Localisation: Where Am I?

Shane Reynolds

January 31, 2019

Contents

1 Introduction

Suppose a robot has an unknown pose in an environment for which it has a map. The robot takes sensor readings, and based on these observations, must infer a set of poses where it could be located in the environment. In robotics, this scenario is known as the *localisation problem* [1, 2]. In simpler terms, localisation is the problem of estimating a mobile robot's location and orientation relative to its environment, given sensor data [3]. This is not as straight forward as it appears given robot actuators are subject to small random perturbations in performance, and sensors provide imperfect measurement. Compounding these problems is the fact that the robot may not even know it's initial pose relative to the environment. This paper explores two approaches to solving the localisation problem, after providing a brief discussion on the common variations of the problem. The first of these solutions is the Extended Kalman Filter (EKF): an adaptation of the Kalman Filter, suited to the estimation of non-linear system responses. The second solution is a particle filter method called Monte Carlo Localisation (MCL). The paper concludes with an application of MCL in a Gazebo simulation using an off-the-shelf MCL package in ROS. The MCL implementation is trialled across two different robot models providing opportunities for discussion of implementation robustness, and an analysis of parameter tuning for reliable performance.

2 Background

The mobile robot localisation problem comes in three flavours. The simplest involves tracking the robot's pose relative to its environment - called *position tracking.* This scenario requires an initial known estimate of the robot's pose relative to the environment. Robot actions require pose updates using noisey sensor data [4]. Often initial pose estimates are unavailable - this scenario represents a more meaningful problem version called the *global localisation problem.* The goal here is for the robot determine it's pose from scratch, given it is unaware of the initial pose. Finally, the most challenging problem type is referred to as the *kidnapped robot problem* which sees a localised robot tele-ported to another location on the map without knowledge of the move. This is different to the *global localisation problem* scenario because, after tele-porting, the robot incorrectly believes it is somewhere other than its current location. This final scenario is used to test whether a localisation algorithm can recover from a catastrophic failure.

2.1 Kalman Filters

Kalman Filters can be used to solve the simplest localisation problem: position tracking. In an ideal world, position tracking is a trivial task in which the robot knows it's starting location, and updates position with perfect actuation and perfect sensor readings. In the real world, however, this is not the case. Actuation is not perfect, and is subject to minor perturbations, or wheel slippage can occur. Furthermore, sensor readings are noisey. The implications are that both movement and measurement are imprecise and subject to stochastic errors. Consider the a mobile robot moving along a one dimensional trajectory. Let the discrete time position be x_j, at time j. We note that the position at time j is dependent on the previous position at time $j-1$, denoted as x_{j-1}, plus any movement action taken by the robot, denoted as u_j. Additionally, as previously noted, the robot's actuators are not perfect and are subject to noise, ω_j. We can write x_j as follows:

$$x_j = ax_j + bu_j + \omega_j \tag{1}$$

This description is sometimes easier to understand when visualised as a computational block diagram, shown in Figure 1.

Almost always random phenomena are hidden from the observer behind a dynamical system - Kalman described this as follows:

A random function of time may be thought of as the output of a dynamic system excited by an independent Gaussian random process. [5]

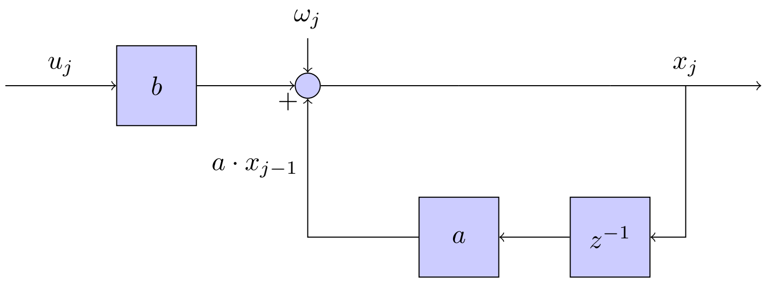

Figure 1: text

This is true for robotic motion, given that we can only attempt to estimate the robot's stochastically perturbed motion, through the use of sensors like odometry, or laser range finders, which are subject to stochastic noise. Letting the measurement of signal x_j be represented by z_j, we can write the following equation defining the measurement of x_j, subject to noise ν_j, as:

$$z_j = hx_j + \nu_j \tag{2}$$

Equation (2) can be represented in a computational graph as follows:

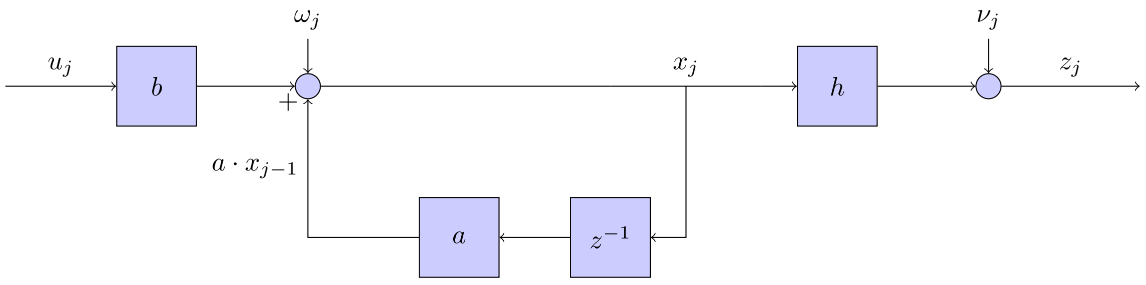

Figure 2: text

The fundamental problem that the Kalman Filter tries to answer is: how can an accurate estimate of a hidden stochastic signal be found when it is observed as the output of a dynamical system? The first step to answering this is to create a mathematical model of the system we can use to find an estimate of the measurement, z_j, which we call $\hat{z}_j$. This estimate is based on an estimation of the hidden variable, which we denote with $\hat{x}_j^-$. The mathematical model that provides these estimates does not factor in stochasticity seen in actual signals x_j and z_j. This is the *a priori* estimate of x_j, and is mathematically described as:

$$\hat{x}_j^- = a\hat{x}_{j-1} + bu_j \tag{3}$$

The *a priori* estimate is used to predict an output estimate, $\hat{z}_j$. In turn, $\hat{z}_j$, is used to estimate the difference between predicted signal and the observed signal, referred to as the residual:

$$z_j - \hat{z}_j = z_j - h\hat{x}_j^- \tag{4}$$

If the residual is small, then our estimate is good. If it is large then our estimate is not good. We can use the residual in (4) to update our *a priori* estimate, $\hat{x}_j^-$:

$$\hat{x}_j = \hat{x}_j^- + k(z_j - h\hat{x}_j^-) \tag{5}$$

A visualisation of the computational architecture for the process can be seen in Figure 3. This is a useful picture as it allows us to see the Kalman filter process decomposed into two distinct phases:

1. the state prediction phase in which the residual is used to update the *a priori* providing a state prediction, known as the *posterior belief*

2. the measurement update phase in which the *posterior belief* is updated with control actions u_j and used as the new *a priori*, which is combined with sensor readings to compute the new residual.

,

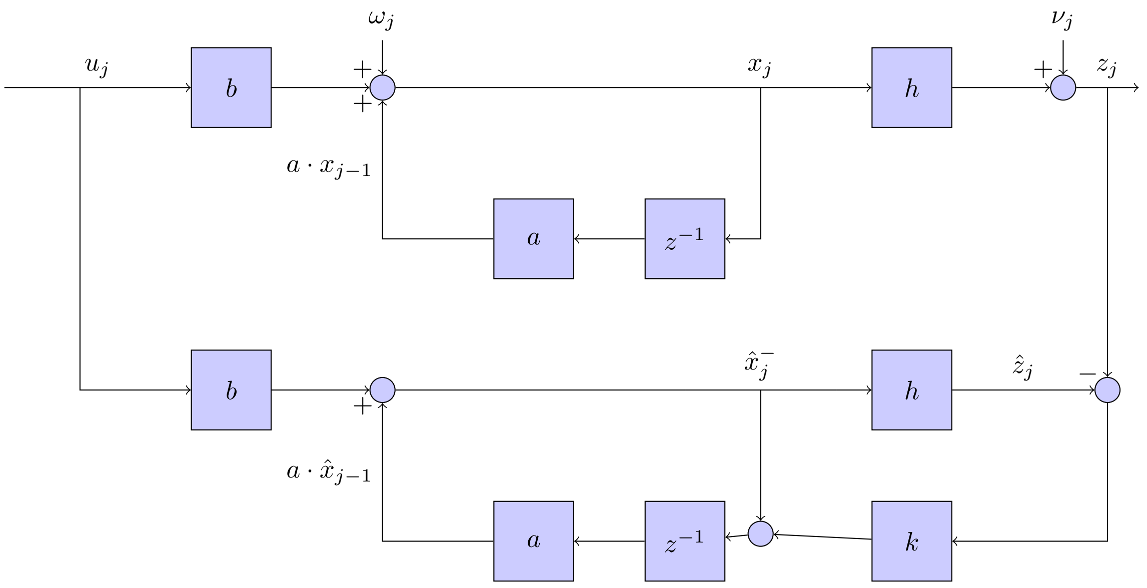

Figure 3: text

The task of determining k, used to refine our estimate, is at the heart of the Kalman filtering process. This is not a trivial task. Further, our problem is complicated when considering robotic motion in multiple dimensions - so far, estimation has only been considered for a 1D problem. Most mobile robots move in 2D, and we may also want to estimate other important locomotion variables such as velocity. Suppose that state was now represented as vector of variables such as position in 2D, including velocity. We define this using bold typeface $\mathbf{x}_j$. Similarly, control actions are represented as $\mathbf{u}_j$, and the measurement for each state variable in $\mathbf{x}_j$ is represented by $\mathbf{z}_j$. Without loss of generality equation (1) becomes:

$$\mathbf{x}_j = \mathbf{A}\mathbf{x}_j + \mathbf{B}\mathbf{u}_j + \boldsymbol{\omega}_j \tag{6}$$

Similarly, equation (2) becomes:

$$\mathbf{z}_j = \mathbf{H}\mathbf{x}_j + \boldsymbol{\nu}_j \tag{7}$$

The *a priori* estimate of the state variable, shown in equation (3), in multidimensional form, is given by:

$$\hat{\mathbf{x}}_j^- = \mathbf{A}\hat{\mathbf{x}}_j + \mathbf{B}\mathbf{u}_j \tag{8}$$

The *a priori* update, shown in equation (5) is given by:

$$\hat{\mathbf{x}}_j = \hat{\mathbf{x}}_j^- + \mathbf{K}_j(\mathbf{z}_j - \mathbf{H}\hat{\mathbf{x}}_j^-) \tag{9}$$

The error for the *posterior*, $\mathbf{e}_j$, can be written as:

$$\mathbf{e}_j = \mathbf{x}_j - \hat{\mathbf{x}}_j \tag{10}$$

A common way to measure the accuracy of $\mathbf{e}_j$ is using the covariance matrix, denoted in the literature as $\mathbf{P}_j$. This can be expressed as follows:

$$\mathbf{P}_j = \mathbb{E}[\mathbf{e}_j \cdot \mathbf{e}_j] \tag{11}$$

The Kalman filter attempts to select the Kalman Filter gain, $\mathbf{K}$, such that the *posterior* covariance is minimised. Applying a classical optimisation technique for multivariate calculus, we arrive at the following expression:

$$\frac{\partial \mathbf{P}_j}{\partial \mathbf{K}_j} = \frac{\partial \mathbb{E}[(\mathbf{x}_j - \hat{\mathbf{x}}_j)(\mathbf{x}_j - \hat{\mathbf{x}}_j)^T]}{\partial \mathbf{K}_j} = 0 \tag{12}$$

Solving equation (12) yields the following result for the Kalman filter gain:

$$\mathbf{K}_j = \frac{\mathbf{P}_j \mathbf{H}^T}{\mathbf{H}\mathbf{P}_j\mathbf{H}^T + \mathbf{R}} \tag{13}$$

The denominator of equation (13) contains the measurement noise, represented by matrix $\mathbf{R}$. In fact, the denominator is an important calculation step that maps the state prediction covariance, $\mathbf{P}_j$, into the measurement space - often this is expressed with the variable $\mathbf{S}$, that is:

$$\mathbf{S} = \mathbf{H}\mathbf{P}_j\mathbf{H}^T + \mathbf{R} \tag{14}$$

In addition to the *posterior* error, $\mathbf{e}_j$, there is an error for the *a priori*, denoted $\mathbf{e}_j^-$. The covariance matrix for $\mathbf{e}_j^-$ is given by $\mathbf{P}_j^-$. The final step to arriving at an algorithm for the Kalman filter is being able to recursively update both the *posterior* and *a priori* covariance matrices. The *a priori* error covariance update is derived from considering the error expression $\mathbf{x}_j - \hat{\mathbf{x}}_j^-$, and is expressed using the movement noise matrix $\mathbf{Q}$. The expression is as follows:

$$\mathbf{P}_j^- = \mathbf{A}\mathbf{P}_{j-1}\mathbf{A}^T + \mathbf{Q} \tag{15}$$

The *posterior* update can be derived from equation (11), and is expressed as:

$$\mathbf{P}_j = (\mathbf{I} - \mathbf{K}_j\mathbf{H})\mathbf{P}_j^- \tag{16}$$

Equations (8), (9), (13), (14), (15), and (16) make up the Kalman Filter algorithm. A single pass of the Kalman Filter algorithm can be seen in Algorithm 1 - the pass starts with the *a priori* state estimate update, and concludes with the *posterior* error covariance update. The full algorithm would see the code run continuously, in a loop, iteratively updating state and covariance estimations.

Algorithm 1 Kalman Filter

```
1: $\hat{\mathbf{x}}' \leftarrow \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}\mathbf{u}$ (A Priori State Estimate Update)
2: $\mathbf{P}' \leftarrow \mathbf{A}\mathbf{P}\mathbf{A}^T + \mathbf{Q}$ (A Priori Error Covariance Update)
3: $\mathbf{S} \leftarrow \mathbf{H}\mathbf{P}'\mathbf{H}^T + \mathbf{R}$ (Covariance Update In Measurement Space)
4: $\mathbf{K} \leftarrow \mathbf{P}'\mathbf{H}^T\mathbf{S}^{-1}$ (Kalman Gain Update)
5: $\mathbf{x} \leftarrow \mathbf{x}' + \mathbf{K}(\mathbf{z} - \mathbf{H}\mathbf{x}')$ (Posterior State Estimate Update Using Kalman Gain)
6: $\mathbf{P} \leftarrow (\mathbf{I} - \mathbf{K}\mathbf{H})\mathbf{P}'$ (Posterior Error Covariance Update)
```

NEED TO TALK ABOUT EXTENDED KALMAN FILTER

2.2 Particle Filters

Particle filters can be used to solve the global localisation problem, that is, it can localise a robot in an environment when its initial position is unknown. This section looks at a specific type of particle filter known as Monte Carlo Localisation (MCL). The central idea of MCL is the estimation of a posterior distribution across robot poses, often referred to as the *belief.* The *belief* is defined as probability density over the pose state space, which is conditioned on sensor measurement data, and odometric action data. Mobile robot pose, $\mathbf{x}$, is made up of location coordinates x and y and orientation θ. Letting observation from a laser range finder be represented by $\mathbf{o}$, and odometric sensor data by represented by $\mathbf{a}$, we can mathematically express the belief as follows:

$$Bel(\mathbf{x}_t) = p(\mathbf{x}_t|\mathbf{o}_t, \mathbf{a}_{t-1}, \mathbf{o}_{t-1}, \mathbf{a}_{t-2}, \ldots, \mathbf{o}_0) \tag{17}$$

Particle filters estimate belief recursively, meaning that an update rule from time t to time $t+1$ is required, and an initial belief is also needed. The initial belief characterizes the initial knowledge about the system state - if little is known about the initial system state, then a uniform distribution can be used over the state space. To derive the update equation we start by expressing (17) using Bayes rule:

$$Bel(\mathbf{x}_t) = \frac{p(\mathbf{o}_t|\mathbf{x}_t, \mathbf{a}_{t-1}, \ldots, \mathbf{o}_0)\ p(\mathbf{x}_t|\mathbf{a}_{t-1}, \ldots, \mathbf{o}_0)}{p(\mathbf{o}_t|\mathbf{a}_{t-1}, \ldots, \mathbf{o}_0)} \tag{18}$$

The denominator is a constant value relative to $\mathbf{x}_t$, and (4) can be written more compactly as:

$$Bel(\mathbf{x}_t) = \eta\ p(\mathbf{o}_t|\mathbf{x}_t, \mathbf{a}_{t-1}, \ldots, \mathbf{o}_0)\ p(\mathbf{x}_t|\mathbf{a}_{t-1}, \ldots, \mathbf{o}_0) \tag{19}$$

where η is a constant value:

$$\eta = p(\mathbf{o}_t|\mathbf{a}_{t-1}, \ldots, \mathbf{o}_0)^{-1} \tag{20}$$

A recursive relation seeks to provide an update at each time step. Information from the past is factored into the belief as changes occur. Therefore it can be reasoned that only new information presented to the system from current state is important when modifying the belief. This simplifying assumption, referred to as the *Markov assumption*, suggests that future data is independent of past data given knowledge of the current state. Mathematically, this allows equation (5) to be re-written as:

$$Bel(\mathbf{x}_t) = \eta\ p(\mathbf{o}_t|\mathbf{x}_t)\ p(\mathbf{x}_t|\mathbf{a}_{t-1}, \ldots, \mathbf{o}_0) \tag{21}$$

Partitioning over $\mathbf{x}_{t-1}$ equation (7) can be written as:

$$Bel(\mathbf{x}_t) = \eta\ p(\mathbf{o}_t|\mathbf{x}_t) \int p(\mathbf{x}_t, \mathbf{x}_{t-1}|\mathbf{a}_{t-1}, \ldots, \mathbf{o}_0)\ d\mathbf{x}_{t-1} \tag{22}$$

Applying Bayes' equation (8) can be written as:

$$Bel(\mathbf{x}_t) = \eta\ p(\mathbf{o}_t|\mathbf{x}_t) \int p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{a}_{t-1}, \ldots, \mathbf{o}_0)\ p(\mathbf{x}_{t-1}|\mathbf{a}_{t-1}, \ldots, \mathbf{o}_0)\ d\mathbf{x}_{t-1} \tag{23}$$

Exploiting the Markov property again the first expression in the integral can be simplified. The second expression can be written using the *Bel* notation as follows:

$$Bel(\mathbf{x}_t) = \eta\ p(\mathbf{o}_t|\mathbf{x}_t) \int p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{a}_{t-1})\ Bel(\mathbf{x}_{t-1})\ d\mathbf{x}_{t-1} \tag{24}$$

There are two key components to equation (10): the first is probability $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{a}_{t-1})$ which is referred to as the *motion model*; and the second is the probability $p(\mathbf{o}_t|\mathbf{x}_t)$ which is referred to as the *sensor model.*

2.2.1 Motion model $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{a}_{t-1})$ and sensor model $p(\mathbf{o}_t|\mathbf{x}_t)$

Under noise free ideal conditions the robot moves from state $\mathbf{x}_{t-1}$ to state $\mathbf{x}_t$ with certainty given some control action $\mathbf{a}_{t-1}$ - the assumption of idealness allows the use of kinematic equations to fully describe the robot's motion. As previously discussed, however, physical robot motion is subject to uncertainty since actuators are not ideal, which means there is uncertainty in pose $mathbfx_t$. The *motion model*, $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{a}_{t-1})$, describes a posterior density over successors to pose $\mathbf{x}_{t-1}$. A graphical depiction of what this looks like can be seen in Figure XXXX, which shows the distribution over where the robot could be located given some control action. The image on the right shows the distribution of the robot location after moving 40 meters. The picture on the right shows a more complicated path. The darker regions of the distribution indicate more likely robot locations.

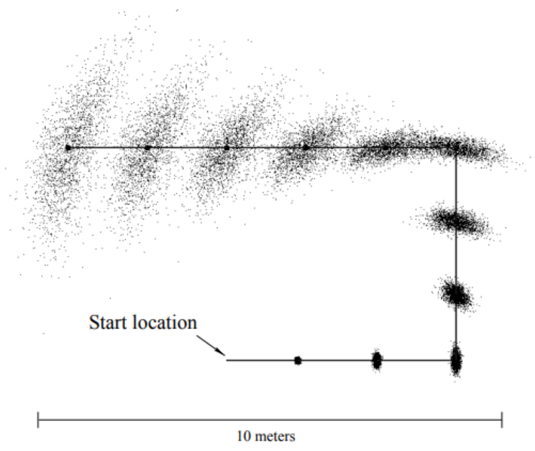

Figure 4: text

The sensor model provides the probability, or likelihood, of obtaining the sensor measurements, $\mathbf{o}_t$, given a robot pose, $\mathbf{x}_t$ and control action $\mathbf{a}_{t-1}$, assuming that we have a known map. Sensors such as laser range finders are made up of multiple beams.

2.2.2 MCL Implementation

The state space for mobile robot localisation is continuous, meaning that implementing the recursive localisation equation shown in (24) is computationally intractable. Particle filter algorithms attempt to represent the belief $Bel(\mathbf{x}_t)$ using a set of m weighted samples distributed according to $Bel(\mathbf{x}_t)$. Mathematically, we can express this as:

$$Bel(\mathbf{x}_t) \approx \{\mathbf{x}^{(i)}, \omega^{(i)}\}_{i=1,\ldots,m} \tag{25}$$

The sampled particles, $\mathbf{x}^{(i)}$, are discrete hypotheses of the robot's pose drawn from $Bel(\mathbf{x}_t)$. The $\omega^{(i)}$, called the *importance factor*, are derived based on how likely the hypothesis particle is, given the sensor measurements and known map. The idea is that over time the set of sample particles converge on the true robot pose in the environment. This means that the algorithm needs to discard particles that are unlikely, and keep particles that are likely. Indeed the MCL algorithm can be thought of as featuring two distinct components:

1. **Motion and sensor update**: this phase sees the robot undergo a control action which is applied to its pose, as well as the pose of all particles. Additionally, sensor measurements are obtained, and then these sensor readings are used to determine new likelihood weights, ω, for each of the particles.

2. **Resampling**: this phase allows us to discard particles which are not likely (low weights), and keep particles with high likelihood (i.e. large weights). This is undertaken by randomly

sampling m times from the pool of particles, with replacement. Higher ω weights mean it is more likely the particle will feature in the re-sampled set.

The MCL algorithm pseudocode is shown in Algorithm 2 below.

Algorithm 2 Monte Carlo Localisation

```
1: procedure MCL($X_{t-1}$,$u_t$,$z_t$)
2:     $\bar{X}_t = X_T = \emptyset$
3:     for $m = 1$ to $M$ do
4:         $x_t^{[m]}$ = motion_update($u_t, x_{t-1}^{[m]}$)
5:         $\omega_t^{[m]}$ = sensor_update($z_t, x_t^{[m]}$)
6:     for $m = 1$ to $M$ do
7:         draw $x_t^{[m]}$ from $\bar{X}_t$ with probability $\propto \omega_t^{[m]}$
8:         $X_t = X_t + x_t^{[m]}$
9:     return $X_t$
```

2.3 Comparison & Model Selection

Compare the two approaches and determine which approach was implemented to provide localisation for the two robot models.

3 Simulations

Gazebo, a physical simulation environment, was used to test an off-the-shelf implementation of MCL on two different mobile robot platforms. The first robot model was a benchmark designed to provide easy ROS implementation of the MCL package. Further, the benchmark model utilised existing xml interfacing for Gazebo which guaranteed performance of the MCL package, for appropriate hyper-parameter selection. The second robot model was designed to assess how different physical locations for sensors might impact the performance of MCL. Physical dimensions of the robot were also altered, compared to the benchmark, to assess further performance changes in MCL.
WHAT WERE THE TESTS THAT WERE UNDERTAKEN - BASICALLY, A LOCATION WAS SET FOR THE ROBOT AUTOMATICALLY AND THE TEST WAS SATISFIED

3.1 Benchmark Model

3.1.1 Model Design

PROVIDE A BRIEF DESCRIPTION OF THE ROBOT MODEL INCLUDING SIZE AND SENSOR LAYOUT TALK ABOUT THE LOCOMOTION DRIVERS FOR THE ROBOT
PROVIDE A PICTURE OF THE ROBOT MODEL

3.1.2 Packages Used

TALK ABOUT THE PACKAGES THAT WERE USED FOR THE ROBOT AND THEIR CONFIGURATION

3.1.3 Parameters

TALK ABOUT THE HYPERPARAMETERS USED FOR THE

3.2 Personal Model

PROVIDE A BRIEF DESCRIPTION OF THE ROBOT MODEL INCLUDING THE SIZE AND SENSOR LAYOUT TALK ABOUT THE LOCOMOTION DRIVERS FOR THE ROBOT PROVIDE A PICTURE OF THE ROBOT MODEL

4 Results

4.1 Localisation

TALK ABOUT THE PERFORMANCE OF THE LOCALISATION ALGORITHM WAS THE ALGORITHM ABLE TO CONVERGE ON THE ROBOT POSE - SHOW PICTURES FROM BOTH ROBOT MODELS DID ONE CONVERGE FASTER THAN THE OTHER
ONCE LOCALISED WAS THE ROBOT ABLE TO SUCCESSFULLY NAVIGATE TO THE DESIRED LOCATION PROVIDED

4.2 Technical Comparison

5 Discussion

5.1 Topics

6 Conculsion / Future Work

6.1 Modifications for Improvement

References

[1] I. J. Cox, "Blanche: An experiment in guidance and navigation of an autonomous robot vehicle," *IEEE Trans. Robotics Automat*, vol. 7, pp. 193–204, 1991.

[2] C. M. Wang, "Location estimation and uncertainty analysis for mobile robots," *Proceedings from IEEE International Conference on Robotics and Automation*, pp. 1231–1235, 1988.

[3] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust monte carlo localisation for mobile robots," *Artificial Intelligence*, 2001.

[4] S. Thrun, D. Fox, and W. Burgard, "Markov localisation for mobile robots in dynamic environments," *Journal of Atrificial Intelligence Research*, vol. 11, pp. 391–427, 2001.

[5] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME - Journal of Basic Engineering*, vol. 82 (Series D), pp. 35–45, 1960.