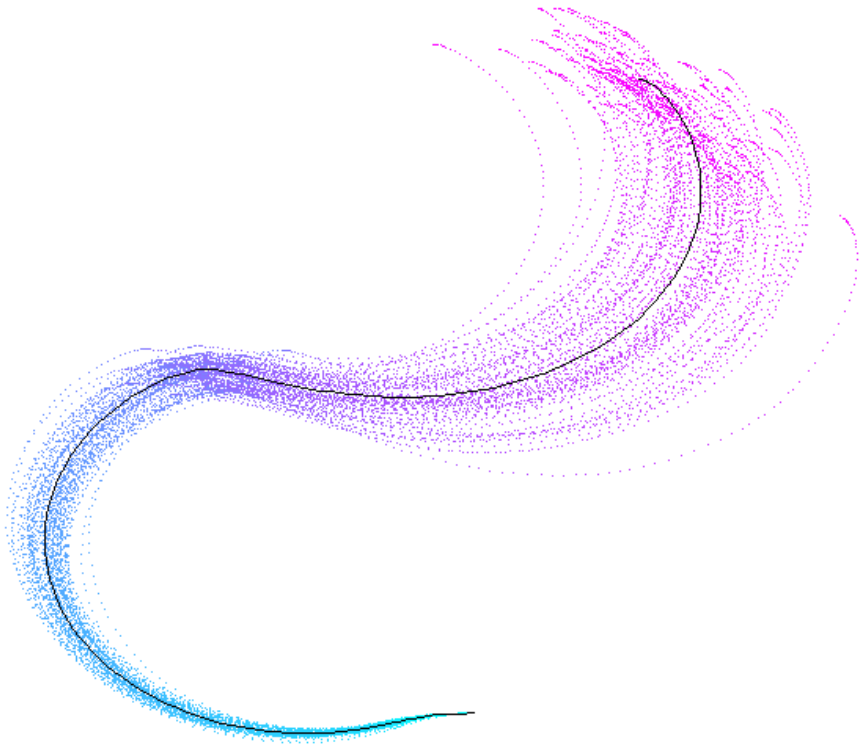
PROBABILISTIC ROBOTICS



Author: Francis Poole

Candidate Number: 18752

The University of Sussex - School of Engineering and Informatics

Project Supervisor: Dr Christopher Buckley

DECLARATION OF ORIGINALITY

This report is submitted as part requirement for the degree of Bachelor of Science in Computer Science and Artificial Intelligence at the University of Sussex. It is the product of my own labour except where indicated in the text. The report may be freely copied and distributed provided the source is acknowledged.

Signature:

Francis Poole

Date:

ACKNOWLEDGEMENTS

I would like to thank Dr Christopher Buckley, University of Sussex, for introducing me to this topic of robotics and for his time, input and support during the project. I would also like to thank my girlfriend Elizabeth for her patience and kindness while I wrote this report and to my parents, Tony and Yvonne, for allowing me to repurpose their dinning room when a space to test the robot was needed.

SUMMARY

The field of robotics is now a huge commercial industry and its applications are growing each year. Locating robots in the real world, known as localisation behaviour, plays a large part in the current research done in this field, as noise in the world causes sensors to return the wrong values and motors to turn at the wrong speeds. This dissertation explains the process of implementing an off-the-shelf Extended Kalman Filter (EKF) both in simulation and on a purpose built, real world robot to accurately locate it as it move across a noisy environment. This produced highly accurate localisation behaviour in simulation by using the EKF's ability to combine different predictions of where the robot is in the world, producing a more accurate and certain belief. Transferring this to a real robot however produced a number of challenges and problems, some of which were never overcome due to the EKF's limitations. Moderate localisation was seen on the real world robot however proposed extensions would be expected to better this result.

CONTENTS

Declaration of Originality ... 3

Acknowledgements ... 4

Summary ... 5

1 Introduction ... 8
- **1.1 Project Aim** ... 8
- **1.2 The Power of Robotics** ... 8
- **1.3 An Uncertain World** ... 8
- **1.4 Probabilistic Robotics** ... 9
  - Recursive State Estimation ... 10
  - Extended Kalman Filter ... 12
  - Other Probabilistic Models ... 12
  - Simultaneous Localisation And Mapping ... 12
- **1.5 Approach** ... 12
- **1.6 Overview** ... 13

2 Professional Considerations ... 14
- **2.1 Public Interest** ... 14
- **2.2 Professional Competence and Integrity** ... 14
- **2.3 Duty to Relevant Authority** ... 14

3 Requirements Analysis ... 15
- **3.1 System Requirements** ... 15
  - Functional ... 15
  - Non-Functional ... 15
- **3.2 User Requirements** ... 16
  - Simulation ... 16
  - Hardware ... 16

4 Software Simulation ... 17
- **4.1 Agent Configuration** ... 17
- **4.2 Body** ... 17
  - Body Pose ... 17
  - Body motion model ... 17
- **4.3 Brain** ... 18
  - Brain Pose ... 18
  - Brain Motion Model ... 18
  - Brain Measurement Model ... 20
  - Extended Kalman Filter Algorithm ... 22
- **4.4 Problems encountered** ... 24
- **4.5 Calibration** ... 24

5 Simulation Results ... 25
- **5.1 Motion Model** ... 25
- **5.2 EKF** ... 28
  - One Feature ... 28

Two Features ... 31
Three Features ... 34
5.3 Comparison ... 37
5.4 Evaluation ... 38
Limitations ... 38
6 Transferring to Hardware ... 39
6.1 Robot Configuration ... 39
Arduino ... 40
Raspberry Pi ... 40
Communication ... 40
Movement ... 40
Getting Motion Command ... 40
Get Sensory Measurement ... 41
6.2 Map of the Environment ... 45
6.3 Transferring EKF ... 45
6.4 Problems Encountered ... 45
6.5 Calibration ... 45
7 Hardware Results ... 46
7.1 Comparison ... 46
7.2 Evaluation ... 50
Limitations ... 50
8 Conclusion ... 52
9 Bibliography ... 53
10 Appendices ... 54
10.1 Meeting Log ... 54

1 INTRODUCTION

1.1 PROJECT AIM

This dissertation aims to explain the process of implementing localisation behaviour on a differential-drive robot, where the robot will be able to accurately locate its position while roaming a noisy environment. This localisation is to be achieved through the use of an off-the-shelf Extended Kalman Filter (EKF) provided by Thrun et al. (2006), which recursively combines two unimodal Gaussian predictions of its position and orientation in the world. The first prediction is a belief distribution modelled by the velocity of the robot over a small time period, the second is a probability density modelled by sensory measurements to nearby landmarks in the world. This combining of two different predictions should give a more accurate prediction of location than either has on its own.

After being first implemented and analysed on an agent in simulation, the EKF algorithm will be transferred to a hardware implementation on a purpose built mobile robot, allowing for analysis of the algorithms' proficiency in the real world. During localisation on the hardware implementation, motion will be modelled by using wheel encoders to measure the number of wheel spins. Feature based sensory measurements of the environment are to be made using multiple light sensors attached to the body of the robot, determining the range and bearing to a single light source in the world. It is hoped that the sensory measurement can then be extended through the use of a single panoramic 360-degree lens attached to a camera, determining a more accurate range and bearing to multiple coloured landmarks in the world.

1.2 THE POWER OF ROBOTICS

Robotics is the science of computer-controlled devices that manipulate the physical world through motors and actuators, coupled with the aid of sensors to perceive its surroundings (Thrun et al., 2006). The 20th and 21st centuries have given rise to startling new developments in this field, from autonomous cars that are able to effectively navigate the chaotic traffic systems of our cities, to rovers that are autonomously exploring uninhabited planets millions of miles away. Our manufacturing industries now heavily depend on robotic arms to work within microscopic levels of precision and with extreme speed, while autonomous Unmanned Aerial Vehicles (UAVs) are more frequently being used in civil and military applications to fight fires (CNN, 2012), or gather intelligence. However the need for accurate localisation in many of these devices is extremely high. Incorrectly determining the position of a car could lead to traffic accidents, while lost rovers could become stuck and rendered useless. The precision of robotic arms relies on knowledge of their position and UAVs travelling across large distances need to know if they have reached their target.

Even greater amounts of funding is also being pushed into the field as increasing numbers of consumers are becoming accustomed to the presence of robotics in their day-to-day lives, with autonomous vacuum cleaners making their way into homes across the world, while intelligent therapeutic robots are being used in hospitals and nursing homes to help comfort distressed patients suffering from dementia (The Guardian, 2014).

1.3 AN UNCERTAIN WORLD

A major obstacle that all robotic systems face is that they have to operate in the presence of the uncertainty that exists in the physical world (Murphy, 2000) the means to negate this noise is a complex and challenging task that is still being tackled today. A robot's world is "inherently unpredictable" (Thrun et al., 2006) due to working in an environment where almost nothing is directly controllable or calculable – people and animals in the world have minds of their own. In

addition to this, a robot's own sensing and actuation is inherently noisy and sometimes even broken, leading to uncertainty in any action the robot might take. Although great strides have been made to lower the noise in both sensors and actuators, even minute amounts of noise can create extreme impacts on a robot's behaviour over longer periods of time. All software models are therefore only abstractions of the real world and have to approximate or "best guess" their actions.

To show how even the smallest amounts of noise in a system can give extreme results, figure 1 below shows the random path of a robot (black line) as it moves around a noise-free 2-dimensional world. This can be compared with 100 generated paths (coloured dotted lines) that are given the same motion control but where some small amount of noise is added to their movement. It is clear to see that as the robot moves from its starting point, at the top right of the figure, the spread of possible positions the noisy robot could arrive at grows increasingly larger, lowering the probability that the noisy robot is at the same position as the noise-free model. This can also be seen in figures 2 and 3 where, as time goes on, the distance (blue line) between the noisy (red line) and noise free (black line) robots increases. Code for these models was adapted from the motion model provided by Thrun et al. (2006).

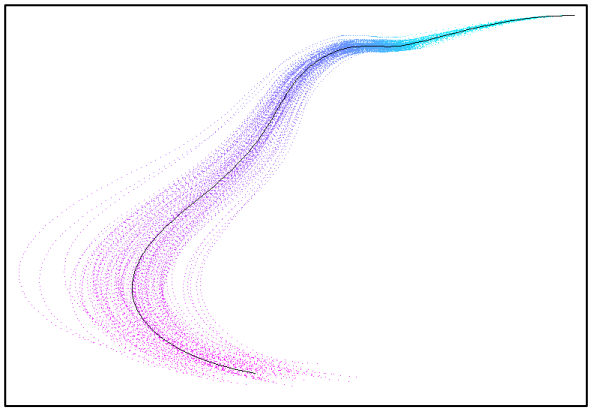


FIGURE 1 - NOISE FREE ROBOT'S PATH (BLACK LINE) AGAINST 100 POSSIBLE NOISY PATHS (COLOURED DOTS)

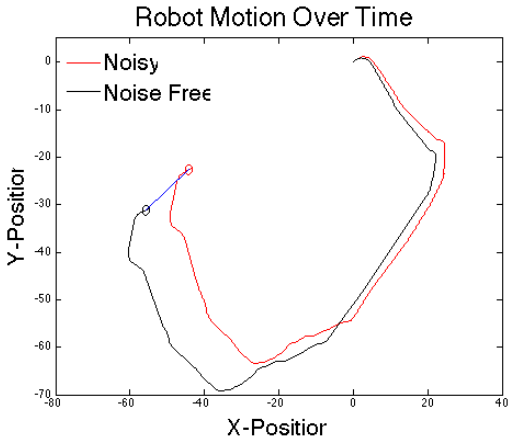


FIGURE 2 - NOISY AND NOISE FREE ROBOT'S MOTION OVER TIME

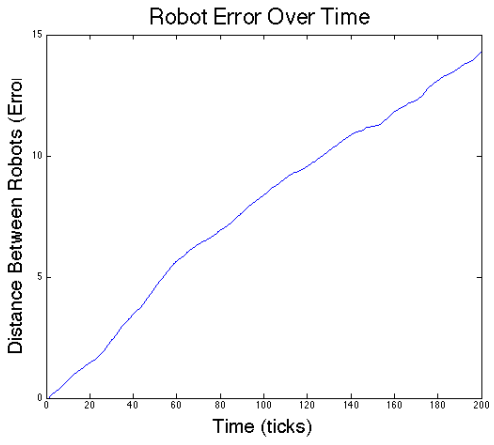


FIGURE 3 - NOISY ROBOT'S ERROR (DISTANCE FROM NOISE FREE ROBOT) OVER TIME

1.4 PROBABILISTIC ROBOTICS

Probabilistic robotics is a relatively new and growing approach to the field. It bestows robots with a high level of robustness in real world situations by using probability theory to represent

the uncertainty of the world explicitly. This allows for ambiguity and a "degree of belief in a mathematically sound way" (Thrun et al., 2006). Its framework is based on an understanding that any prediction of a robot's state is incomplete, this is because models of its motion in the world and models produced by sensory measurement to nearby landmarks in the world are both noisy and limited. But by "fusing" (Murphey, 2000) these motion models and sensory measurements recursively over time, a more intelligent and accurate estimation of its state can be produced. While this recursive state estimation works well on complex real world problems such as localisation, it also drastically increases the computational complexity as entire belief distributions must be considered instead of a single best guess (Thrun et al., 2006). It can also be difficult to compute a robot's uncertainty, sometimes given approximations of noise in the world are too crude and a more complicated model is needed to achieve an accurate estimate.

Recursive State Estimation

Recursive state estimation, sometimes called the Bayes filter, is the basis of probabilistic robotics and facilitates this recursive combination of belief distributions. An initial state estimate is first made predicating the initial state x_{t-1} of the robot. A prediction step is then made after being given some motion control u_t, producing a belief of its new state x_t after performing the motion. The measurement update step then combines x_t with the probability density of its state given that some measurement z_t was observed, producing a final and more accurate belief of its state. It should be noted that this algorithm is modelled in discrete time measurements of t. The recursive nature of the algorithm means it makes the Markov assumption: that previous beliefs do not need to be stored in memory as the state is a complete summary of the past. It is worth noting that unmodelled dynamics in the world, such as people moving, can cause small problems here, although in practice the algorithm is robust and not affected by such problems (Thrun et al., 2006). As all information given to it is used, regardless of precision, it can also be said to be an optimal combination of models. While multiple pieces of information can be combined, with each leading to a more accurate prediction, only two models will be used for this project.

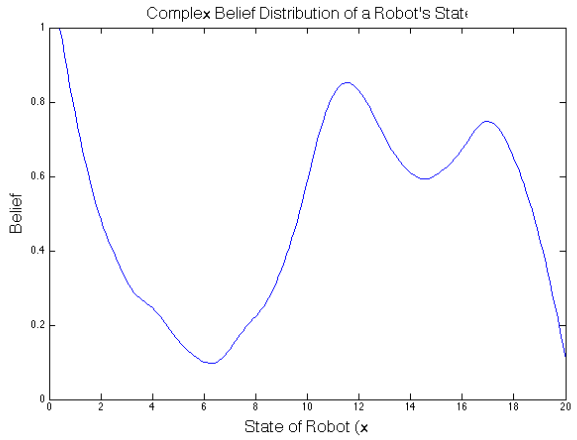


FIGURE 4 - COMPLEX CONDITIONAL PROBABILITY DISTRIBUTION OF A ROBOT'S STATE X, WHERE THE BELIEF OF EACH POSSIBLE VALUE OF X IS SHOWN.

Kalman Filter

As can be easily seen in figure 4, the Bayes filter is intractable due to its use of complex belief distributions. A popular approach is to instead use a linear unimodal Gaussian approximation of the distribution as the belief giving what is known as a Kalman Filter (Kalman, 1960). By simplifying the belief to just a mean and covariance some information is lost, however adequate accuracy is still expected as long as this linearization is performed with an appropriate algorithm. A Gaussian distribution also has the added benefit that the noise in the system is represented Normally, a usual practice in engineering (Maybeck, 1979).

Figure 5 below shows the initial position estimate given a linear Gaussian prior belief of being in some state x.

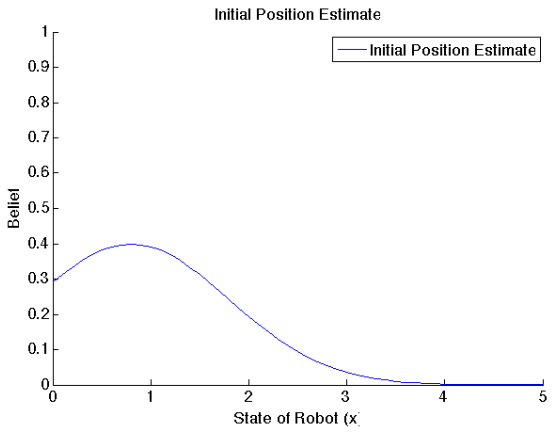


FIGURE 5 – INITIAL STATE ESTIMATE OF A ROBOT

Figure 6 shows the prediction step, where a new prediction of the robots state is made after some motion. Due to uncertainty in the world this Gaussian can be seen to have a larger variance.

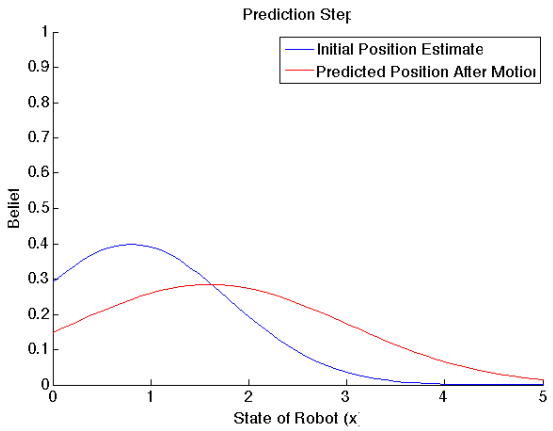


FIGURE 6 – PREDICTION STEP

Figure 7 shows the measurement update step where some measurement is made giving probability density of the robot's state, this is then combined with the predicted position to produce a new more certain belief.

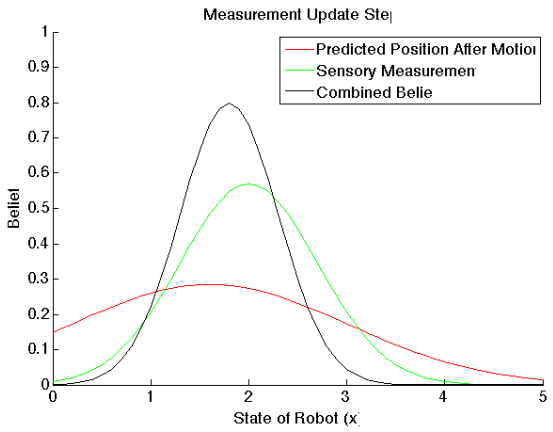


FIGURE 7 – MEASUREMENT UPDATE STEP

Extended Kalman Filter

Two main versions of the Kalman filter exist, the first being the Extended Kalman Filter (EKF). This version relaxes the assumption that the state transition and measurement made by the Kalman filter are linear (Maybeck, 1979). In practise both are rarely true for mobile robots as it is not possible to describe a curved trajectory linearly. Taylor expansion is instead used to linearize them during the prediction and correction steps, however the mathematics behind this process are beyond the scope of this report and will not be explained here. The use of an EKF for localisation in this project was chosen due to its simplicity compared with other more advanced Bayes filters, some of which are described briefly below.

A more detailed breakdown and explanation of EKF localisation, where the EKF algorithm is used to accurately estimate the position of a mobile robot, can be found in section 4.3.

Other Probabilistic Models

Unscented Kalman Filter

The second main version of the Kalman filter is the Unscented Kalman Filter (UKF). Like the EKF it linearizes the Gaussian probability distributions however it instead uses a method called the unscented transform. This transform uses so-called sigma points that are positioned on the mean, as well as symmetrically along the main axis of the covariance. These sigma points are then passed through the non-linear state transition function, a function that represents the action of the robot, and used to create a new linearized Gaussian. This use of sigma points gives the effect of being closer to the original function and allows it to work more effectively with highly non-linear position estimates. More information on this can be found in Julier and Uhlmann's paper (1997), which introduced the formula.

Nonparametric Filters

Rather than using linear Gaussians to represent the true belief like in the Kalman filter, monte-carlo methods can be deployed. This involves repeatedly sampling the true belief to produce a finite number of values that represent it. Two main methods for sampling are used, the first being the Histogram filter and the second being the Particle filter (West and Harrison, 1997; Metropolis and Ulam 1949). These techniques can produce extremely efficient results in non-linear systems, much more so than Kalman filters, as when the number of samples tends to infinity they match the true belief exactly. Particle filters are also particularly quick to implement, however they are computationally expensive as they require a relatively high number of samples to be made for each step, while the Kalman filter only needs to produce the mean and covariance.

Simultaneous Localisation And Mapping

A major problem that all the previous methods face when used for localisation is that they rely on premade maps for comparison in the measurement update step. Simultaneous Localisation and Mapping (SLAM) is the problem of having no map and instead estimating a map of the robot's environment while the localisation is simultaneously taking place. The methods shown previously have all been used to solve this problem through difficult adaptations where the map must be estimated along the way. While an interesting future problem to attempt, it is not within the scope of this report.

1.5 Approach

The approach taken for this project will be to first produce a working EKF in simulation to gain some understanding of its efficiency, speed and functioning. This will be implemented using Matlab, as the linear algebra required when linearizing and combining Gaussians can be easily coded and read using its matrix manipulation capabilities. This simulation will take the form of a

simulated body model of a differential drive robot with an EKF brain model doing passive localisation on top, predicting the position of the body model. Two simulated robots will therefore be produced from this, the first the simulated real world robot that moves in a noisy environment (the body), the second the EKFs localisation model of what it believes the simulated robot is doing (the brain).

Once the simulation is completed the EKF will be transferred to a purpose built robot to allow for analysis of its abilities in the real world. Landmark feature detection will be used as part of the measurement update step and two versions of landmarks and sensors will be attempted. The first will simply be using multiple light sensors on the top of the robot to calculate the bearing and range to a single light source. The second will use a 360-degree panoramic camera to detect multiple coloured landmarks placed around the map. These will be tracked using simple computer vision techniques and their bearing and range will then be passed back. The purpose built robot will be created through a combination of an Arduino microcontroller, to control the robot's motors and collect sensory data, which is connected through Serial to a Rapberry Pi computer, to allow for fast processing of the EKF algorithm. Due to the use of the Raspberry Pi, Python will be used as it can communicate with the Arduino very easily. The Arduino is programmed in C/ C++ however with added Arduino functions to control the microprocessor.

1.6 Overview

Section 2 of this report outlines the various professional considerations that were felt to be specifically relevant to this dissertation and how they were met. Section 3 details the requirements analysis of both the simulation and hardware implementation. Section 4 describes the implementation of the localisation behaviour in simulation, while section 5 shows and analyses the results of running different localisation brain models on a simulated body that wanders a noisy world randomly. Section 6 describes the implementation of the localisation behaviour on hardware. Section 7 shows the results of running different localisation techniques on the purpose built robot that moves back and forward across the world. Finally section 8 gives a general overview of the project and discusses next steps.

2 PROFESSIONAL CONSIDERATIONS

Any and all work in this project has been done with due consideration for the BCS Code of Conduct (BCS, 2011). This section outlines the various considerations that were felt to be specifically relevant to this dissertation and how they were met.

2.1 PUBLIC INTEREST

- *1b) – Have due regard for the legitimate rights of Third Parties.*
    - I shall reference any third party code or libraries I use where relevant. All other intellectual property from books or academic papers will also been referenced carefully to ensure due credit is given.

2.2 PROFESSIONAL COMPETENCE AND INTEGRITY

- *2c) - Develop your professional knowledge, skills and competence on a continuing basis, maintaining awareness of technological developments, procedures, and standards that are relevant to your field.*
    - This project will help me develop my skills and competence in the field and I will ensure that I maintain awareness of other technological developments that may aid the undertaking of my project.
- *2d) Ensure that you have the knowledge and understanding of Legislation and that you comply with such Legislation, in carrying out your professional responsibilities.*
    - I will ensure I comply with all Legislation that relates to my project.
- *2e) Respect and value alternative viewpoints and, seek, accept and offer honest criticisms of work.*
    - Criticisms of others work will all be of an honest and educated nature. I will also ensure that I critically analyse my findings and results and accept any other honest criticism that may be given by others.

2.3 DUTY TO RELEVANT AUTHORITY

- *3e) Not misrepresent or withhold information on the performance of products, systems or services (unless lawfully bound by a duty of confidentiality not to disclose such information), or take advantage of the lack of relevant knowledge or inexperience of others.*
    - All findings will be published, with performances of my implementations carefully shown.

3 REQUIREMENTS ANALYSIS

This section details the requirements of both the simulation and hardware implementation of localisation using an EKF. The system requirements are the important guidelines to be followed regarding what the system should do (functional requirements) and how the system should do it (non-functional requirements). User requirements explain important points regarding the ease of use of the system when it is being used and analysed.

3.1 SYSTEM REQUIREMENTS

While function requirements describe the behaviour of the system, non-functional requirements describe software used along with performance characteristics.

FUNCTIONAL

SIMULATION

- Simulate a differential-drive agent's body model whose pose – x and y coordinate and bearing – can be controlled using left and right wheel velocities
- Agent's brain model produces a belief distribution of the body model's pose based on its wheel rotational velocities
- Agent's brain model produces a probability density of the body model's pose based on sensory measurement – range and bearing – to multiple known features
- Agent's brain model combines belief distribution and probability density, performing localisation behaviour and giving an accurate posterior belief of its body model's pose

HARDWARE

- Differential-drive real robot built whose pose – x and y coordinate and bearing – can be controlled using left and right wheel velocities
- Wheel rotational velocities measurable and converted to translation and rotational velocities
- Internal model produces belief distribution of robot's pose based on measured wheel rotations
- Internal model produces probability density of robot's pose based on light based sensory measurement – range and bearing – to a single light source
- Sensory measurement to be extended to measure multiple known features through feature tracking of large coloured landmarks using panoramic 360-degree camera
- Internal model combining belief distribution and probability density, performing localisation behaviour and giving an accurate posterior belief of the robot's pose

NON-FUNCTIONAL

SIMULATION

- Extended Kalman Filter algorithm written by Thrun et al. (2006) will be basis of algorithm used to combine models
- Simulation to be programmed in Matlab 7.12.0 R2011a
- Run time of EKF of time length 1000 shall not exceed 100 milliseconds
- Simulation shall be portable across both Apple OSX 10.9 and Windows 8
- Brain model to predict location of body model with an error of less than 1

HARDWARE

- Extended Kalman Filter produced by simulation to be used in internal model
- Arduino Uno R3 used to control motors and make sensory measurements

- Raspberry Pi 2 Model B used to calculate internal model of robots pose
- Optical rotary encoders used to measure number of wheel rotations
- Four light dependent resistors (LDRs) attached to robot used to measure range and bearing to single known light source for sensory measurement of EKF
    - Extended to use panoramic 360-degree lens with a camera attached to the robot to detect coloured landmarks
- Arduino to be programmed using native environment (C/C++)
    - Compiled using avr-g++ compiler to allow for execution on Arduino's AVR microcontroller
- Raspberry Pi to run Raspbian and be programmed in Python 2.7
- Communication time between Raspberry Pi and Arduino should not exceed 100 milliseconds
- For safety reasons voltage running through robot will not exceed 5V
- Batteries used will be new and undamaged to avoid leaks
- Motors to brake when not in use

3.2 User Requirements

This project has no user base and is instead an attempt to understand and replicate current research in the field of Probabilistic Robotics. As such, there are very few user requirements for this project.

Simulation

- Noise parameters to be modifiable by user
- Code to be easily understandable through commenting, allowing for modification by user
- Code to be easily transferable to real robot
- Figures and plots produced by code to be easily readable and understood by user

Hardware

- Allow for wireless connection to Raspberry Pi using SSH, ensuring robot can be monitored remotely
- Estimated pose of robot to be clearly shown as robot moves
- Access to robot motor's brakes to be given to Raspberry Pi through inputting a 0 in the Serial output
- Access to other motion commands to be limited to Arduino board, i.e. motors cannot be controlled, via Raspberry Pi

4 SOFTWARE SIMULATION

The section describes the implementation of the localisation behaviour in simulation. It explains key concepts understood and goes on to detail the steps of the EKF algorithm used.

4.1 AGENT CONFIGURATION

A 2-dimensional circular agent will be used that is controlled through differential drive – where there is one wheel either side of the body, each with its own motor – allowing for simple control of the agent's movement. Differential drive was chosen as models of its motion were easy to simulate and could be a lot more accurate. The diameter D of the agent determines the distance between the wheels while the radius r determines the radius of the wheels.

4.2 BODY

The body model used gives a simple model of how a two wheeled differential drive robot would move in real life. This takes the place of a real robot, giving a simulated agent that the EKF can attempt to localise.

BODY POSE

The pose of the agent's body comprises of two location variables, x and y, along with a bearing variable θ. This pose is represented by the state variable x_t (not to be confused with the x location) – it's pose at some time t. The following vector more simply describes this pose:

$$x_t = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$$

The x and y location variables represent coordinates in relation to a map m. The θ bearing variable represents the number of radians, in a counter clockwise direction, from the x-axis. A bearing of θ = 0 therefore points the robot in the direction of the x-axis while a bearing of θ = 0.5π points it in the direction of the y-axis. This is illustrated more clearly in figure 8 below.

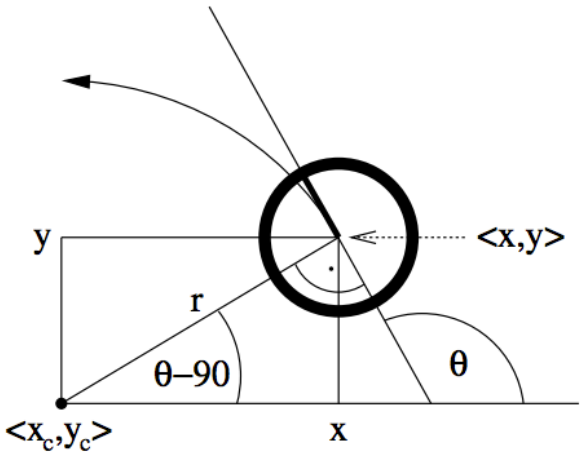


FIGURE 8 - Kinematic Representation of Robot
(Thrun, Burgard And Fox, 2007)

BODY MOTION MODEL

Controlling the velocity of the left wheel V_L and the velocity of the right wheel V_R allowed for simple and realistic control of the agent in a way that mirrored the control of the real robot

produced later. This similarity later allowed for quick transferring of generated wheel controls to the real robot, one of the requirements of the project.

After a small time-step dt the agent's pose would be updated to reflect the affect the wheel velocities had on the position and bearing of the agent discussed above. The equations below show this change after each time step:

$$x_t = x_{t-1} + dt\left(\frac{V_R + V_L}{2}r + \gamma\right)\cos(\theta_{t-1})$$

$$y_t = y_{t-1} + dt\left(\frac{V_R + V_L}{2}r + \gamma\right)\sin(\theta_{t-1})$$

$$\theta_t = \theta_{t-1} + dt\left(\frac{V_R - V_L}{D}r + \gamma\right)$$

As can be seen a small amount of zero-mean Gaussian noise γ was added to give a rough simulation of the noise that can occur in the real world.

4.3 Brain

The brain model used gives an estimate of where the body model moved to after some motion command, producing a posterior belief distribution of this new pose. It does this by using an EKF to combine a belief distribution of its new pose, produced by its motion model using the motion command, with a probability distribution of its new pose, which was produced by a measurement model using sensory measurements to features in the map.

Brain Pose

The pose of the brain model x_t at some time t is estimated using a Gaussian distribution of mean μ with a covariance Σ:

$$x_t = \begin{pmatrix}\mu \\ \Sigma\end{pmatrix}$$

$$\mu = \begin{pmatrix}\mu_x \\ \mu_y \\ \mu_\theta\end{pmatrix}$$

$$\Sigma = \begin{pmatrix}\sigma_x & \sigma_{x,y} & \sigma_{x,\theta} \\ \sigma_{x,y} & \sigma_y & \sigma_{y,\theta} \\ \sigma_{x,\theta} & \sigma_{y,\theta} & \sigma_\theta\end{pmatrix}$$

A Gaussian distribution was used to represent the pose as it allows for some degree of belief in its position, a fundamental principle of probabilistic robotics explained earlier.

Brain Motion Model

The motion model produces a belief distribution of the agent's pose after some motion command sent to the body's wheels. Two options for controlling the brain model's motion in the EKF are through either calculating the translational v_t and rotational w_t velocity commands sent to the body, or though odometry measurements, found by integrating the number of wheel rotations after the agent has moved. While both models suffer from slippage in their wheels, the velocity motion command also has a mismatch, when used on real robots, between the velocity command sent to the motor and the actual velocity produced. As the simple body model above does not define this motor error, by instead simply having an overall noise value attached to its new position, the simpler velocity motion commands were used for the simulation. This gave the motion command u_t at some time t:

$$u_t = \begin{pmatrix} v_t \\ w_t \end{pmatrix}$$

Conversion was needed however from the body's left wheel velocity V_L and right wheel velocity V_R to give the translational and rotational velocity:

$$v_t = \frac{(V_R + V_L)}{2} r$$

$$w_t = \frac{V_R - V_L}{D} r$$

The diameter of the agent D and the radius of the wheels r were also used to allow for easy transfer to the real robot later. Conversions given by LaValle (2006) were adapted to allow for D and r to be incorporated.

The velocity motion command led to the velocity motion model being used, where after some small time step dt the mean of the agent's pose would be updated to reflect the effect the translational and rotational velocities had. Taken from Thrun et al. (2006), the equations below show this change after each time step:

$$\mu_t^x = \mu_{t-1}^x + (-\frac{v_t}{w_t}\sin(\theta) + \frac{v_t}{w_t}\sin(\theta + w_t dt))$$

$$\mu_t^y = \mu_{t-1}^y + (\frac{v_t}{w_t}\cos(\theta) - \frac{v_t}{w_t}\cos(\theta + w_t dt))$$

$$\mu_t^\theta = \mu_{t-1}^\theta + w_t dt$$

As the velocity motion model is used to generate the posterior distribution of the agent's location in a noisy environment, the covariance of the agent's pose must also be updated. The mathematical formula behind this goes beyond the scope of this report, however it is important to note that it is first required to model some real world noise on the motion command:

$$\begin{pmatrix} \hat{v} \\ \hat{w} \end{pmatrix} = \begin{pmatrix} v \\ w \end{pmatrix} + \begin{pmatrix} \varepsilon_{a_1} v^2 + \varepsilon_{a_2} w^2 \\ \varepsilon_{a_3} v^2 + \varepsilon_{a_4} w^2 \end{pmatrix}$$

Where $\hat{v}$ = a noisy translational velocity, $\hat{w}$ = a noisy rotational velocity, ε_a = some zero-centred Gaussian noise with variance a_i and a_i = the noise parameters to be calibrated to fit the body model. By altering the variance, this noise is used in the EKF to determine the spread of possible poses at which the agent may arrive.

Figures 9 and 10 show how this motion model is used to describe the posterior distribution of a robot's location after executing some motion command on a starting pose in a noisy world. Note that taking the maximum probability of possible orientations has allowed for the orientation of the robot to be ignored. In both figures, the robot starts at coordinates (0,0) and is facing along the y-axis.

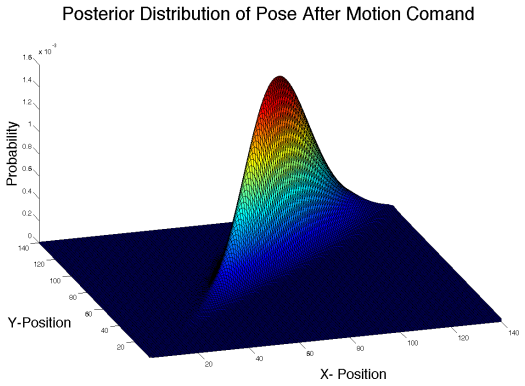


FIGURE 9 - 3D PLOT OF POSTERIOR PROBABILITY DISTRIBUTION OF END POSE XT AFTER MOTION COMMAND UT ON STARTING POSE

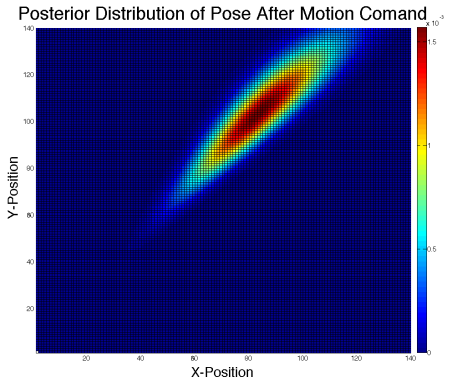


FIGURE 10 - HEAT MAP OF POSTERIOR PROBABILITY DISTRIBUTION OF END POSE XT AFTER MOTION COMMAND UT ON STARTING POSE

BRAIN MEASUREMENT MODEL

A feature-based measurement model is used to allow for the agent to update its brain model with a probability distribution of its body's location in the EKF. These features correspond to landmarks around the environment that have known positions relative to the agent's starting pose. All landmarks are listed in some feature based map m where each has a property p, for example a colour or other defining characteristic, along with an x and y position:

$$m = \begin{pmatrix} p_1 & p_2 \\ x_1, & x_2, \dots \\ y_1 & y_2 \end{pmatrix}$$

By measuring the range r_t and bearing ϕ_t to each landmark, the agent can work out its position relative to this global coordinate frame. When there are multiple landmarks a signature s_t is also produced to decipher which landmark is which by comparing it to the property of each landmark. These three parameters define the list of sensory measurements z_t:

$$z_t = \begin{pmatrix} r_t^1 & r_t^2 \\ \phi_t^1, & \phi_t^2, \dots \\ s_t^1 & s_t^2 \end{pmatrix}$$

Detection of landmarks may not always be possible however, due to landmarks or sensors being obscured or from other sensor errors, and feature-based measurement models can be highly prone to large fluctuations from this (Cox, 1991, pp.196-197). For the purposes of the simulation however this was not modelled and it was assumed that all landmarks could be detected at all times.

In the simulation imaginary sensors were used that could produce estimates of the correct range, bearing and signature with some level of zero-mean Gaussian noise γ added to each. These are calculated using standard geometric laws provided by Thrun et al. (2006):

$$\begin{pmatrix} r_t^i \\ \phi_t^i \\ s_t^i \end{pmatrix} = \begin{pmatrix} \sqrt{(m_{j,x} - x)^2 + (m_{j,y} - y)^2} \\ atan2(m_{j,y} - y, m_{j,x} - x) - \theta \\ s_j \end{pmatrix} + \begin{pmatrix} \gamma_r \\ \gamma_\phi \\ \gamma_s \end{pmatrix}$$

The probability density $p(z_t \mid x_t, m, c_t)$ can then be shown as the probability of having some measurement based on each possible pose, some map and correspondences c between features in the map and in the measurement. This probability density can be visualised as a unimodal

Gaussian, which is to be later folded with the belief distribution given by the prediction step. An estimate of the sensor's noise is also calibrated in this step to predict how accurate your sensors are and in essence how broad your Gaussian is.

The number of landmarks, I predict, will have a large influence on the later accuracy of the EKF. This is because sensory measurement to one landmark will simply make a ring around that landmark at the measured range, as shown in figure 11. When two landmarks are used the two circles produce two points where they overlap that the agent could be in, see figure 12. Three landmarks will further narrow this down to one position, see figure 13.

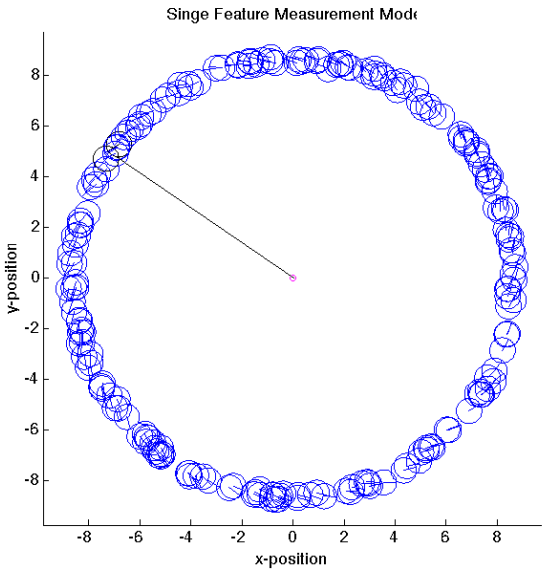


FIGURE 11 – SINGLE FEATURE MEASUREMENT MODEL

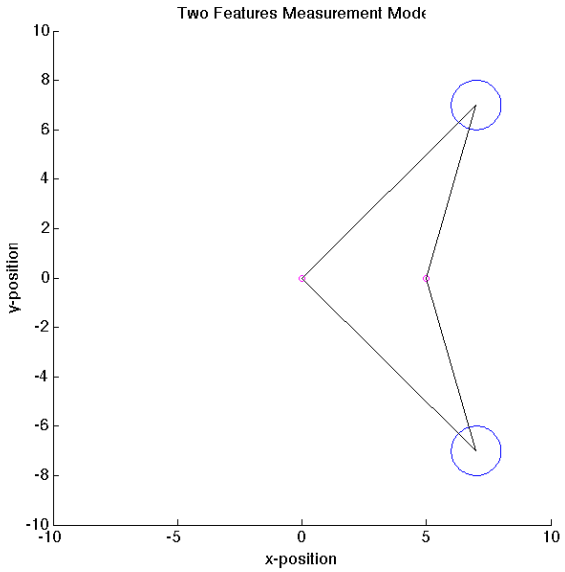


FIGURE 12 – TWO FEATURE MEASUREMENT MODEL

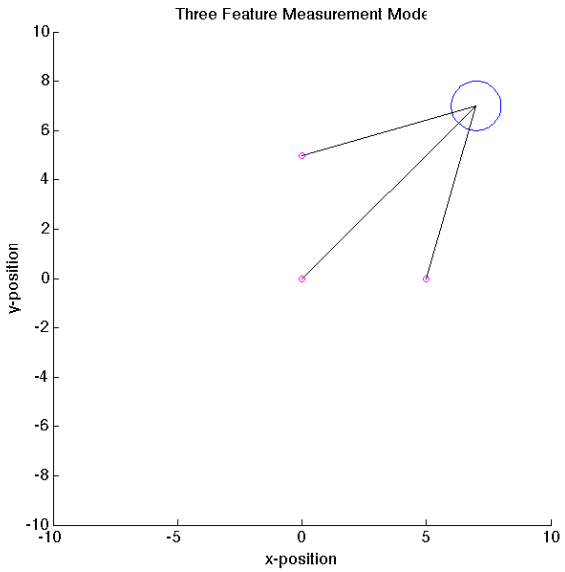


FIGURE 13 – THREE FEATURE MEASUREMENT MODEL

Due to uncertainty however each of these must instead be modelled as a probability density rather than distinct points or lines.

EXTENDED KALMAN FILTER ALGORITHM

The EKF algorithm combines the unimodal Gaussian distributions given by the velocity motion model and the feature based measurement model. The algorithm has been taken and slightly adapted from one proposed by Thrun et al. (2006).

1 – INITIAL POSE ESTIMATE

An initial estimate of the pose of the agent was needed before the EKF could begin. For the purposes of the simulation it was assumed that this prior belief was known with quite a high degree of accuracy and the initial pose of the brain was set to reflect that. Its mean values were each set to equal the body's initial pose of (0, 0, 0) and it was given only a small amount of variance along the diagonal and no covariance, indicating each axis only had some small amount of variance with itself:

$$\mu = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\Sigma = \begin{pmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{pmatrix}$$

This has been visualised for an agent in a 1-dimensional environment in figure 14.

FIGURE 14 – INITIAL POSE ESTIMATE OF A 1-DIMENSIONAL ROBOT

2 – PREDICTION STEP

The prediction step uses the brain's velocity motion model and updates the pose of the agent after some small motion command u_t is sent. This gives a new belief distribution of the agent's position, however the noise modelled causes the new belief distribution to be less certain, being viewed as broader and lower. In figure 15 below this has again been visualised for an agent in a 1-dimensional environment. It is important to note that the prediction step must also linearize the covariance matrix of the agent's pose, ensuring it can then be combined with the linearized measurement model in the correction step.

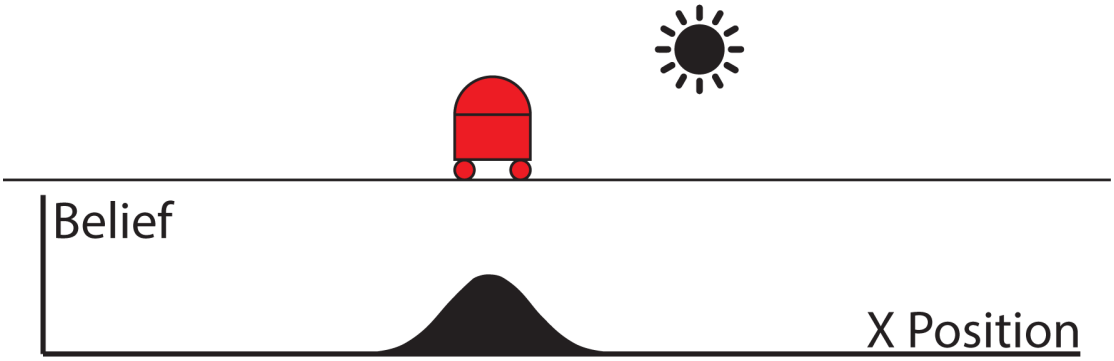


FIGURE 15 – PREDICTION STEP OF A 1-DIMENSIONAL ROBOT

3 – CORRECTION STEP

The correction step optimally combines the belief distribution, given by the just completed prediction step, with a linearized probability density of the agent's pose, given by the feature based measurement model. This folding of the prediction with the measurement model's generated probability distribution produces a more accurate and confident posterior belief. Similar to the prediction step, the correction step must linearize the covariance matrix of the sensory measurement, as the combined models must both be linear for the Kalman filter to work. This is repeated for each feature that has been measured. Figure 16 shows first a probability density produced by the measurement model and then the result of it being folded with the belief distribution from 15.

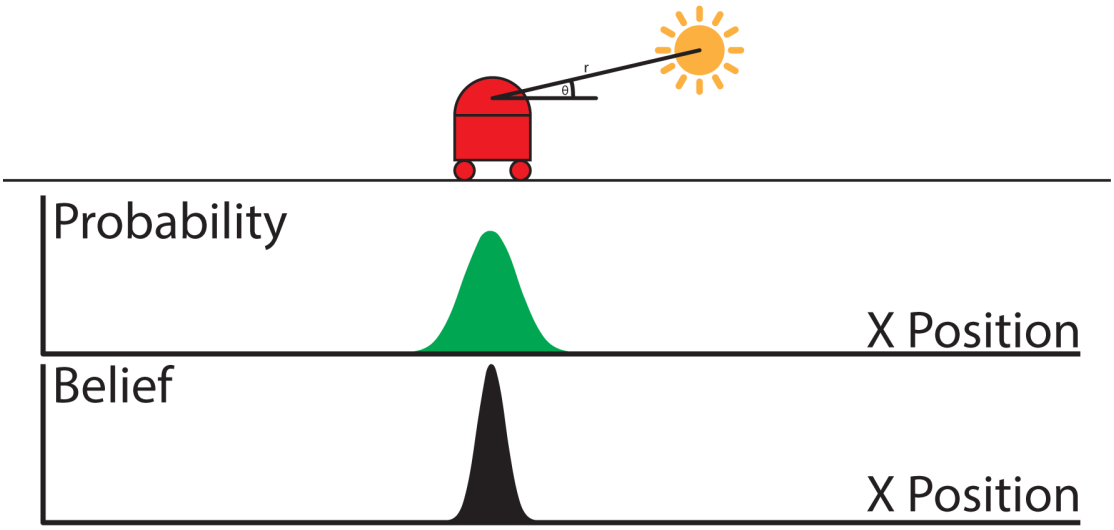


FIGURE 16 – MEASUREMENT UPDATE STEP OF A 1-DIMENSIONAL ROBOT

After the completion of the correction step the algorithm loops back to the prediction step and continues until the agent is stopped.

4.4 Problems encountered

Something not covered in Thrun et al.'s (2006) supplied algorithm was how to compute the difference of two angles during the correction step, where different bearings were taken away from each other. Bearings were within the limit of π radians and -π radians, where a bearing of π was equal to a bearing of -π. When calculating the difference between π and -π however the answer is ±2π, not the correct 0. This caused the agent to behave erratically. To solve this however a fix was calculated using modulus to keep the answer to within π and -π:

$$mod(\theta_1 - \theta_2 + \pi, 2\pi) - \pi$$

Other sections of the algorithm also at times pushed the bearing above or below the π or -π limit, so this fix was added to any section where the bearing was altered.

Another problem faced was that when there was no rotational velocity, errors were thrown. The velocity motion model used calculates the new positions based on turning circles with radius r:

$$r = \left|\frac{v}{w}\right|$$

However when the rotational velocity is 0 the radius becomes infinity and the algorithm breaks down. No fix was found for this, however when testing it was easy to ensure the wheel velocity motion commands were never the same.

4.5 Calibration

Calibration of the brain's motion model involved running the body model 10 times for 1 time step, producing a distribution of positions. A small amount of time was then spent altering the alpha values of the motion model until its belief distribution fit the random sampling produced, however the initial error values of 0.1 for every a_i value seemed to be best.

Calibrating the measurement model simply involved using the Gaussian noise variance values added to the simulated measurement reading. This would not be so easy when calibrating for a real robot, as the values are unknown.

5 Simulation Results

This section shows the results of running different localisation brain models on a simulated body that wanders a noisy world randomly. To produce a random set of motion commands, simulating random wandering, the value of two nodes from a 100 node Continuous Time Recurrent Neural Network (CTRNN) were used, producing a smooth random movement around the world. See provided code for details of its implementation.

Two error values were calculated for each type of brain model, showing how it faired at localisation. First the Euclidian distance between the body and brain's final positions were calculated, then the difference between the body and brain's bearings were calculated. The average of 100 different runs for each model over 1000 time steps was used for these errors.

5.1 Motion Model

Before analysing the EKF's performance in simulation it was first vital to see how well the velocity motion model did by itself, without any features to measure. This would allow for a comparison to see if the added measurement update of the EKF had any real positive affect on the brain's accuracy. Figures 17 to 20 below show snapshots, at increasing time intervals of 250, of the agent body's position along with the motion model prediction of where the body is. The broken red line highlights the position error between them.

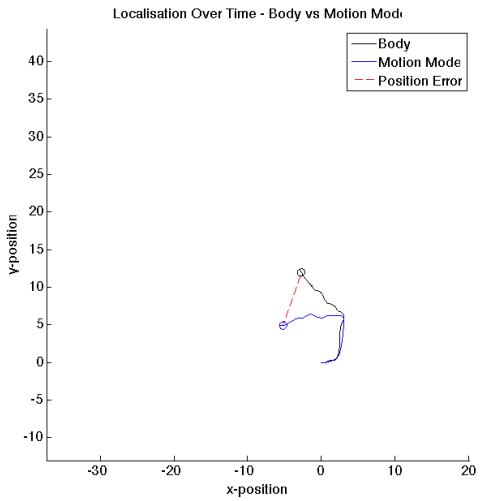


FIGURE 17 – LOCALISATION AT TIME = 250

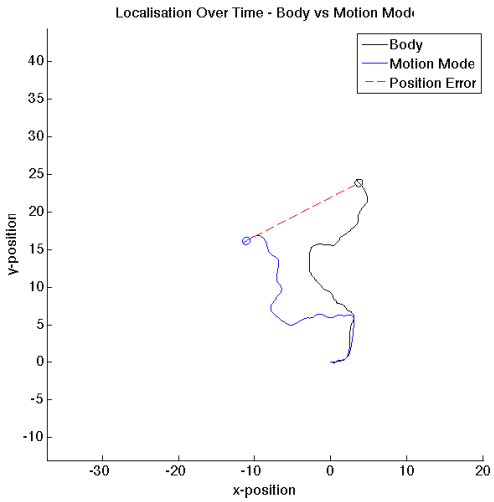


FIGURE 18 - LOCALISATION AT TIME = 500

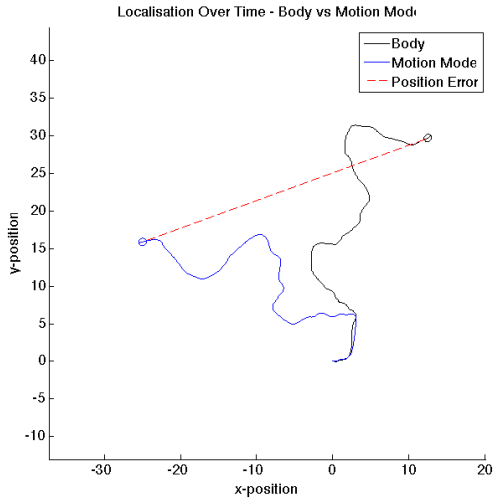


FIGURE 19 – LOCALISATION AT TIME = 750

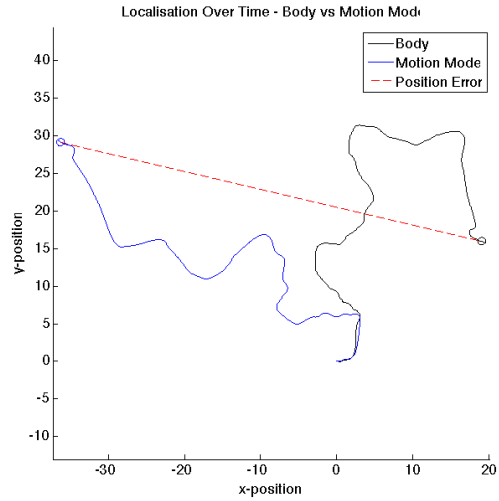


FIGURE 20 - LOCALISATION AT TIME = 1000

It can be quickly seen from the snapshots from figures 17 to 20 above that the measurement model suffers very heavily from the small amount of noise in the world. The motion model ends up predicting the body to be facing in the opposite direction and very far away from where it really is.

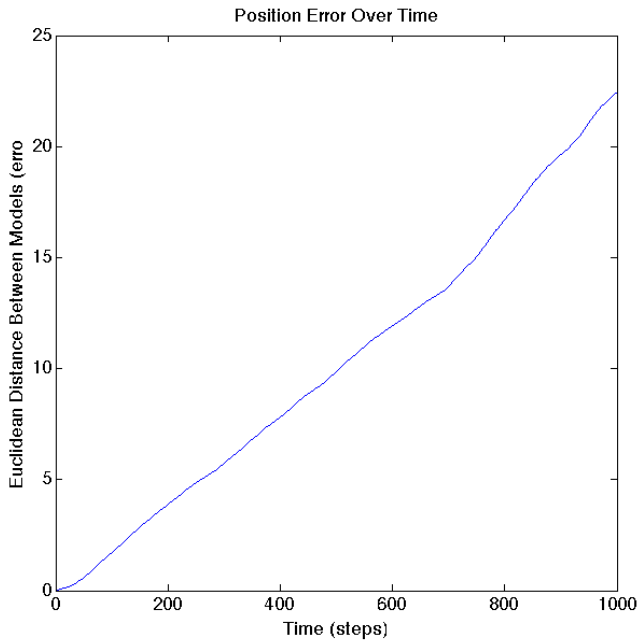


FIGURE 21 – MOTION MODEL POSITION ERROR OVER TIME

The average position error over time in figure 21 above shows clearly the problems with a simple motion model. As time increases so does the error, and with no clear end in site.

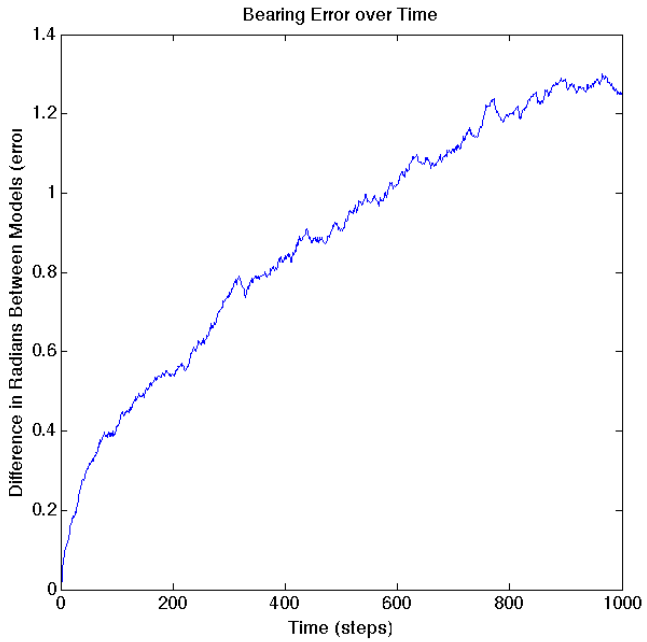


FIGURE 22 – MOTION MODEL BEARING ERROR OVER TIME

As time increases the body model's bearing also seems to diverge from the brain model predicting it, shown in figure 22 above. This large average bearing error over time may give some indication of why there was such a large position error, as even a small bearing error causes the agent to move in the wrong direction.

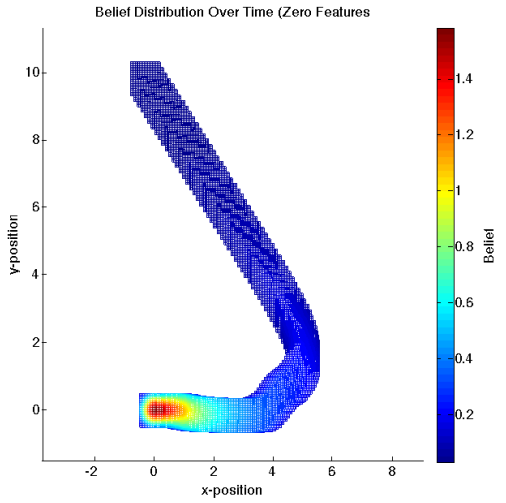


FIGURE 23 – MOTION MODEL BELIEF DISTRIBUTION OVER TIME HEATMAP

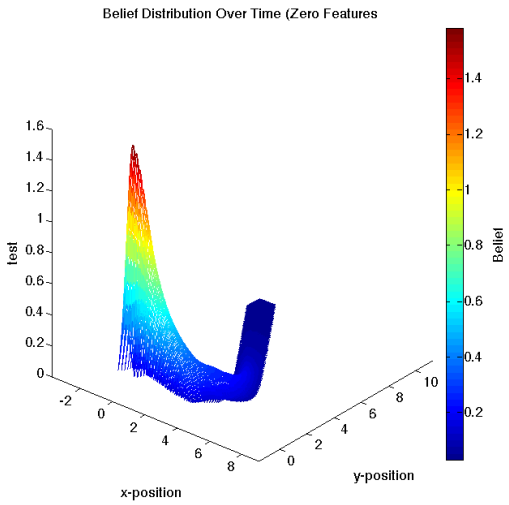


FIGURE 24 - MOTION MODEL BELIEF DISTRIBUTION OVER TIME 3D PLOT

Figures 23 and 24 above show the belief distribution of the brain's pose for a shorter 200 time steps. As the agent moves away, the brain's belief distribution shrinks very rapidly as the noisy motion model loses track of its position.

5.2 EKF

The EKF model shows the motion model combined with a measurement model and the affect this has on localisation. Different numbers of features were compared to see the affect it had on the accuracy of localisation.

One Feature

Figures 25 to 28 below show snapshots, at increasing time intervals of 250, of the agent body's position along with the mean of the EKF's prediction of where the body is. The broken red line highlights the position error between them and the magenta circle indicates the single feature that the EKF's measurement model will be using.

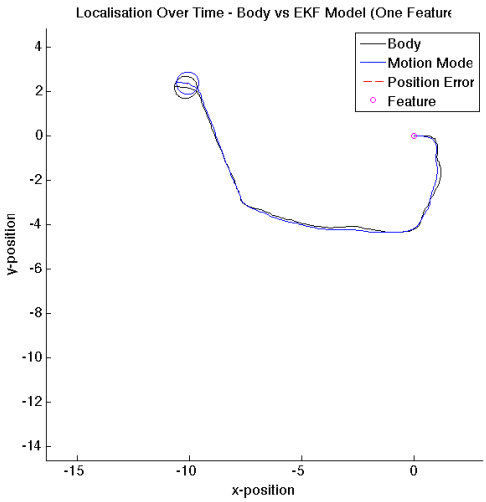


FIGURE 25 - LOCALISATION AT TIME = 250

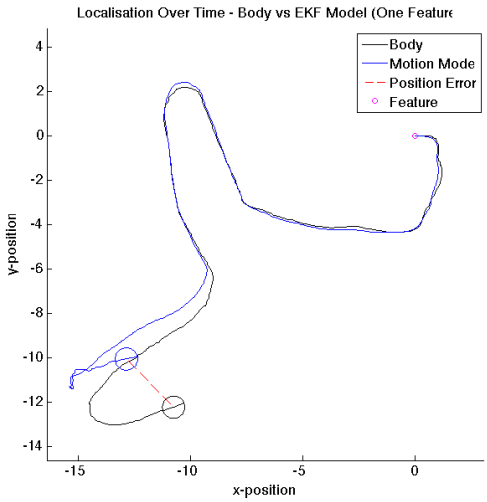


FIGURE 26 - LOCALISATION AT TIME = 500

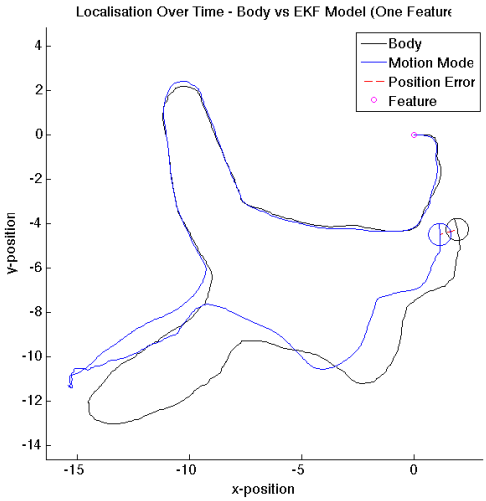


FIGURE 27 - LOCALISATION AT TIME = 750

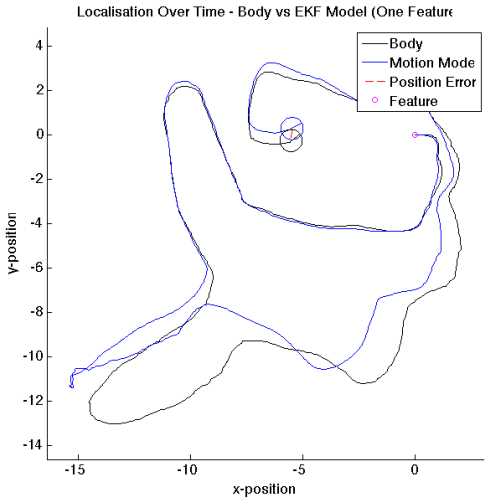


FIGURE 28 - LOCALISATION AT TIME = 1000

First impressions of the one feature EKF algorithm are mostly positive due to how close the brain stays to the body. Closer examination of figures 25 to 28 however shows that as the agent moves further away from the feature, the distance between the brain and body get larger. This is most likely an affect produced by the increased size of the circular probability density created by the measurement model. See section 4.3 above.

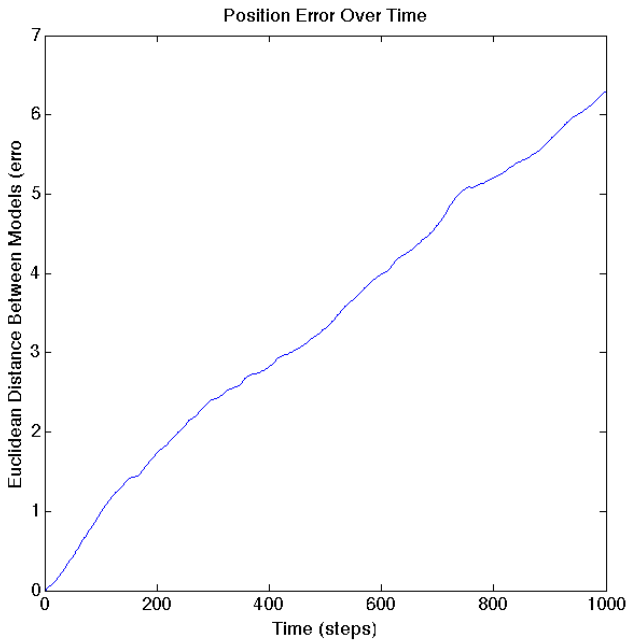


FIGURE 29 – ONE FEATURE EKF POSITION ERROR OVER TIME

Figure 29 shows how over time the localisation produced by an EKF with only one feature gets worse. As was discussed above, this is most likely due to larger distances between the agent and the feature giving a larger area of possible positions in the measurement model.

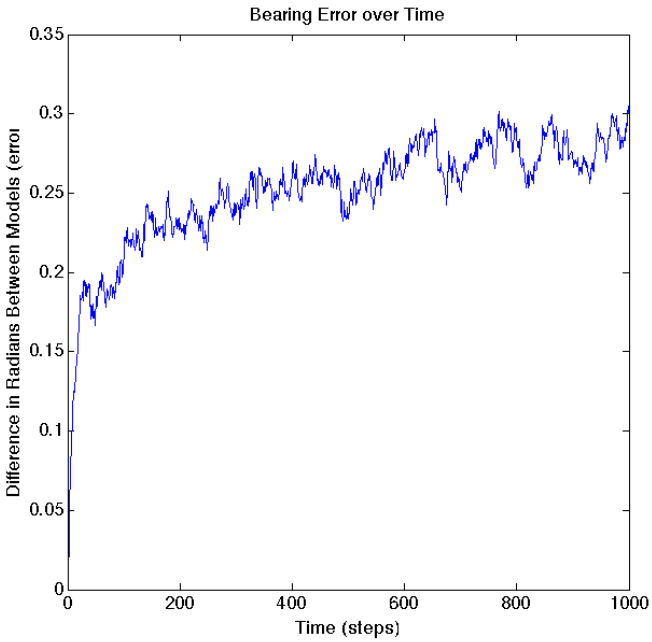
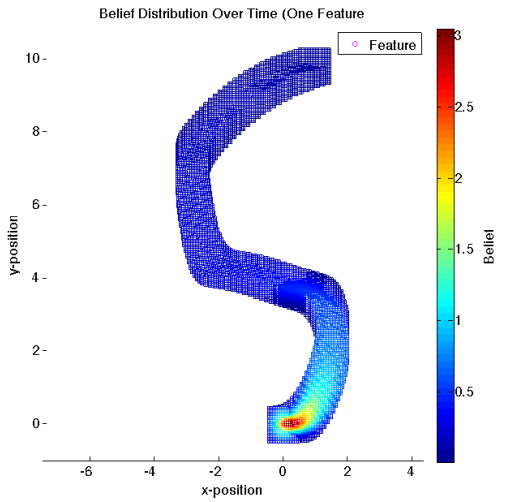


FIGURE 30 – ONE FEATURE EKF BEARING ERROR OVER TIME

While the distance error continues to grow, looking at the average bearing error over time in figure 30 above shows that it seems to stop at around 0.3, indicating only a slight deviation from the true bearing. This will have aided in the lower distance between the brain and body models.



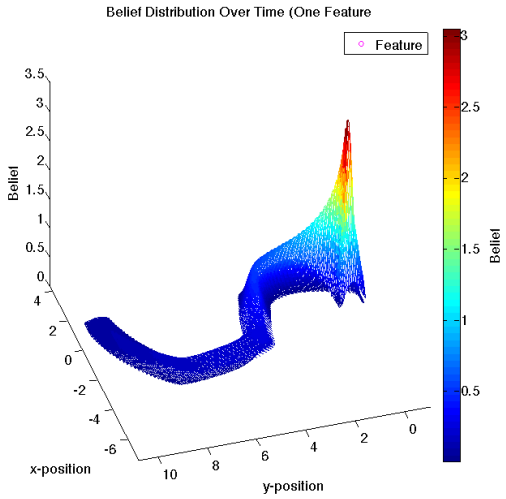


FIGURE 31 – ONE FEATURE EKF BELIEF DISTRIBUTION OVER TIME HEATMAP

FIGURE 32 - ONE FEATURE EKF BELIEF DISTRIBUTION OVER TIME 3D PLOT

Figures 31 and 32 above show the belief distribution of the brain's pose for another short 200 step run with large amounts of sensory noise. This highlights the problem that as the agent moves away from the landmark positioned at (0, 0) it gets more uncertain about its position. Although it does not drop as fast as without a measurement model, it does not do much better.

TWO FEATURES

Figures 33 to 36 below show snapshots, at increasing time intervals of 250, of the agent body's position along with the EKF prediction of where the body is. The broken red line highlights the position error between them and the magenta circles indicate the features that the EKF's measurement model will be using.

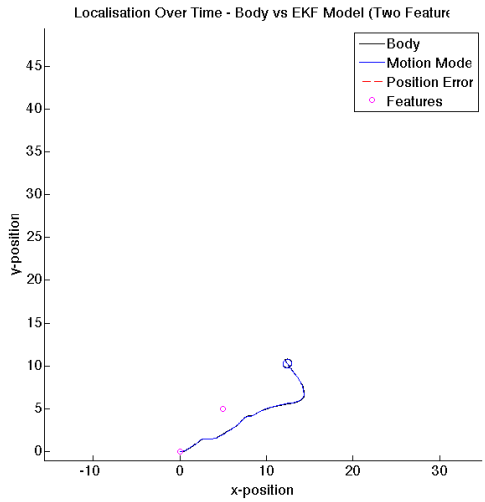


FIGURE 33 – LOCALISATION AT TIME = 250

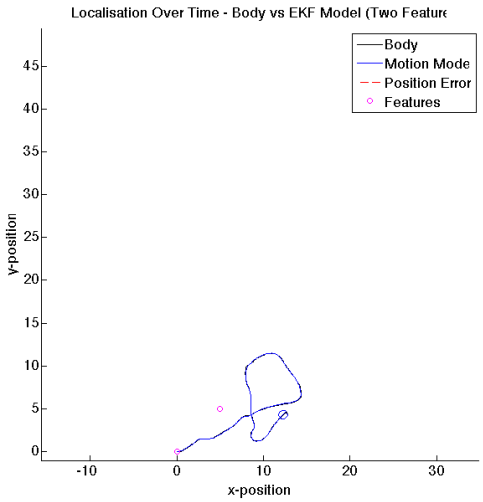


FIGURE 34 – LOCALISATION AT TIME = 500

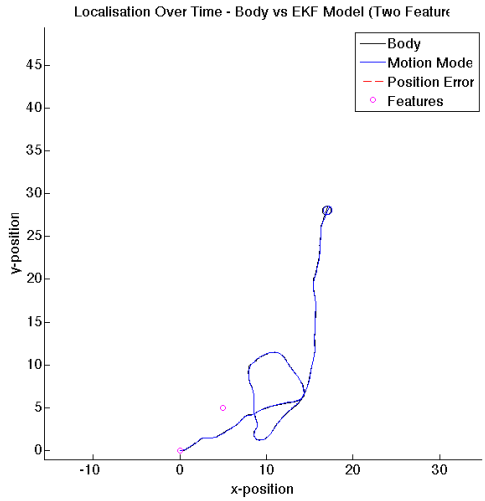


FIGURE 35 – LOCALISATION AT TIME = 750

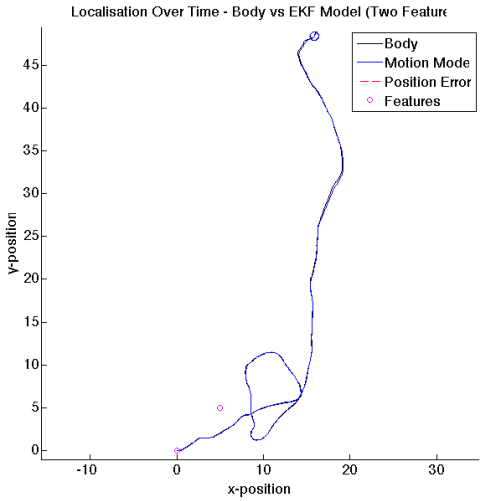


FIGURE 36 – LOCALISATION AT TIME = 1000

The results shown in figures 33 to 36 above are a huge improvement. The brain model produced by the EKF is shown to be almost identical to the body model, even when moving far away from the features. An understanding of the measurement model when given two features, as discussed in section 4.3, suggests that with two features the measurement model only shows two areas the agent could be in.

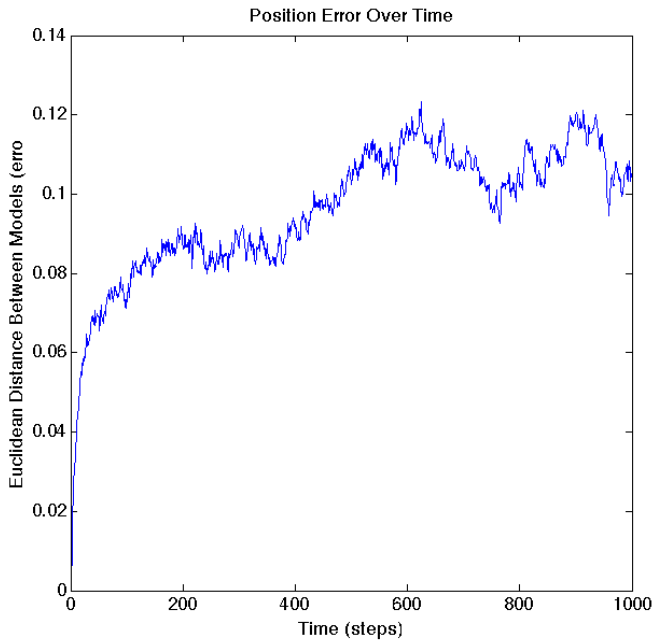


FIGURE 37 – TWO FEATURES EKF POSITION ERROR OVER TIME

Figure 37 shows how the average position error over time reaches a maximum value and doesn't seem to go higher, producing a very accurate belief in position. Interestingly the position error line is also more jagged than before, this may be due to the measurement model pushing it back.

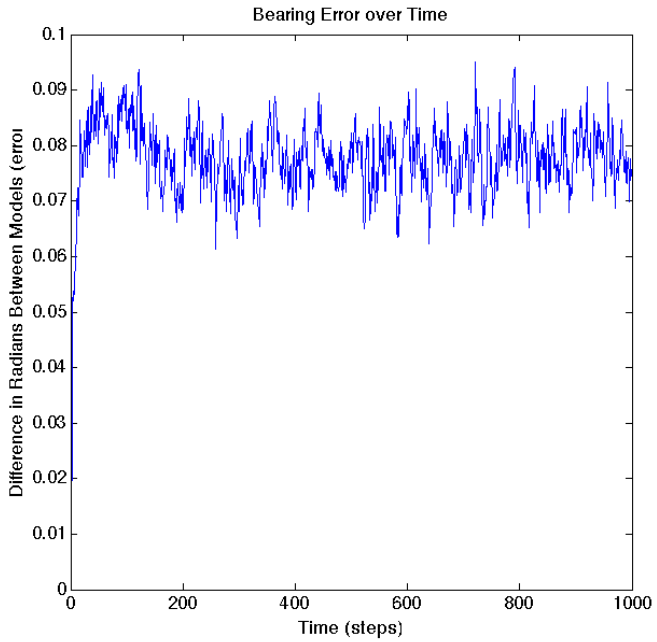


FIGURE 38 – TWO FEATURES EKF BEARING ERROR OVER TIME

Figure 38 above shows that the average bearing error over time is very small and gives only a small amount of deviation, leading to the low position error.

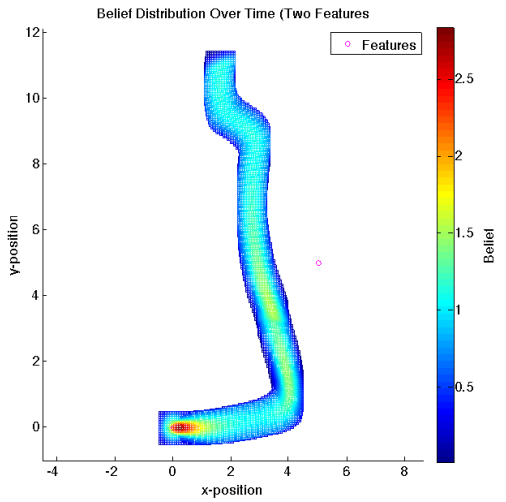


FIGURE 39 – ONE FEATURE EKF BELIEF DISTRIBUTION OVER TIME HEATMAP

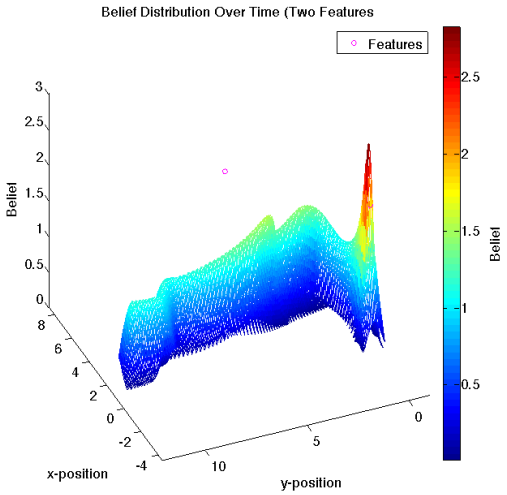


FIGURE 40 – ONE FEATURE EKF BELIEF DISTRIBUTION OVER TIME 3D PLOT

Figures 39 and 40 above show the belief distribution of the brain's pose for another short 200 step run with large amounts of sensory noise. With the addition of a second landmark at position (5,5), the agent keeps a higher and narrower belief distribution, with its belief increasing as it moves towards a new landmark. This belief stays high even as it moves away from both landmarks.

THREE FEATURES

Figures 41 to 44 below show snapshots, at increasing time intervals of 250, of the agent body's position along with the EKF prediction of where the body is. The broken red line highlights the position error between them and the magenta circle indicates the feature that the EKF's measurement model will be using.

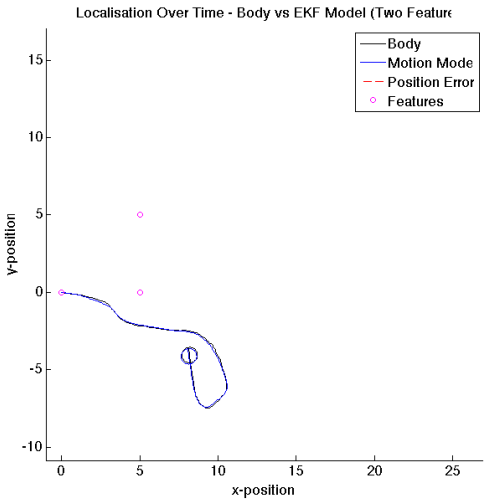


FIGURE 41 – LOCALISATION AT TIME = 250

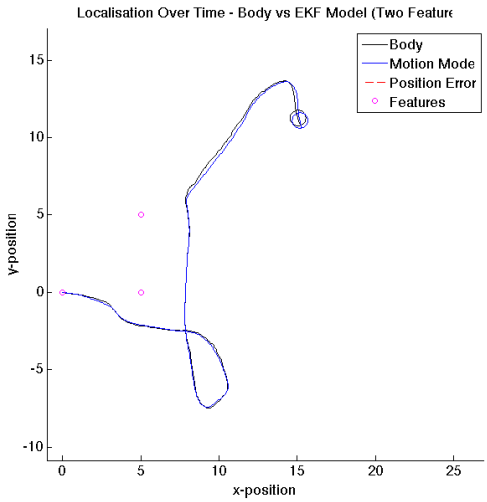


FIGURE 42 – LOCALISATION AT TIME = 500

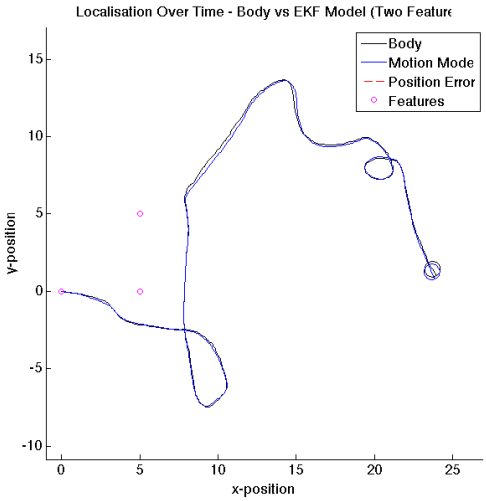


FIGURE 43 – LOCALISATION AT TIME = 750

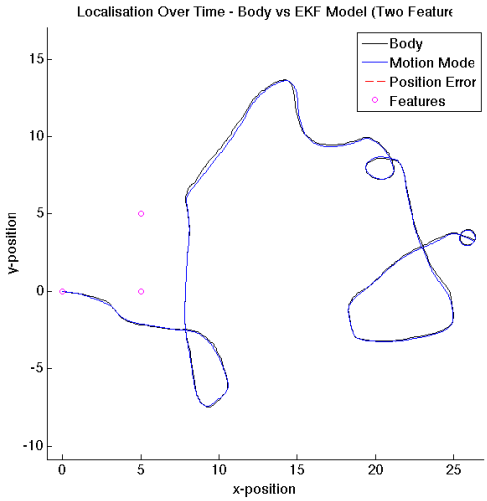


FIGURE 44 – LOCALISATION AT TIME = 1000

Much like the two feature EKF, figures 41 to 44 above show almost no deviation from the real position, producing very accurate localisation behaviour. Understanding the measurement model, section 4.3, shows that three landmarks allow for exact triangulation or trilateration, giving only one area in which the body could be located.

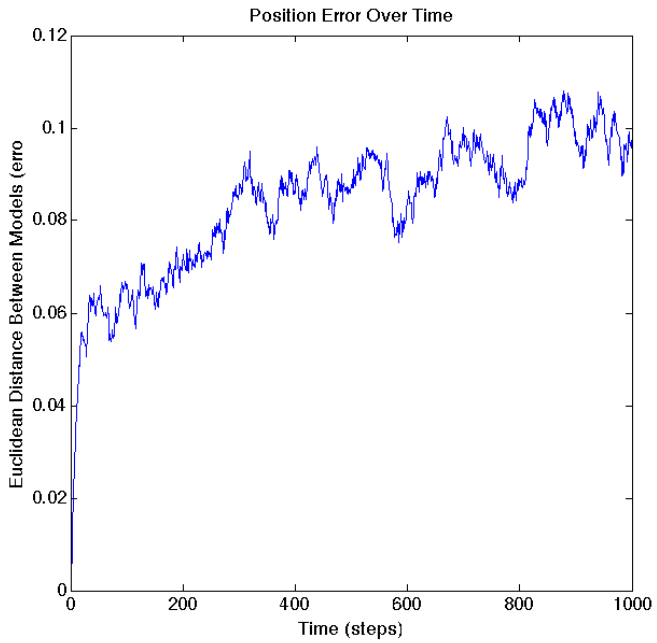


FIGURE 45 – THREE FEATURES EKF POSITION ERROR OVER TIME

Again, figure 45 shows this accurate localisation through a very small position error that maxes out at about 0.1. It is interesting to note again that the line is very jagged.

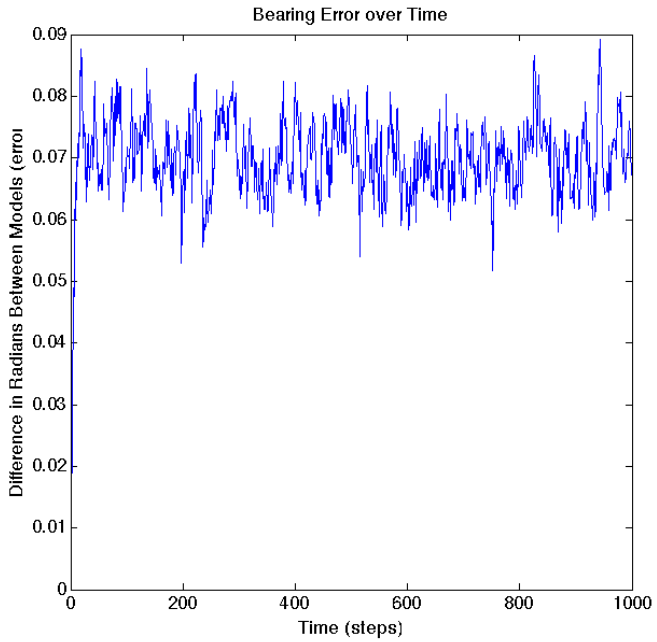


FIGURE 46 – THREE FEATURES EKF POSITION ERROR OVER TIME

Finally, the average bearing error over time in figure 46 is very low, thereby causing the agent to be very accurate.

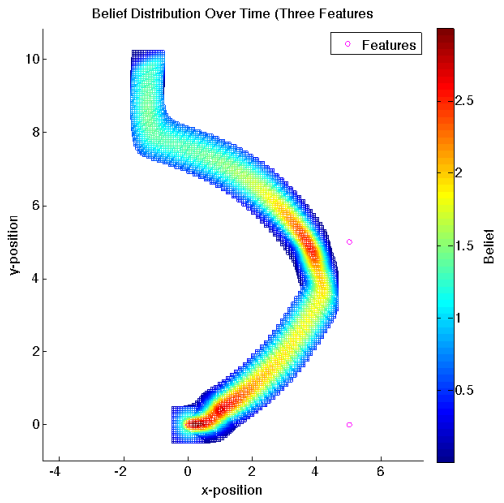


FIGURE 47 – ONE FEATURE EKF BELIEF DISTRIBUTION OVER TIME HEATMAP

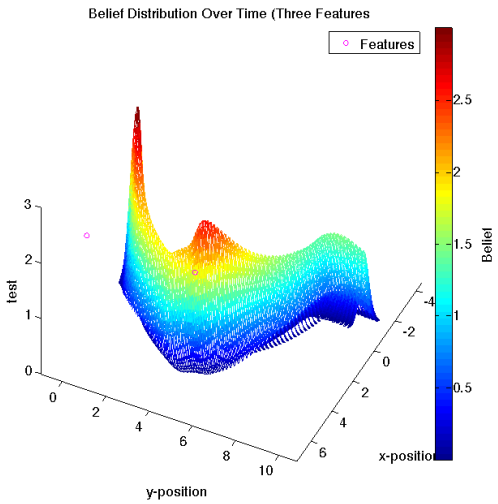


FIGURE 48 – ONE FEATURE EKF BELIEF DISTRIBUTION OVER TIME 3D PLOT

Figures 47 and 48 above show the belief distribution of the brain's pose for another short 200 step run with large amounts of sensory noise. A third feature, located at position (5,0) increases and narrows the belief even more, indicating less uncertainty about the body's position. Even as the agent moves far away from the landmarks its belief stays relatively high.

5.3 Comparison

Below are comparisons of the different position and bearing errors that have been shown. Their errors have been shown in log scale to better highlight their differences.

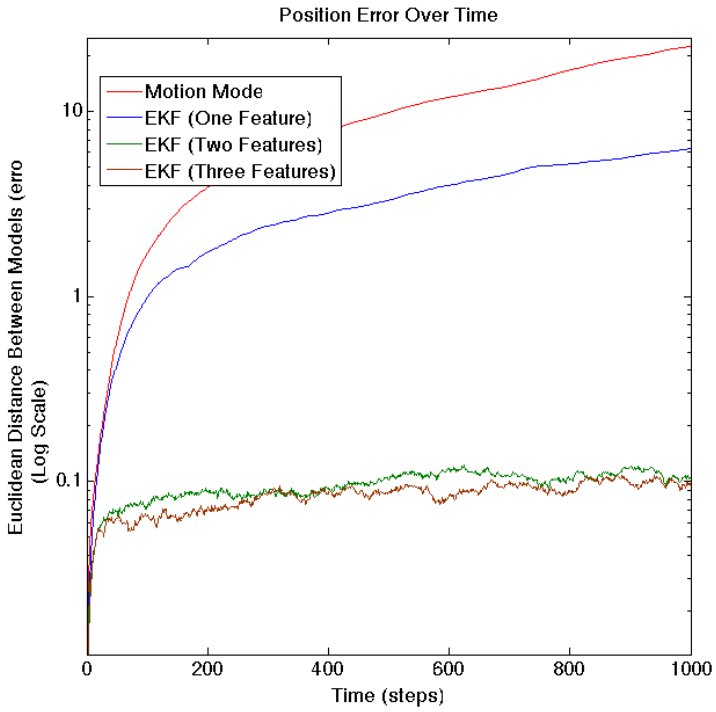


FIGURE 49 – COMPARISON OF POSITION ERROR

Figure 49 above clearly shows a large drop in position error when the EKF is used. The use of more than one feature produces another large drop. The difference between two and three features is quite small yet it is clear that increasing the number of features is better. Two and three features also plateau quite quickly, indicating that the error will not increase beyond this. The same cannot be said for the motion model by itself nor can it for the single feature EKF.

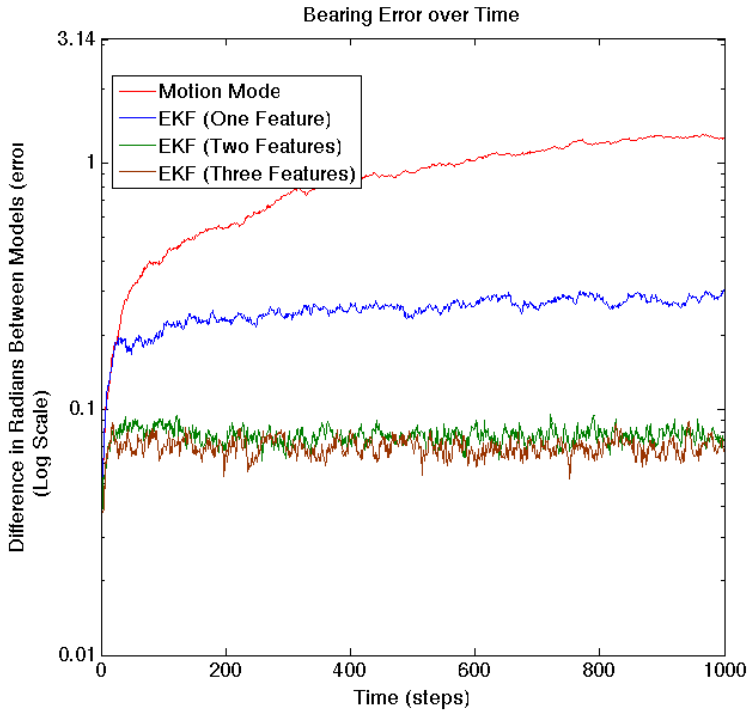


FIGURE 50 – COMPARISON OF BEARING ERROR

Figure 50 above similarly shows a large drop in bearing error when the EKF is used. The use of more than one feature again produces another large drop while the difference between two and three features is quite small. It is clear however that an increased number of features are better. One feature however is enough to limit the bearing error, shown by it plateauing, but the motion model seems to keep increasing.

5.4 Evaluation

Analysis of the simulation results shows the theory behind recursive state estimation holds up and that linearizing the belief distribution and probability density does not cause too many problems. The use of an EKF correctly localised the agent's body, shown by a significant drop in both distance and bearing error. While using only one feature did not give a low enough error to meet my requirements, it is still leagues ahead of the motion model by itself. Two and three features gave very low errors of below 0.1, much better than had been required.

Limitations

Even with such strong results in simulation however, the modelled noise is unrealistic and cannot compare to the complex noise that affects a robot in the real world. Linear noise was given to the simulation, but as discussed earlier, noise in the real world is very rarely linear. The EKF is expected to have problems when facing this as it can only be calibrated for linear noise.

6 TRANSFERRING TO HARDWARE

This section describes the implementation of the localisation behaviour on hardware. It explains the configuration of the robot used along with the transferring of the EKF algorithm.

6.1 ROBOT CONFIGURATION

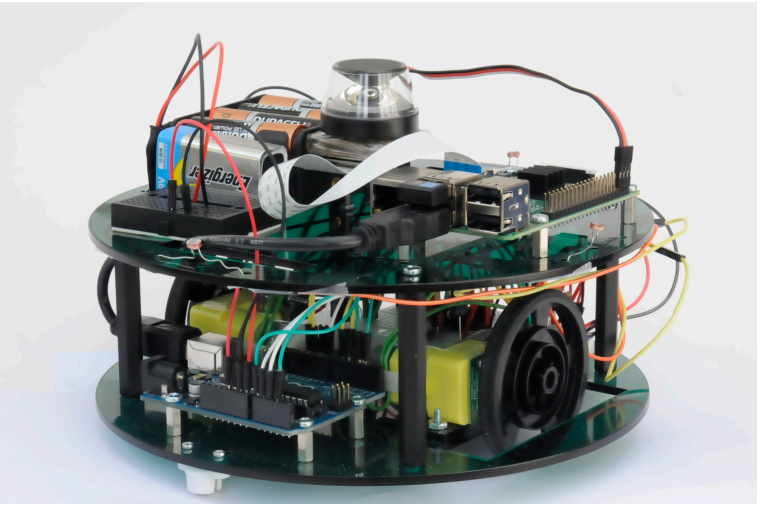


FIGURE 51 - ROBOT

Similarly to the simulation, the hardware implementation was based around differential-drive, where two wheels either side of the robot would allow it to move. Castors at the front and back would stop it from tipping over. A two-layer design was used for the robot's body, with the bottom layer, shown in figure 52, housing an Arduino used to control the robot's movement and sensing, along with the motors and main wiring. The top layer, shown in figure 53, seats the Raspberry Pi used to run the complex and expensive EKF algorithm, along with four light sensors and the robot's power supply.

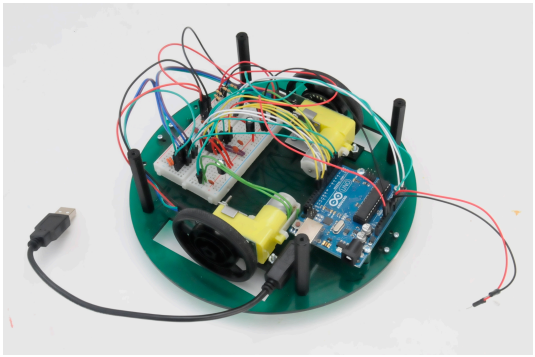


FIGURE 52 – ROBOT BOTTOM LAYER

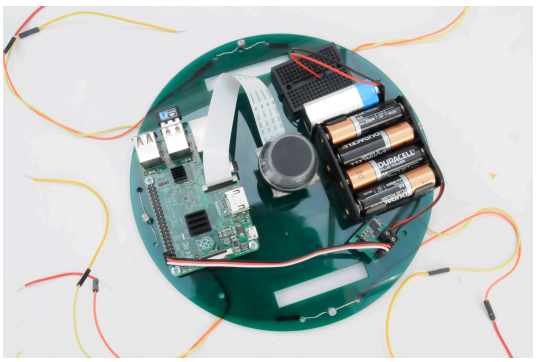


FIGURE 53 – ROBOT TOP LAYER

Optical rotary encoders, attached to the motors, allowed for accurate reading of the robot's motion for the motion model while four light sensors, on the top of the robot, allowed for a feature based sensory measurement to a single light feature in the world for the measurement model. A small Raspberry Pi camera was also attached to the top of the robot to allow for visual feature detection however, as discussed later, this was not fully implemented and instead left as a future project.

ARDUINO

To control the motors and read the sensors of the robot the Arduino was selected. Arduinos are programmable hobby microcontrollers that allow for basic digital and analog reading and writing. They are very popular for use with simple robotics projects due to their ease of use when interfacing with other electronics such as motors and sensors. They are not however full computers in the everyday sense of the word, as they do not provide an operating system, instead they must have programs compiled to them via a personal computer.

RASPBERRY PI

Because of the Arduino's slow processing speed and lack of an operating system it was decided it was not fast enough to run the EKF algorithm, instead a much faster Raspberry Pi 2 was used. The Raspberry Pi is a small but fully-fledged computer that runs its own Linux distro operating system – in this case Raspbian. Its processor is much more powerful and was felt capable of dealing with the EKF algorithm much more easily. As it ran a full operating system it was also possible to control remotely and wirelessly via SSH, meaning data it produced could be read instantaneously from a nearby laptop.

COMMUNICATION

Connection was needed between the Raspberry Pi and Arduino to allow for vital data to be sent back and forward. Originally the connection was made using I2C, which consists of a clock line and data line connecting the boards, the clock line being used to keep both boards in sync while the data line is used to send data between them. Problems occurred however and it was later discovered that the new Raspberry Pi 2 model had firmware issues regarding I2C. Instead a simple Serial connection was made with USB cable. While not as fast as I2C, it was found to be fast enough to meet the project's system requirements of 100 milliseconds, allowing for relatively small times between updating the internal model. The PySerial library was used with Python on the Raspberry Pi to allow for communication between the separate devices.

To communicate with the Arduino, simple binary pings of 1 or 0 were sent, with 1 requesting new data and 0 indicating the motors were to stop. Keeping this to a small binary digit ensured fast transfer and lowered latency. When data was requested, the Arduino sent back a string containing the measured motion command – translational v and rotational w velocities – of the robot, the sensory measurement – the range r and bearing ϕ to the light source – and the time since the last request dt. It was sent in the form of a delimited list as follows:

$$v, w; r, \phi; dt$$

This format allowed for the motion control, sensory measurement and time difference to be quickly extracted on the Raspberry Pi.

MOVEMENT

Two small, geared servomotors were used to control the movement of the robot. This simple differential drive configuration ensured a simple motion model would be needed, as more wheels would increase the model's complexity. The use of an H-Bridge allowed the Arduino to drive these motors both forwards and backwards, a simple function was also written to allow for independent control of each wheel, allowing each wheel power to be set to between -255 and 255, with 0 meaning brake.

GETTING MOTION COMMAND

To calculate the motion of the robot wheel's motors encoders were used. This allowed a more accurate estimate of how the robot was actually moving. Simply using the value sent to the motors suffered from a mismatch between the input and output due to friction on the wheels

slowing them down. Wheel encoders ignore this by measuring exactly how much each wheel turns. The encoders pinged each 128th of a turn (called one tick), as well as returning the direction of the turn. An interrupt was set up for each encoder on the Arduino, allowing each tick to interrupt the main program and update a global counter used to represent the ticks each wheel has turned. Each tick incremented or decremented this counter based on whether its direction was forwards or backwards. After each read of the motion command the counters were set back to 0.

By knowing the time since the last read along with how many ticks made a full turn, it was possible to calculate the angular velocities of each wheel:

$$V_R = \frac{\left(\frac{count_R}{64} * \pi\right)}{dt}$$

$$V_L = \frac{\left(\frac{count_L}{64} * \pi\right)}{dt}$$

Where V_R = the angular velocity of the right wheel, V_L = the angular velocity of the left wheel, $count_R$ = the number of ticks of the right wheel, $count_L$ = the number of ticks of the left wheel, 64 = the number of ticks for a full rotation of the wheel and dt = the time since the last read. With the angular velocities calculated, the same adapted conversions given by LaValle (2006) and used in the simulation were implemented to give the translation velocity v_t and rotational velocity w_t:

$$v_t = \frac{(V_R + V_L)}{2} r$$

$$w_t = \frac{V_R - V_L}{D} r$$

Where r = the radius of the wheels and D = the distance between the robots wheels. This then gave an estimated motion command u_t for the robot that would be used by the EKF:

$$u_t = \begin{pmatrix} v_t \\ w_t \end{pmatrix}$$

GET SENSORY MEASUREMENT

As in the simulation implementation, a feature based measurement model was used. This was chosen over a beam-based model, which uses a spinning laser range finder to estimate the exact shape of the world, as it reduces the dimensionality of the measurement by a huge degree, only needing to represent a few (in this case one) features as opposed to the detailed map given by the range finder. Feature based models have been used by many robots (Leonard, 1991; Thrun, 1998) but can be incredibly complex when using techniques such as computer vision to locate features in the world.

LIGHT SOURCE

The simpler version I chose to begin with was using a single light source in the world and calculating the distance to it using four light sensors attached to the top of the robot, one attached to each side. Artificial landmarks are easier to spot (Salichs, 1999, pp.231) and having only one of them meant there would be no need to confirm which feature on the map it corresponded to. Calculating the range to a light source has its own problems however, as light sensors do not drop linearly as they move away from a light source, this can be seen in figure 54 below.

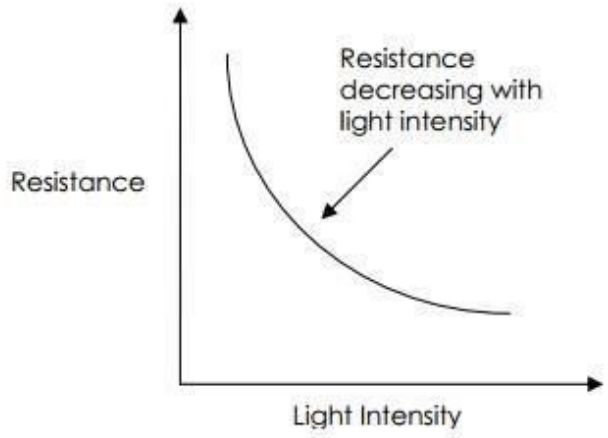


FIGURE 54 - HTTPS://WWW.KITRONIK.CO.UK/WP/WP-CONTENT/UPLOADS/2015/03/HOW_A-LIGHT_DEPEDANT_RESISTOR_WORKS_RESISTANCE_V_LIGHT_INTENSITY.JPG

Linear interpolation was used to map each sensor reading to a distance, due to this nonlinear drop however each light sensor was calibrated individually against a set of distances, giving a more accurate mapping between the sensor's value and the distance between it and the light source. This was done by first taking an average of 10 readings for each sensor, when the robot was placed with the sensor facing the light source, with the centre of the robot between 0cm and 300cm away in steps of 50cm. This produced a list of values that could be used to calculate a set of linear interpolants, roughly modelling the drop in light for each sensor when it was facing the light source. The recorded values are shown in table 1 below and the produced interpolation graph shown in figure 55.

TABLE 1 – LIGHT SENSOR CALIBRATION DATA

Range to Centre of Robot from Light Source (r)	Sensor Reading When Facing Light Source (y)			
	Back Sensor	Right Sensor	Front Sensor	Left Sensor
0cm	750	811	771	760
50cm	731	803	760	743
100cm	564	673	601	616
150cm	447	566	473	500
200cm	382	497	408	450
250cm	350	455	365	398
300cm	319	420	321	367

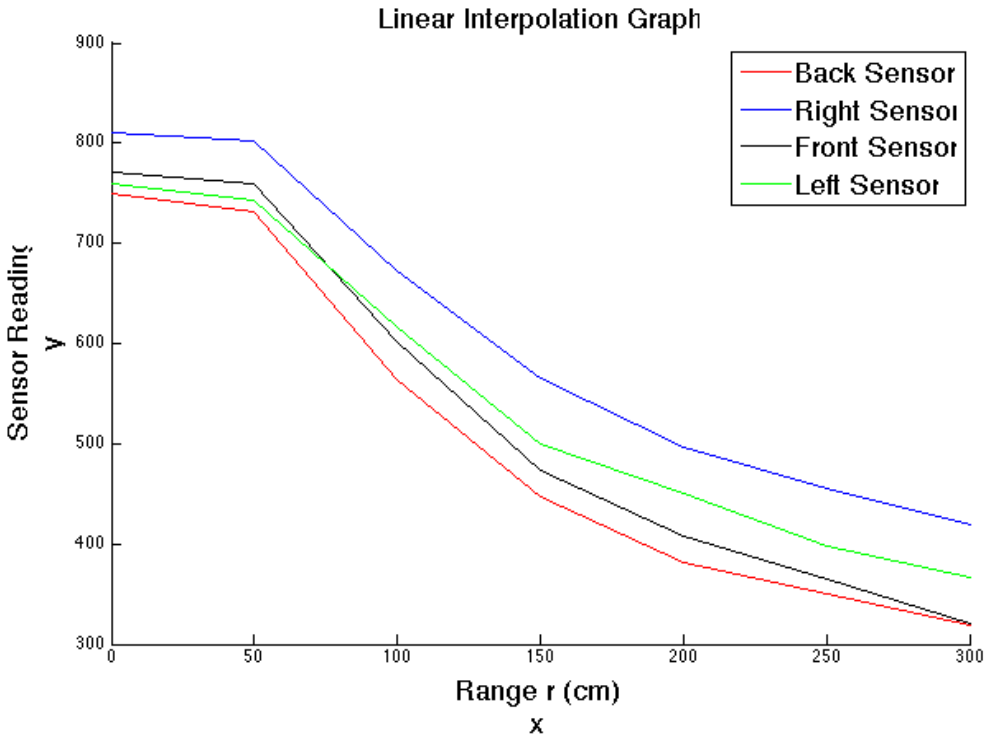


FIGURE 55 – LINEAR INTERPOLATION GRAPH OF SENSOR CALIBRATION DATA

To calculate this mapping, the sensor value was first compared to the data set to see which values it was between in its respective column. Linear interpolation was then done with these values used to create the single straight line it was mapped against:

$$r_t = x_0 + \frac{y - y_0}{\frac{y_1 - y_0}{x_1 - x_0}}$$

Where r = the calculated range to the light source, y = the light sensor reading, y_0 = the lower sensor reading value from the table, y_1 = the upper sensor reading value from the table, and x_0 and x_1 their ranges from the table respectively. To save processing time polynomial interpolation was not used, although it would produce a smoother interpolation graph, it would be more costly without providing too much accuracy. A comparison however may be a possible extension to consider at a later date.

With the range of each sensor calculated, the smallest was chosen. This was due to the smallest range being more likely to be from the sensor facing the light source. The other sensors, due to their position, were likely to give faulty readings from being blocked by other parts of the robot or by not facing the light source correctly. Earlier calibration was also done when the sensor was facing the light source, so sensors facing away would give un-calibrated values anyway.

The bearing ϕ_t to the light sensor was originally calculated by using trilateration to calculate the position of the light source in reference to the three closest light sensors; the bearing could then be worked out using simple trigonometry. The trilateration equations to produce the x and y position of the light source in relation to the robot along with the bearing from the robot to the light source are shown below:

$$x = \frac{r_1^2 - r_2^2 + d^2}{2d}$$

$$y = r_1^2 - r_3^2 - 2xi + i^2 + j^2$$

$$\widehat{\phi_t} = \tan^{-1}\frac{y}{x}$$

Where x and y = position of light source in relation to light sensors, r_1^2, r_2^2 and r_2^2 = the ranges from light sensors 1, 2 and 3 respectively, d = the x offset of the relative right most sensor, i and j = the x and y offset of the relative central sensor respectively and $\widehat{\phi}$ = the relative bearing of the robot. This relative bearing was then offset by either 0, $\frac{\pi}{2}$, $-\frac{\pi}{2}$ or π based on the sensor that was closest. This trilateration algorithm however faced the same problem mentioned above with sensors facing the wrong direction and being blocked. Large errors in the bearing were more common than not and a simpler method was adopted.

Instead of trilateration, the sensor deemed closest to the light source determined a rough estimate of the bearing. The back sensor gave a bearing of π, the right sensor gave a bearing of $-\frac{\pi}{2}$, the front sensor gave a bearing of 0 and the left sensor gave a bearing of $\frac{\pi}{2}$. This estimate would be off by a maximum of $\pm\frac{\pi}{4}$ as long as the correct side was chosen. This produced a much more stable estimate of the bearing. This then gave an estimated sensory measurement z_t for the robot that would be used by the EKF:

$$z_t = \begin{pmatrix} r_t \\ \phi_t \\ 1 \end{pmatrix}$$

Note that the signature has been set to 1 as it is the one and only feature.

CAMERA

The second feature based sensory measurement model attempted was using a 360-degree panoramic Kogeto Dot lens attached to the Raspberry Pi's camera to see coloured landmarks placed around the world. Unfortunately this was not completed due to initial problems trying to unfold the panoramic image was proving too slow. It would however have allowed for the use of multiple features in the environment that, as seen from the simulation model, greatly increases the robot's localisation accuracy.

Unfolding one of these polarised images gives a reasonable image of the world around the robot, feature detection could then be used to pick out the large coloured landmarks in the world and their position in the image would dictate their bearing to the robot, ranging from pi on the left to -pi on the right. Triangulation would then be able to predict the ranges to each feature with the feature's colour dictating which feature it corresponded to in the map (using the signature and property values).

The original problem faced, as discussed before, was due to the time it took to convert the taken image from a polar to Cartesian coordinate frame (unfolding it). Even with the fast processing power of the Raspberry Pi, this was not fast enough and a recommended algorithm found online (KScottZ, 2013) could take up to 5 seconds for each image. This would have meant vast amounts of time between each update of the EKF and would have lead to incredibly poor results. Later into the project it was realised that unfolding the image was unnecessary. Each features angle from the centre of the polar image could have been used to give the correct bearing. Due to lack of time however this was left as a future extension for the project.

6.2 Map of the Environment

As only a single light source was used, creating a map of the environment was simply placing the light feature at the origin (0, 0) of the map:

$$m = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

The property of the light feature was set to 1 to match the signature of the sensory measurement. Again, it is important to remember that the correspondences between features measured and in the map was not needed, as only one feature existed. As the floor under the light source was impassable due to a stand holding the light up, the robot's x, y starting position was set to 50, 50. Its bearing was also set to $\frac{\pi}{2}$ to ensure it was looking along the y-axis. This setup can be seen in a photo of the real world environment in figure 56.

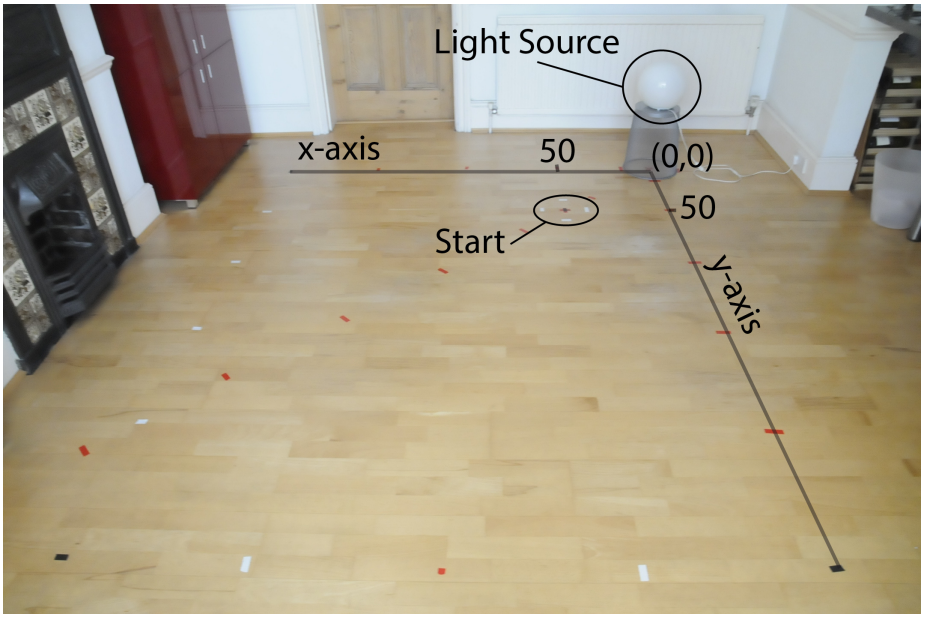


FIGURE 56 – LABELLED REAL WORLD SETUP OF ENVIRONMENT

6.3 Transferring EKF

Transferring the EKF from simulation to hardware was relatively easy. It was simply re-coded as a function in Python to allow for it to run on the Raspberry Pi and communicate with the Arduino easily. Linear algebra on Python was however tricky and Numpy matrices were used to store the data and the Math library used to allow for the required linear algebra.

6.4 Problems Encountered

A small fix was used to overcome the errors thrown when 0 translational velocity was recorded as detailed in section 4.4 above. When found to be 0, it was instead set to a very small decimal, hopefully causing little affect on the algorithm.

6.5 Calibration

Error values for the motion model were estimated at 0.1. The error values for the sensory measurement were again estimated as 0.1 however the bearing error ε_ϕ was set to 5 based on its large variance explained above. Trial and error was mainly used for the robot's calibration due to time constraints; however further testing could refine these values.

7 HARDWARE RESULTS

This section shows the results of running different localisation techniques on the purpose built robot that moves back and forward across the world. The robot was run for 1500 iterations of the localisation algorithms with each motor being set to a power of 200, its starting position being (50cm, 50cm) and its bearing being $\frac{\pi}{2}$. After every 200 iterations the robot would reverse its current direction, thereby ensuring it did not leave the confines of the map. This produced a noisy back and forth movement that can be seen in the video attached to this report.

Three error values were calculated for each technique, showing how it faired at localisation. First the average distance in cm between the robot and the prediction's final positions were calculated, then the average difference between the robot and the prediction's bearings were calculated. Finally, the average difference between the robot and the prediction's range from the light source was calculated. The average of 10 different runs for each technique was used for these errors.

7.1 COMPARISON

The figures 57 to 66 below show the final position of the robot along with the estimated positions of the robot given by the EKF and the motion model. The broken blue line highlights the position error between the EKF and the robot while the broken red line highlights the position error between the motion model and the robot. The magenta circle indicates the light source the EKF used as a landmark and the green star indicates the starting position of the robot.

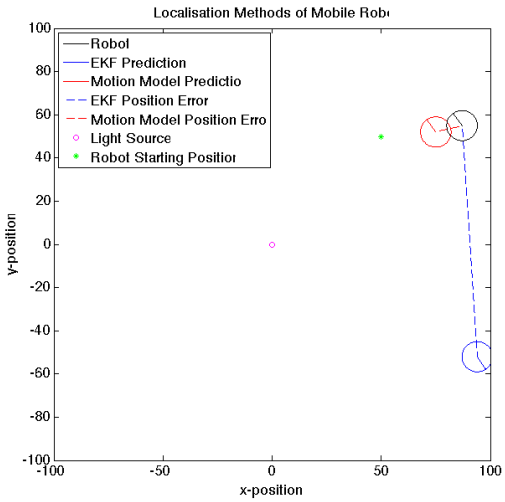


FIGURE 57 - RUN 1

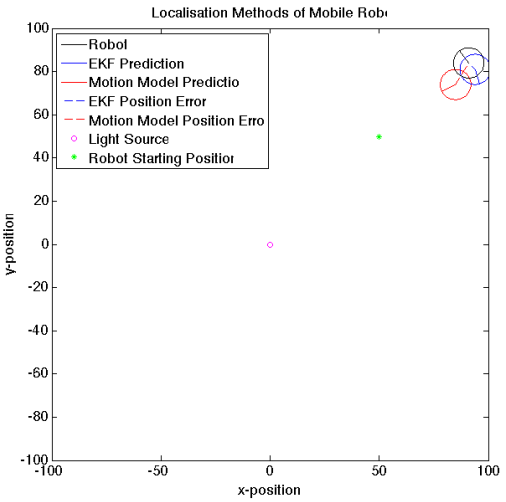


FIGURE 58 – RUN 2

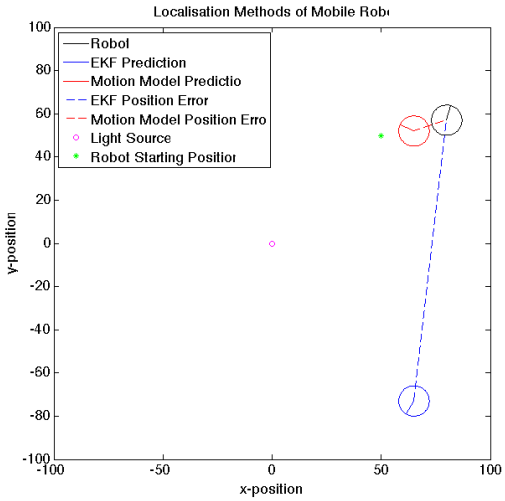


FIGURE 59 – RUN 3

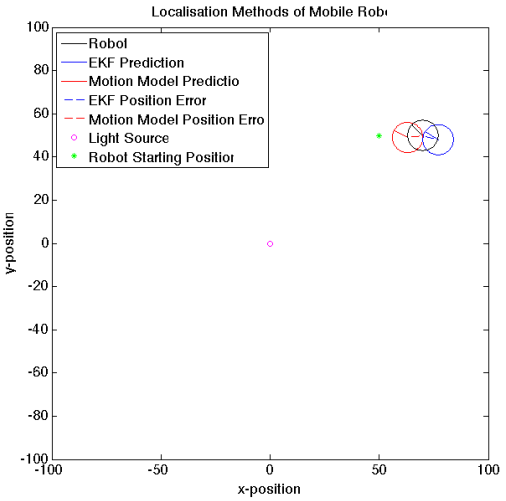


FIGURE 60 – RUN 4

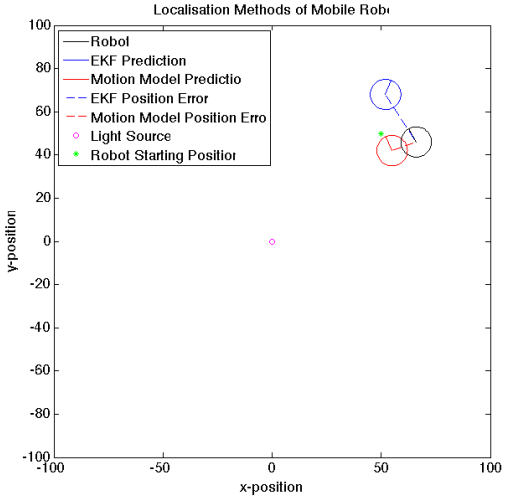


FIGURE 61 – RUN 5

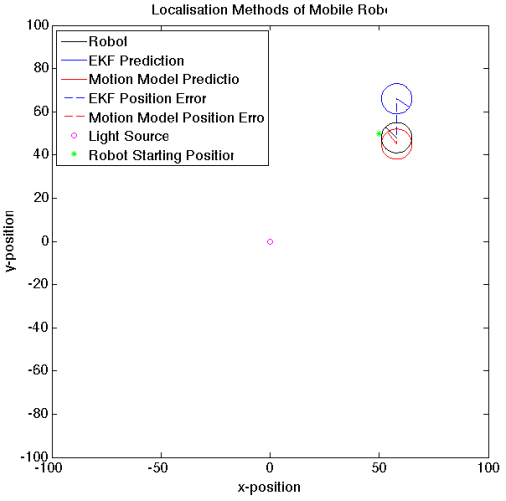


FIGURE 62 – RUN 6

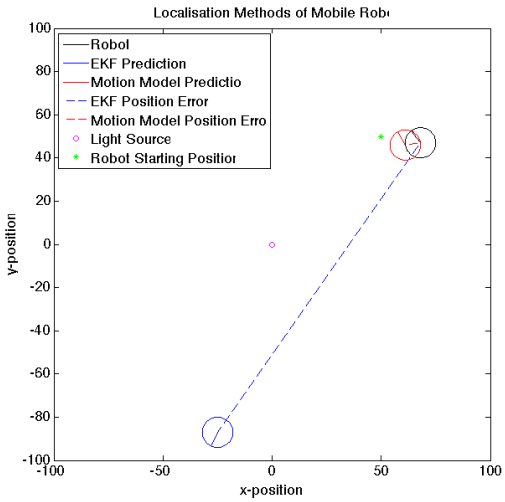


FIGURE 63 – RUN 7

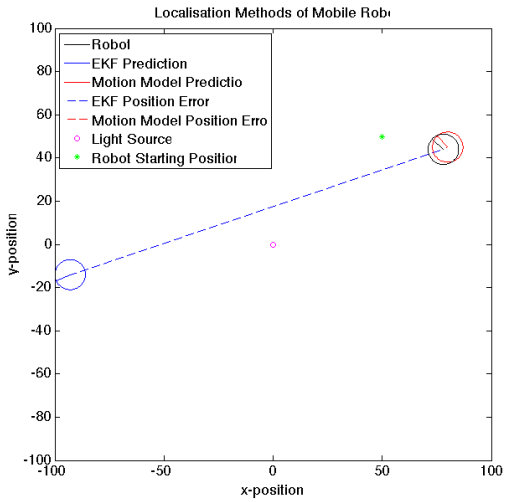


FIGURE 64 – RUN 8

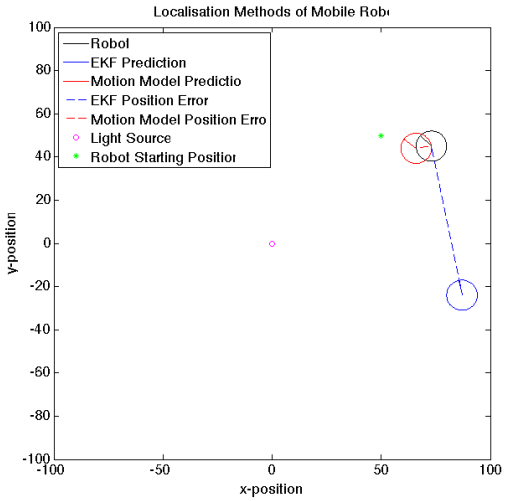


FIGURE 65 – RUN 9

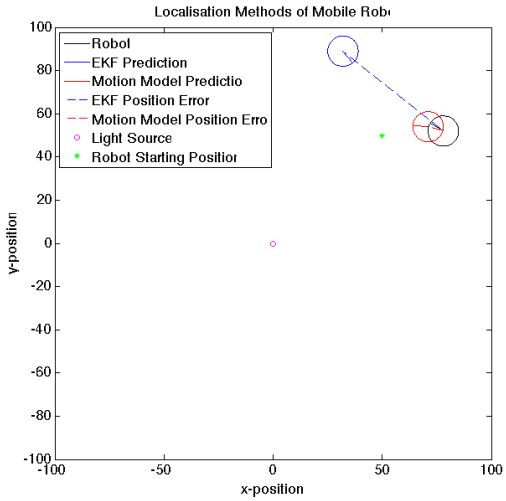


FIGURE 66 – RUN 10

Unfortunately first impressions of the EKF do not look good from looking at figures 57 to 66. In almost all runs it produced a result further away from the robot than the motion model, which did relatively well.

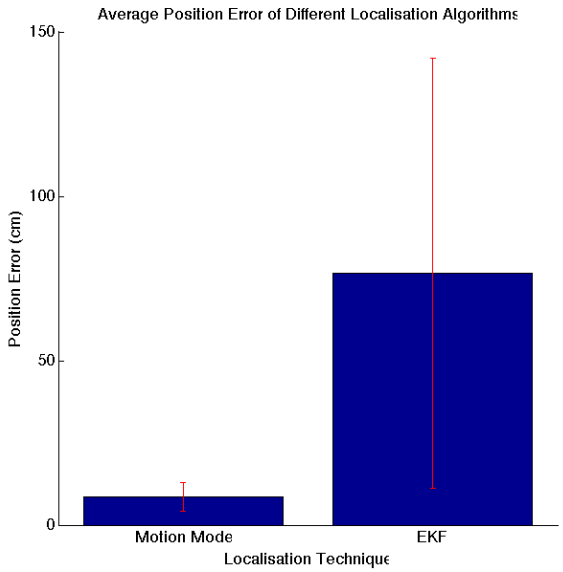


FIGURE 67 – POSITION ERROR OF MOTION MODEL VS EKF

In figure 67 the large position error of the EKF is made more obvious. While the average position error for the EKF reaches around 80cm, the motion model stays at around 10cm. The standard deviation of the EKF is also massive in comparison to the motion model, showing there are large amounts of variability in its effectiveness at localising the robot.

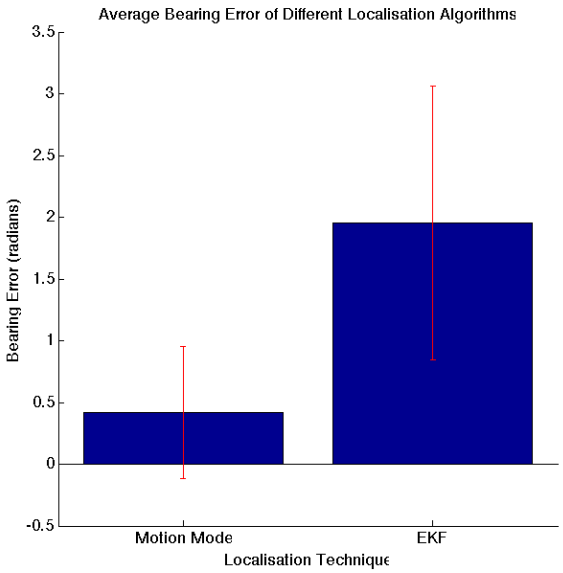


FIGURE 68 – BEARING ERROR OF MOTION MODEL VS EKF

Figure 68 above shows the average bearing error of each technique, showing the EKF has little idea of the correct bearing. The motion model again produces a surprisingly good result however. This has highlighted the problem with using the simple bearing calculator, described in section N, to calculate the bearing from the robot to the light source, where the bearing could be one of 4 values: $0, \frac{\pi}{2}, -\frac{\pi}{2}$ or π. It seems the small error this produced has grown to cause a very large deviation.

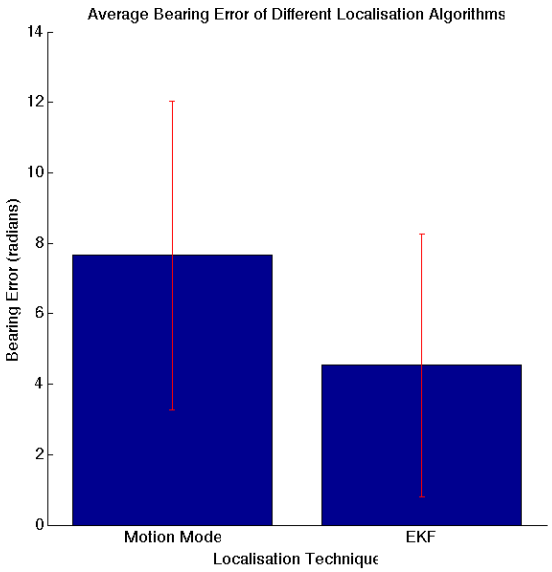


FIGURE 69 – RANGE ERROR OF MOTION MODEL VS EKF

While the EKF algorithm produced has given very bad localisation, figure 69 above shows it is very accurate at keeping the same distance away from the landmark as the real robot. The motion model however is not very good at this. It is likely this is due to the range calculated by the sensory measurement in the EKF being very accurate and keeping its predicted position somewhere along the circle this range creates around the landmark, as seen in section 4.3. The large error in the bearing however meant that although it was kept on the same circle, its point on that circle could deviate wildly.

7.2 EVALUATION

Analysis of the hardware implementation results shows similarities to the one feature EKF simulation produced however the problem with its measurement model, described in the results above and in section 4.3, has been greatly amplified. While able to stay at the same range from the light source as the real robot, its angle from the light source varies, as does its own bearing. Adding a second feature would help in this instance however, in practical terms, a second light source would be problematic. Interference between the light produced would have large negative effects on calculating the range to the light sources. Discovering which light corresponds to which feature would also be challenging due to there being no detectable difference with the current setup. Using a camera to track features would be of benefit here as the light sources could be replaced with coloured landmarks that are easily distinguishable.

A surprising result from this testing was how well the motion model did by itself. It is assumed that the wheel encoders negated a large amount of error that would have been given by simply using the power sent to each wheel, such as in the simulation. It would be interesting to compare the motion model based on the wheel turns given by the encoders and the power sent to each wheel.

LIMITATIONS

A major problem with running the EKF algorithm on hardware is amount of control that needs to be taken of the environment. Ensuring the position of the light source and the robot were correct were paramount, where small errors in this would have large affects later down the line. The requirement for the world to be fully modelled also posed the problem that this robot would only ever be able to work when the world has been accurately mapped, something that is rarely the

case in the real world. As discussed earlier, using light for the measurement model also vastly limits the accuracy of the EKF due to it only allowing one feature.

8 CONCLUSION

This project has provided an insight into the cutting edge field of probabilistic robotics. Almost all my original requirements were met and while the simulation produced demonstrated accurate localisation behaviour, it was interesting to see it fail in a real world robot. Further work into increasing the number of features the robot could measure would hopefully produce a more stable and accurate robot.

Throughout this report further steps that could be taken have constantly arisen. There are a number of avenues I wish to explore after the completion of this project, the first being to finish the camera based measurement model. I am hopeful that by increasing the number of features detected the bearing of the measurement model will be more accurate and better localisation behaviour will be produced. It would also be of interest to implement "return to base" behaviour, where after localisation the robot would be able to return to its starting position. This was suggested earlier in the project however it was deemed a trivial step and ignored. However further discussion with my supervisor has brought me to the realisation that attempting to move back to the original position would be noisy and the robot would be very likely to miss. This would instead require constant updating of the motors speed based on information given by the EKF.

Working on this project has allowed me to spend time expanding my knowledge of Bayesian based mathematics, a topic I am keen to continue with as postgraduate research, as well as introducing me to the world of hobby electronics and robotics.

9 BIBLIOGRAPHY

- Borenstein, J., Everett, H., Feng, L., Wehe, D., 1996, *Navigating Mobile Robots: Systems and Techniques*, A.K. Peters Ltd, Wellesley MA
- Borenstein, J., Everett, H., Feng, L., Wehe, D., 1997, *Mobile Robot Positioning – Sensors and Techniques*, In Journal of Robotic Systems: Special Issue on Mobile Robots, 14: 231-249
- Cox, I., Wilfong, G., 1990, *Autonomous Robot Vehicles*, Springer Verlag
- Cox, J., 1991, *Blanche – An Experiment in Guidance and Navigation of an Autonomous Robot Vehicle*, in IEEE Transactions on Robotics and Automation, 7: 193-204
- Dellaert, F., Fox, D., Burgard, W., Thrun, S., 1999, *Monte Carlo Localization for Mobile Robotics*, In Proceedings of the International Conference on Robotics and Automation (ICRA)
- Feng, L., Borenstein, J., Everett, H., 1994, *"Where am I?" Sensors and Methods for Autonomous Mobile Robot Positioning*, Technical Report UM-MEAM-94-12, ANN Arbor MI
- Julier, S., Uhlmann, J., 1997, *A New Extension of the Kalman Filter to Nonlinear Systems*, In International Symposium on Aerospace/ Defence Sensing, Simulate and Controls, Orlando, FL
- Kalman, R., 1960, *A New Approach to Linear Filtering and Prediction Problems*, In Transactions of the ASME–Journal of Basic Engineering, 82D: 35-45
- LaValle, S., 2006, *Planning Algorithms*, Cambridge University Press, NY
- Leonard, J., Durrant-Whyte, H., 1991, *Mobile Robot Localisation by Tracking Geometric Beacons*, in IEEE Transactions on Robotics and Automation, 7: 376-382
- Maybeck, P., 1979, *Stochastic Models, Estimation and Control*, Academic Press, NY
- Metropolis, N., Ulam, S., 1949, *The Monte Carlo Method*, Journal of American Statistical Association, 44: 335-341
- Murphey, R., 2000, *Introduction to AI Robotics*, MIT Press, Cambridge MA
- Salichs, M., Moreno, L., 2000, *Navigation of Mobile Robots: Open Questions*, Robotica, 18: 227-234
- Thrun, S., 1998, *Finding Landmarks for Mobile Robot Navigation*, In IEEE International Conference on Robotics and Automation (ICRA)
- Thrun, S., Burgard, W., Fox, D., 2006, *Probabilistic Robotics*, London: MIT Press
- West, M., Harrison, P., *Bayesian Forecasting and Dynamic Models*, 2ed, Springer Verlag, NY

- BCS Code of Conduct, 2011, BCS, [online] Available at: <http://www.bcs.org/category/6030>
- Dewarped Panoramic Images From a RaspberryPi Camera Modle, 2013, KScottZ, [online] Available at: <http://www.kscottz.com/dewarped-panoramic-images-from-a-raspberrypi-camera-module/>
- Fighting Fire With Data, Spacecraft and Drones, 2012, CNN, [online] Available at: <http://edition.cnn.com/2012/07/26/tech/innovation/technology-fighting-fire/> [Accessed 1 November 2014]
- How Paro the robot seal is being used to help UK dementia patients, 2014, The Guardian, [online] Available at: <http://www.theguardian.com/society/2014/jul/08/paro-robot-seal-dementia-patients-nhs-japan> [Accessed 1 November 2014]

10 APPENDICES

10.1 MEETING LOG

- Summer Discussions
  - September 2014
  - Discussed project fundamentals and possible deliverables.
  - Assigned reading by supervisor of early chapters in Probabilistic Robotics (Thrun, Burgard and Fox, 2007) to help me decide if I was interested in the topic.
  - Assigned implementation of simulated low inertia wheeled robot
- Meeting 1
  - 7/10/14
  - Discussed and created initial project plan
  - Discussed and assigned implementation of simulated robot
  - Discussed and assigned implementation of velocity motion model
- Meeting 2
  - 31/10/14
  - Discussed problems found with implementation of velocity motion model
  - Equations in book (Thrun, Burgard and Fox, 2007) do not allow for 0 rotational velocity or give probability of location staying the same.
  - Discussed interim report content
  - Discussed and assigned posterior distribution graphs of robot's pose after motion command
- Meeting 3
  - 16/11/14
  - Asked to tune and calibrate alphas
  - Asked to finish motion models
- Meeting 4
  - 21/11/14
  - Discussed interim report
- Meeting 5
  - 26/11/14
  - Finished motion model
  - Discussed motion model
- Meeting 6
  - 04/12/14
  - Discussed use of robot
  - Discussed movement and feature extraction
- Meeting 7
  - 11/12/14
  - Discussed basics of Kalman filters
- Meeting 8
  - 22/1/15
  - Discussed feature extraction and sensory measurements
- Meeting 9
  - 5/2/15
  - Discuss EKF
  - Asked to start EKF
- Meeting 10
  - 12/2/15
  - Discussed report
  - Asked to start intro to report
- Meeting 11

    - 19/2/15
    - Discussed neat plotting of models
    - Discussed how motion model fits in EKF
- Meeting 12
    - 26/2/15
    - Discussed limitations of single feature EKF
    - Discussed Taylor expansion
- Meeting 13
    - 5/3/15
    - Discussed plotting and calibration of EKF
    - Discussed poster
- Meeting 14
    - 26/3/15
    - Discussed poster
- Meeting 15
    - 2/4/15
    - Discussed report
    - Discussed implementation of robot
- Meeting 16
    - 22/4/15
    - Discussed draft of report
- Meeting 17
    - 30/4/15
    - Final discussion of report