff);
%     fprintf('Cluster: %f\tScore: %f\n',cCount,score(cCount))
%     fprintf('ID Match: %f\tRSSsc: %f\n\n',idMatch,rssDiff)

end % for each Custer/Centroid (cCount)
% Eliminate Low ID scores from RSSscore (difference)
maxScore=max(IDscore);
for i=1:clusters
    if IDscore(i) < 0.7*maxScore
        rssDiff(i)=Inf;
        IDscore(i)=0;
    end
end
% Find minimum non-zero RSS Difference
minDiff = 50000;
for i=1:clusters
    if rssDiff(i)~=0 && rssDiff(i)<minDiff
        minDiff = rssDiff(i);
    elseif rssDiff(i)==0
        rssDiff(i)=Inf;
    end
end
%
for i=1:clusters
    if rssDiff(i) > 1.3*minDiff
        %rssDiff(i)=Inf;
        IDscore(i)=0;
    end
end
[maxID,state]=max(IDscore);
if maxID==0
    fprintf('Failed to Localise...\n')
end

end
```