5-2013

An Analysis of Simultaneous Localization and Mapping (SLAM) Algorithms

Megan R. Naminski
Macalester College, mnaminski@gmail.com

5-1-2013

An Analysis of Simultaneous Localization and Mapping (SLAM) Algorithms

Megan R. Naminski

An Analysis of Simultaneous Localization and Mapping (SLAM) Algorithms

Megan R Naminski
Advisor: Susan Fox
Macalester Math, Statistics, and Computer Science Department

May 7, 2013

Abstract

This paper provides an introduction to two Simultaneous Localization and Mapping (SLAM) algorithms: EKF SLAM and Fast-SLAM. SLAM allows an autonomous robot to accurately map an unknown environment as well as locate itself within the environment. These algorithms work iteratively, by moving about the environment and extracting and observing various landmarks in the environment. EKF SLAM and Fast-SLAM solve the SLAM problem by using probabilities to control for errors in the robot's sensors. This paper provides a discussion of these two algorithms and compares their run times and the accuracy of the maps they produce.

1 Introduction

The Simultaneous Localization and Mapping (SLAM) problem is a widely researched problem in Artificial Intelligence that asks if a robot can autonomously build an accurate map of an unknown environment. SLAM requires the robot to simultaneously build a map of the environment and locate itself within the map. As Durrant-Whyte and Bailey (2006) put, "A solution to the SLAM problem has been seen as a 'holy grail' for the mobile robotics community as it would provide the means to make a robot truly autonomous" (p. 1).

The independent problems of mapping and localization, though not trivial, are fairly straightforward. The complexity of SLAM arises from the unknown nature of both the robot's path and environment due to their interdependence. Grisetti et al. (2005) explain that "[SLAM] is considered to be a complex problem, because for localization a robot needs a consistent map and for acquiring the map the robot requires a good estimate of its location" (p. 1). Beyond the complexity caused by this interdependence are the complexities introduced by the sensors of the robot itself. The values retrieved from the robots sensors are often inaccurate in unpredictable ways. This issue increases the difficulty of obtaining both a correct location of the robot and an accurate map of the environment.

This paper provides a brief introduction to SLAM and its key concepts. It offers an outline of the underlying algorithmic pattern used for solving the SLAM problem. Finally, it introduces EKF SLAM and Fast-SLAM as solutions to the SLAM problem and analyzes these algorithms comparitively for their run-time and accuracy.

1.1 SLAM History

According to Riisgaard and Blas, the SLAM problem was developed by Hugh Durrant-Whyte and John J. Leonard, who got the idea from work done by Smith, Self and Cheeseman (Riisgaard and Blas (2003)). The origin of probabilistic SLAM, according to Durrant-Whyte and Bailey, occurred at the IEEE Robotics and Automation Conference in San Francisco 1986 (Durrant-Whyte and Bailey (2006)). Since this, probabilities have been used to limit the impact of inaccurate sensor readings on the accuracy of the resulting map.

Despite the new approach, SLAM researchers continued to have limited success in solving the problem. Researchers limited most work to the problem of either localization or mapping due to the difficulties of accurately accomplishing both at once. The process of SLAM provides measured information about the distance between the locations of landmarks used in localization known as landmark correlations. SLAM researchers generally ignored or attempted to minimize these correlations because they believed them to be excess noise that polluted their estimations. They later discovered the convergent nature of the SLAM problem and found the correlations they had previously sought to eliminate were an integral part of the solution to the SLAM problem. In fact, as Durrant-Whyte and Bailey state, "the more these correlations grew, the better the solution" (Durrant-Whyte and Bailey (2006)).

1.2 Applications of SLAM

Solutions to SLAM are not just the "holy grail" for the Artificial Intelligence community; they have important implications for various real world problems. SLAM enables robots to autonomously explore environments that are too dangerous or inaccessible to humans. SLAM has applications in the military, deep sea navigation, mine exploration, search and rescue, space exploration, and many other areas. There are numerous practical uses for SLAM as a means of gaining knowledge in areas humans cannot access.

The DARPA Grand Challenge DAPRA DRC, a prize competition for autonomous vehicles, has been a stage for multiple SLAM implementations throughout the years. This contest is issued by the Defense Advanced Research Projects Agency of the US government's Department of Defense with the goal of developing robots capable of navigating and completing tasks in potentially dangerous environments. Several teams in the past, including MIT's and Cornell's teams in 2007 have competed using SLAM implementations. Both these teams were one of the six to finish the course and MIT came in 4th.

2 Background

2.1 Probability and SLAM

Probability plays a large role in successful SLAM solutions. SLAM robots build an accurate map from inaccurate measurements of their environment. The robot has no prior knowledge of its surroundings so it must use its sensors in order to gather information. Therefore, SLAM must deal with the uncertainty of locations as a result of inaccurate sensor data. Figure 1 shows a simple example of an estimated robot path and some estimated landmark locations according to sensor data in comparison to the actual path and locations.

In Figure 1, $\mathbf{x}_k$ is the state vector of the robot at time step k which describes the robot's orientation and location. The control vector, $\mathbf{u}_k$ describes the readings from the odometer on changes in orientation and location at time step k. The landmark

Figure 1: Visualization of the estimated and true locations of the robot and landmarks at various states from Durrant-Whyte and Bailey (2006).

vector, $\mathbf{m}_i$ describes the location of landmark i, and is independent of time. The observation vector $\mathbf{z}_{ik}$ describes the observation of the ith landmark's range and bearing from the robot's sensor at time step k. The figure describes the path of a robot at various time steps. As the figure depicts, the robot's estimation of its path is inaccurate. The figure also shows the differences between the robot's estimation of the landmark locations and the actual landmark locations.

The robot's location estimates depend on two types of input. The first is the input from the robot's odometer which predicts the location of the robot based on its previous position and the odometer's readings on its most recent movement. The second type of input, readings from the vision sensor. There are a variety of different types of vision sensors from cameras to sonar sensors but the most commonly used is a laser sensor. Vision sensors provide the range and bearing of landmarks in the environment from the robot's position. These landmark estimates can be used to estimate the location of the robot. SLAM's general probability distribution, used to describe the robot's location, combines the location information from both sensors as shown.

$$P(\mathbf{x}_k, \mathbf{m} | \mathbf{z}_{0-k}, \mathbf{u}_{0-k}, \mathbf{x}_0)$$

- $\mathbf{m}$ = Set of all landmark locations (independent of time).
- $\mathbf{z}_{0-k}$ = Set of all landmark observations describing the range and bearing of landmarks visible at the corresponding state.
- $\mathbf{u}_{0-k}$ = History of control inputs describing the odometry readings on changes in orientation or location in the map.

- $\mathbf{x}_0$ = Starting position of the robot.

This equation describes the probability distribution of the joint posterior density of the vehicle's and landmarks' positions given the recorded landmark readings and control input throughout the robot's movement history and its initial position. This demonstrates how the probability of the position of the robot is dependent on the position of the landmarks, its odometer readings, and the history of its previous states.

2.2 Landmark Correlations

Landmarks are recognizable points in the environment, such as walls or table legs, used for localization. As discussed before, the correlations between landmark locations are a crucial part of successful SLAM solutions. Sensor readings are fundamentally unreliable as a means of determining the true location of landmarks based entirely on their range and bearing from the estimated position of the robot. However, from any given position, the estimation of the distance between two landmarks is significantly more reliable, due to the nature of fairly consistent errors between measurements of range and bearing from the robot. As the robot moves around and takes readings of the landmarks and their relative distances from different positions in the map, these correlations between landmarks converge. This offers a more trustworthy method of measuring landmark location given faulty sensors.

Durrant-Whyte and Bailey (2006) offers a visualization of this method as a network of springs that connect landmarks in Figure 2. The robot's movement and subsequent observations of the landmarks act like displacement to the spring system which is "dependent on the local stiffness (correlation) properties" (p. 1). The more observations on the correlations between landmarks, the stiffer the spring, which allows for an accurate relative map of the landmarks to be built over time. In Figure 2, the red lines represent the springs connecting landmarks, while their thickness represents their stiffness caused by increasing the correlations through multiple observations.

2.3 The SLAM Solution Outlined

Solutions to SLAM follow a general pattern that repeats throughout the process of building the map. This series of steps begins at the end of every movement the robot makes.

2.3.1 Odometry Readings and Location Prediction

The first step in the process is to gather the information about the robot's current position to store in the control vector. The control vector is a matrix which holds information about how different movements affect the position of the robot. This information is in the form of data read by the robot's odometer. The odometer gives information about changes in the orientation of the robot and changes in the distance travelled by the robot since the last state which can be used to determine a prediction about the location of the robot within the map and the direction it faces.

Figure 2: Visualization of the spring analogy from Durrant-Whyte and Bailey (2006).

2.3.2 Sensor Readings and Data Association

The next step is to gather readings on the visible landmarks from the robot's current position. This information gives us the range and bearing of the landmarks in relation to the robot. These values are then used in conjunction with the stored position estimates of the landmarks to estimate the position of the robot in the map through triangulation.

In the middle of this step, the robot undergoes the process of data association. This is the process by which the robot attempts to associate the landmarks currently visible with those that have already been observed from previous positions. This is an important step for both localization and mapping because if the robot cannot determine that a landmark has already been observed, it cannot use its stored position for localization. According to Arras (2003), the most common method of data association is the nearest neighbor standard filter (NNSF) which matches visible landmarks to the nearest known landmark within a certain threshold. NNSF uses Mahalanobis distance to determine the distances between landmark which takes into account the uncertainty of position measurements.

2.3.3 Location Correction

In this step, a new estimate position is determined. This is the step where the probability distribution for the location of the robot is used. Based on the two estimated locations of the robot from the odometer and the landmark triangulation, a new estimate is calculated that is a combination of the two values. The formula used to combine these two values is dependent on the solution algorithm.

2.3.4 Landmark Location Updates

Next, the estimated locations of the landmarks are updated. This is done using the range and bearing data gathered through the sensors and the estimate of the

robot's current position. Correlations between landmark locations are also gathered and updated at this step. The map of the environment is also updated at this point.

2.3.5 Add New Landmarks

The final step is to add new landmarks. In the new position of the robot, new landmarks may be observable. These landmarks are determined, and then their estimated location is added to the list of landmark locations. The correlations between new landmarks and other visible landmarks are also stored. This allows for the exploration of new parts of the environment.

2.4 Determining Landmarks

Determining landmarks is a crucial aspect of SLAM because the robot relies on using landmarks in order to determine its location. New landmarks are gathered from the environment through the process of feature extraction. Algorithms for feature extraction identify many different types of features. Two methods for extracting landmarks are spike landmark extraction and Random Sampling Consensus (RANSAC) discussed in Riisgaard and Blas (2003). These are the most commonly used methods of feature extraction for SLAM.

2.4.1 Spike Landmark Extraction

Spike Landmark Extraction uses extrema in the distance data from the laser sensor to identify landmarks. In this process, the robot analyzes the distance data across the visible area visible for significant changes in distance in order to locate areas of extreme change. The robot can single out these areas as landmarks. Using this method, it often extracts objects such as table legs and desk corners as landmarks.

Figure 3: Using spike landmarks to extract table legs (orange) for use as landmarks from the rest of the environment from Riisgaard and Blas (2003).

This method of extracting landmarks works well for areas with lots of clutter because there are many available extrema to extract as landmarks. It fails in smooth

environments where there are no significant differences in distance because the algorithm is unable to extract the extrema.

2.4.2 Random Sampling Consensus

RANSAC is the process of extracting lines formed in the environment for use as landmarks. This method is useful for extracting walls for landmarks in indoor environments. Wall landmarks are extracted using best fit lines on the distance data read from the sensors of the robot. Lines like this are used as a landmark by extracting at each position the closest point on the line to the robot as the location of the landmark for performing analysis of range and bearing and location updates.

Figure 4: Using RANSAC to extract walls as landmarks from rest of the environment from Riisgaard and Blas (2003).

RANSAC does not perform well in cluttered environments where it cannot extract best fit lines from the input data. The benefit of RANSAC, however is that it generally disregards the use of people as landmarks. Spike landmark extraction may attempt to use a person as a landmark because they generally provide a significant difference in distance data from the rest of the environment. People, and other animate objects, are not ideal for use as landmarks because they are mobile. Moving objects cause errors in the processes of building landmark correlations and data association.

3 Related Work

Many researchers have studied SLAM in the last two decades. The majority of the introductory and background information provided in this paper comes from two sources, Riisgaard and Blas (2003) and Durrant-Whyte and Bailey (2006).

Riisgaard and Blas (2003) is a beginner guide to the SLAM problem and provides a broad overview of the SLAM process. The authors provide a brief overview and history of SLAM. The paper discusses types of hardware required to solve SLAM including sonar sensors, laser sensors, and vision sensors, and the advantages and disadvantages of each. The authors list the three main parts of SLAM as: landmark extraction, data association, and the Extended Kalman Filter. The paper also contains information and analysis on the two feature extraction methods discussed in this paper and an overview of EKF SLAM that provides a basis of understanding for the later portion on EKF SLAM in this paper.

Durrant-Whyte and Bailey (2006) and also Bailey and Durrant-Whyte (2006) are two papers intended as tutorials for working with the SLAM problem. The first of these papers begins by introducing the SLAM problem and discussing its implications as a means for completely autonomous robots. The paper also goes into the history and development of the SLAM problem, describing initial assumptions about solutions and breakthroughs in the area. The authors attribute the main breakthrough in solutions to SLAM to the realization that correlations in the locations of landmarks should be used to determine the location of landmarks instead of discarded as noise. The article goes into detail about the computations involved in probabilistic SLAM, which involve using Bayes Theorem and the Markov model to calculate the probability of landmark observations and vehicle location using data from previous vehicle locations, landmark observations, control inputs, and landmark sets. EKF-SLAM and FastSLAM are both briefly described, along with the calculations and computational complexity associated with them. The tutorial also mentions several implementations of SLAM as well as sources of open source software for SLAM simulations. The tutorial provides useful information for the background of this paper, and an introduction to later sections of this paper. Overall, the tutorial is a good source of information on SLAM but is unsuccessful as an introductory SLAM paper because it fails to define the concepts used throughout the paper which are unfamiliar to beginning readers.

Part two of the tutorial, Bailey and Durrant-Whyte (2006), provides a more in depth look into the processes involved with SLAM. This paper discusses in depth the computational complexity of SLAM and ways to reduce computational complexity through partitioning the map in a variety of ways such that landmarks are only compared to other landmarks in their region. This reduces the necessary comparisons at each step in the process. The paper also talks about data association and the process of determining landmark location by observing a landmark multiple times from different locations and performing triangulation. The tutorial provides relevant information on SLAM processes for use in later sections of this paper.

The section of this paper on the Extended Kalaman Filter as a solution algorithm for SLAM is mostly based on information gathered Choset et al. (2005). Choset

et al. (2005) provides a detailed overview of the Kalaman filter. The chapter on Kalaman filtering starts off with a description of probabilistic estimation and then walks through the derivation of the Kalaman filter for linear systems. The chapter then provides details for the extended Kalaman filter which it applies to SLAM. The book gives a brief description of SLAM, and includes the math behind the equations used to solve SLAM using EKF which will be a useful in providing mathematical equations for the EKF process discussed in this paper.

The EKF SLAM and Fast-SLAM implementations from this paper are based of an implementation of EKF SLAM by Kai Arras available online, which is an implementation based of the method from the book Arras (2003). This book provides a detailed description of EKF SLAM as well as the various componenets involved in implementing SLAM. It provides an approach to EKF SLAM that eliminates the gaussian noise variables from the EKF, and instead relies only on constant error values for the sensors. In the adaptation of the code described in this papers, the gaussian noise values have been added back into the EKF to make it more comparable to the implementation of Fast-SLAM in which it is not possible to eliminate the random values due to the nature of the particle filter.

Information on Fast-SLAM provided in this paper is based on two papers by Montemerlo et al. (2002) and Grisetti et al. (2005). Montemerlo et al. (2002) provides an overview of EKF Slam as well as analysis on its limitations. The paper also discusses FastSLAM, which uses a particle filter to provide estimations on the location of the robot. The authors of this paper thoroughly compare the computational and space complexities EKF SLAM and FastSLAM, determining that FastSLAM provides a more efficient algorithm without significant losses in accuracy. They also discuss possible optimizations that can be made to the storage of information that can make these algorithms more efficient. The analysis in the paper also provides valuable information for the analysis section of this paper.

Grisetti et al. (2005) discusses the use of the Rao-Blackwellized particle filter in SLAM as a substitute for EKF SLAM. This paper analyzes the complexity of Fast-SLAM and discusses several ways to improve Fast-SLAM through an improved sampling distribution and reducing the number of particles. This paper also discusses a selective resampling strategy which reduces the number of resampling steps and improves the outcome of the algorithm. The analysis in this paper provides information used in the Fast-SLAM section of this paper.

4 Solutions to SLAM

The prominence of the SLAM problem in AI has inspired many solutions. These solutions seek to deal with the uncertainties in the sensor data in ways that allow the robot to map an environment fairly accurately. Solutions must also maintain low enough computational complexity to provide feasible implementations in real-world situations. Two such solutions to SLAM are EKF SLAM and Fast-SLAM.

4.1 EKF SLAM

EKF SLAM employs the use of the Extended Kalman Filter (EKF), a non-linear version of the Kalman Filter, to describe state changes. In EKF SLAM, noise values from a zero mean uncorrelated Gaussian distribution are used to deal with the uncertainty of the robot's sensor values used in predicting location. In other words, an element of randomness is added to the location predictions to account for random sensor error.

EKF SLAM employs a system of matrices to store information about the state, or location of the robot as well as the information about landmark locations and correlations. Operations on these matrices reveal the probabilistic location of the robot and landmarks.

4.1.1 Position Vector

The first matrix, $\mathbf{x}$, represents the estimated vector state of the system. This matrix contains the estimated location and orientation of the robot along with the estimated locations of all the landmarks that have been observed thus far. As depicted in Figure 5, this matrix is one column wide and contains a row for the x-coordinate, row for the y-coordinate, and a row for the θ, or orientation, of the robot, as well as a row for the x-coordinates and a row for the y-coordinates of all landmarks in the system.

$$\begin{bmatrix} x_r \\ y_r \\ \theta_r \\ x_1 \\ y_1 \\ \vdots \\ x_n \\ y_n \end{bmatrix}$$

Figure 5: Position matrix holding the x-coordinate, y-coordinate, and bearing of the robot along with the x and y coordinates of all landmarks.

4.1.2 Observation Vector

The second matrix, $\mathbf{z}$, represents the robot's observations of landmarks. This matrix contains the observed range and bearing of landmarks from the robot's position. This matrix is one column wide and twice as long as the number of landmarks. Each pair of rows represents the range of a landmark, or distance between the landmark and the robot, and bearing of the landmark, or the angle between the direction the robot faces and the position of the landmark. Figure 6 shows $\mathbf{z}$ where r_i and b_i are the range and bearing of landmark i.

r_1
b_1
r_2
b_2
$\vdots$
r_n
b_n

Figure 6: Observation matrix holding the range and bearing of the landmarks.

4.1.3 Covariance Matrix

The third matrix, **P** shown in Figure 7, is known as the covariance matrix. The top corner of this matrix, **A**, is the covariance on the robot's position, which essentially describes the uncertainty of the robot's position. The other cells on the diagonal (**B** through **C**) are the covariance matrices on the positions of the landmarks. Cell **D** and the rest of the cells in the first column describes the covariance between the robot and the landmark represented by each row of cells. Excluding the first row and column of cells and the cells on the diagonal, the remaining cells describe the correlations between the positions of different landmarks. In this way, **F** describes the covariance between the position of the first and last landmark. The covariance encompasses the landmark correlations discussed earlier. Throughout the matrix, the cells on either side of the diagonal are symmetrical such that **E** is the transpose of **D** and **G** is the transpose of **F**. **P** represents the covariances and correlations calculated over all previous time steps and is updated at each step. The calculations for these updates are discussed later.

Figure 7: Covariance matrix **P** from Riisgaard and Blas (2003).

4.1.4 Prediction Model Jacobian

The Jacobian of the prediction model, **J**, is used to predict the new location of the robot after movement through data collected from the odometer. This matrix predicts the effect that changes in the robot's rotation and distance will have on its

position and bearing. Figure 8 shows **J** where Δy and Δx describe the robot's change in x and y coordinates, respectfully.

1	0	-Δy
0	1	Δx
0	0	1

Figure 8: Jacobian matrix **J** describing effect of robot's movement on its position.

4.1.5 Measurement Model Jacobian

The Jacobian of the measurement model, **H**, is a stacked matrix that describes the change in range and bearing for each landmark with respect to the robot's predicted state and the landmarks' positions. This can then be combined with the sensor's values using the Kalman gain matrix discussed below to get a more accurate estimation of range and bearing. This, in turn, is used in determining the landmark's location and correlations with other landmarks and the robot. Figure 9 Shows the construction of **H**, where r_i is the range value for landmark i which is the distance between this landmark and the robot.

$\frac{x_r - x_1}{r_1}$	$\frac{y_r - y_1}{r_1}$	0	$\frac{x_1 - x_r}{r_1}$	$\frac{y_1 - y_r}{r_1}$	0	0	$\cdots$	0	0
$\frac{y_1 - y_r}{r_1^2}$	$\frac{x_r - x_1}{r_1^2}$	-1	$\frac{y_r - y_1}{r_1^2}$	$\frac{x_1 - x_r}{r_1^2}$	0	0	$\cdots$	0	0
$\frac{x_r - x_2}{r_2}$	$\frac{y_r - y_2}{r_2}$	0	0	0	$\frac{x_2 - x_r}{r_2}$	$\frac{y_2 - y_r}{r_2}$	$\cdots$	0	0
$\frac{y_2 - y_r}{r_2^2}$	$\frac{x_r - x_2}{r_2^2}$	-1	0	0	$\frac{y_r - y_2}{r_2^2}$	$\frac{x_2 - x_r}{r_2^2}$	$\cdots$	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$\frac{x_r - x_n}{r_n}$	$\frac{y_r - y_n}{r_n}$	0	0	0	0	0	$\cdots$	$\frac{x_n - x_r}{r_n}$	$\frac{y_n - y_r}{r_n}$
$\frac{y_n - y_r}{r_n^2}$	$\frac{x_r - x_n}{r_n^2}$	-1	0	0	0	0	$\cdots$	$\frac{y_r - y_n}{r_n^2}$	$\frac{x_n - x_r}{r_n^2}$

Figure 9: Jacobian matrix **H** describing effect of robot's position on the range and bearing of landmarks.

In Figure 9, there are two rows for each landmark. The first of these rows corresponds to the landmark's range and the second to its bearing from the robot's position. The first three elements in the row describe effect of the changes in the robot's x, y, and θ positions, respectively, on the landmark's range or bearing. We then add two columns for each landmark along the diagonal of the remaining matrix which are the first two columns of the row negated. The third column is not repeated because landmarks have no orientation.

4.1.6 Gaussian Uncertainty Matrices

The final set of matrices involved with EKF SLAM incorporate zero mean uncorrelated Gaussian distributions into the location estimates. This serves to create Gaussian noise proportional to the uncertainty of the values used in computing state changes as the robot moves around the environment. There are two types of noise in the system, the errors in odometer readings and the errors in the range/bearing sensors. Therefore, EKF involves two matrices intended to incorporate the uncertainty of both these readings into the localization and mapping procedure. These matrices operate on the values received by the sensor to produce an estimation of accuracy for the values retrieved from the sensors.

The first of these matrices is **Q** in Figure 10, where c is a zero-mean Gaussian which is calculated seperately for each entry. This describes the error in the odometry sensor on the robot.

$c\Delta x^2$	0	0
0	$c\Delta y^2$	0
0	0	$c\Delta\theta^2$

Figure 10: Uncertainty matrix **Q** describing the random error in the odometer.

The second of these matrices is **W** in autoreffig:W, where c and d are zero-mean Gaussians each calculated seperately, r is the range error value, and b is the error in bearing. These error values are specific to the robot system and are hard-coded estimations of errors based on previously observed readings from the robot. This matrix describes the error in the vision sensor's observations about landmarks. **W** is a stacked vector with the rc and bd for each landmark down its diagonal.

rc_1	0	0	0	$\cdots$	0	0
0	bd_1	0	0	$\cdots$	0	0
0	0	rc_2	0	$\cdots$	0	0
0	0	0	bd_2	$\cdots$	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
0	0	0	0	$\cdots$	rc_n	0
0	0	0	0	$\cdots$	0	bd_n

Figure 11: Uncertainty matrix **W** describing random error in the vision sensor.

4.1.7 Location Prediction

In the prediction step of this algorithm the predictions about the new position and covariance matrix are made. First, the prediction about the state vector **x** at time step $k+1$ is made.

$$\begin{aligned}\mathbf{x}(k+1|k) &= f(\mathbf{x}(k|k), \mathbf{u}(k+1)) \\ &= \mathbf{x}(k|k) + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \begin{bmatrix} \Delta x * q \\ \Delta y * q \\ \Delta\theta * q \\ 0 \\ \vdots \\ 0 \end{bmatrix}\end{aligned}$$

In this, $\mathbf{x}(k|k)$ describes the state vector matrix at time step k, $\mathbf{u}(k+1)$ describes the information from the odometry sensor at time step $k+1$, and q is the error term describing the error in the odometer.

Next, the robot's covariance matrix cell $\mathbf{A}$, the top corner of the covariance matrix $\mathbf{P}$, is estimated.

$$\mathbf{A}(k+1|k) = \mathbf{J}\mathbf{A}(k|k)\mathbf{J}^T + \mathbf{Q}$$

We also update the robot to feature correlations by taking the remaining cells in the first column, such as cell $\mathbf{D}$, and operating on them similarly like we did with $\mathbf{A}$. We also update the cells along the first row so that $\mathbf{E}$ remains $\mathbf{D}^T$ after the operation.

$$\mathbf{D}(k+1|k) = \mathbf{J}\mathbf{D}(k|k)$$

4.1.8 Location Correction and the EKF

EKF SLAM employs the Extended Kalaman Filter, which performs operations on the specified matrices in order to correct the position vector according to observed landmarks. In this step, the robot observes the landmarks from its new position, giving us $\mathbf{z}(k+1)$. From this, we construct $\mathbf{H}$ and $\mathbf{W}$, which we use to calculate the innovation covariance, $\mathbf{S}$

$$\mathbf{S} = \mathbf{H}\mathbf{P}(k+1|k)\mathbf{H}^T + \mathbf{W}$$

$\mathbf{S}$ is then used to calculate the Kalman gain matrix, $\mathbf{K}$.

$$\mathbf{K} = \mathbf{P}(k+1|k)\mathbf{H}\mathbf{S}^{-1}$$

The Kalman gain matrix is a calculation of the uncertainty in the sensor used in measuring the range and bearing of the robot. The sensor's accuracy affects the gain in accuracy achieved from incorporating the sensor's readings into the robot's estimated location (and estimated landmark locations). The Kalaman gain describes this gain in accuracy separated into the gain from the range values and the gain from the bearing values. This matrix is used in combining values from the odometer and sensor effectively to produce more accurate estimations of location.

The stacked vector ν describes the difference between the predicted and observed range and bearing from the robot to the different landmarks.

$$\nu = \mathbf{z}(k+1) - \mathbf{H}\mathbf{x}(k+1|k)$$

We can use ν to calculate the updated state vector based on landmark observations.

$$\mathbf{x}(k+1|k+1) = \mathbf{x}(k+1|k) + \mathbf{K}\nu$$

Finally, we update the covariance matrix $\mathbf{P}$.

$$\mathbf{P}(k+1|k+1) = \mathbf{P}(k+1|k) - \mathbf{KHP}(k+1|k)$$

4.1.9 Complexity

The complexity of EKF SLAM comes from the size of the matrices used for the process of localization and mapping. The dimensions of these matrices depend largely on the number of landmarks observed in the environment. There are several methods discussed in Bailey and Durrant-Whyte (2006) for limiting the computational complexity. These methods are intended to reduce the number of computations required at each step by limiting the number of landmarks used after each state change for updating localization and mapping quantities. Overall, however, the computation complexity of EKF SLAM is directly proportional to the number of landmarks used because the locations of landmarks and their covariance matrix must be updated at each step. This leads to EKF SLAM having quadratic complexity with the number of landmarks.

4.2 Fast-SLAM

Fast-SLAM solves SLAM using the Rao-Blackwellized Particle Filter. Where EKF SLAM employs the use of Gaussian distributions to deal with the uncertainty of sensor values, Fast-SLAM uses the particle filter. The use of the particle filter eliminates the need for the landmark correlations and, consequently, the covariance matrix which decreases the computational complexity involved with solving SLAM.

4.2.1 Rao-Blackwellized Particle Filter

Particle filters are generally based off the Monte Carlo method for estimating values. The Monte Carlo method consists of taking a random sampling of an area to determine the probabilities associated with the area. In particle filtering, each particle represents an instance of the problem at a specific state. Each particle in Fast-SLAM represents an instance of the robot and has its own state vector and covariance matix such that the locations of the robot and landmarks differ between particles. For SLAM, the probability that a given particle correctly represents the true state of the robot is determined by how well the sensors' values correspond with the predicted sensor values at each particle's location. The Rao-Blackwellized Particle Filter allows solving SLAM to be split into the individual problems of localization and mapping.

4.2.2 Localization

In Fast-SLAM, localization is entirely dependent on the particle filter. Each particle receives the odometer readings and estimates its robot's position in the same manner that the location is initially predicted in EKF SLAM using these readings. For Fast-SLAM, there is no location correction step for the robot. Instead, it is assumed that the estimated position of the robot for each particle is the true location. This is allowed because each particle will estimate a different location and through resampling only the particles with positions that are most likely according to range and bearing readings from the landmarks will be resampled. This means that particles with less accurate position estimations will die off.

4.2.3 Mapping

Like EKF-SLAM, Fast-SLAM employs Extended Kalman filters to estimate the locations of landmarks. The estimations of landmark location depend on the location of the robot, so each particle requires its own landmark location estimation. Fast-SLAM's approach to landmark mapping is similar to EKF SLAM, except that instead of using one Extended Kalman Filter, Fast-SLAM uses a separate Extended Kalman Filter for each of its landmarks. In this way, Fast-SLAM splits up the matrices, making each Extended Kalman Filter process faster because of the lower dimensions on the matrices.

4.2.4 Particle Resampling

Since Fast-SLAM does not keep track of landmark correlations, which are the key to accuracy for EKF-SLAM, Fast-SLAM must rely on an intelligent method of resampling particles to choose particles which contain the most accurate robot and landmark positions. Different algorithms implement resampling at different points in the algorithm. The most common method is to resample the particles between every movement that the robot makes. Grisetti et al. (2006) suggest a method of adaptive resampling to reduce time complexity by reducing the number of times the particles are resampled. After each step, this method determines whether or not resampling is necessary based on the differences between particles.

The resampling of particles depends largely on the weight assigned to each particle. Particles with greater weights are more likely to be resampled than those with lower weights. The weight corresponds to the probability that the particle's state vector $\mathbf{x}$ is correct given the observation vector $\mathbf{z}$. Weights are therefore a calculation of probability such that that the new sample is drawn from an approximation of the true distribution, $P(\mathbf{x_0}|\mathbf{X}_{0:k}, \mathbf{Z}_{0:k}, \mathbf{u}_k)$, in the form of the proposal distribution, $\pi(\mathbf{x}_k|\mathbf{X}_{0:k-1}, \mathbf{Z}_{0:k}, \mathbf{u}_k)$, where, as before, $\mathbf{x}$, $\mathbf{z}$, $\mathbf{u}$, and t are the position vector, observation vector, control vector, and time step, respectively. This gives us the weight $w_k^{(i)}$ for the ith particle at time k.

$$w_k^{(i)} = w_{k-1}^{(i)} \frac{P(\mathbf{z}_k|\mathbf{X}_{0:k}^{(i)}, \mathbf{Z}_{0:k-1}) P(\mathbf{x}_k^{(i)}|\mathbf{x}_{k-1}^{(i)}, \mathbf{u}_k)}{\pi(\mathbf{x}_k^{(i)}|\mathbf{X}_{0:k-1}^{(i)}, \mathbf{Z}_{0:k}, \mathbf{u}_k)}$$

4.2.5 Complexity

Since the covariance matrix is eliminated from Fast-SLAM by the separation of the problem into localization and mapping and the inclusion of the particle filter, the dimensions of the matrices used in Fast-SLAM's Extended Kalman Filter are significantly lower than those used in EKF SLAM and a fixed dimension regardless of the number of landmarks. Each particle performs it's own Extended Kalman Filter on each of its landmarks, which creates the complexity of $O(NK)$, where N is the number of particles and K is the number of landmarks. Montemerlo et al. (2002) suggests a optimization in data structures using a binary tree that allows the Fast-SLAM to be reduced to $O(N \log K)$ time.

4.3 Discussion

For both EKF SLAM and Fast-SLAM, the run time is affected by the number of landmarks extracted from the environment. Fast-SLAM offers a more scalable solution to SLAM than EKF SLAM because is able to incorporate larger numbers of landmarks with lesser penalties to the run time of the algorithm than EKF SLAM. However, the accuracy of Fast-SLAM is additionally dependent on the number of particles used. This number also increases the computational complexity of the algorithm leading to a trade-off between accuracy and run time for the algorithm that does not occur in EKF SLAM.

Figure 12: Accuracy of Fast-SLAM vs. number of particles used in algorithm from Montemerlo et al. (2002).

Fast-SLAM also offers a more resilient algorithm. It is less likely to encounter catastrophic failure as a result of false data associations due to the fact that different particles make different associations. It is therefore more likely to recover from these

errors than EKF SLAM, which would depend on this false information for future estimations.

5 Implementation

The code from these Matlab implementations of Fast-SLAM and EKF SLAM originates from the CAS Robot Navigation Toolbox implementation of EKF SLAM, detailed in Arras (2003). The original implementation of this code eliminated the need to add Gaussian noise values, favoring instead an estimated error propagation. This method is intended to favor consistency, which lends itself nicely to a SLAM simulation.

5.1 Changes to EKF SLAM

Most SLAM algorithms assume that odometry data is received in the form of a rotation value and a translation value which tells how much the robot has turned and how far it has moved in the direction it faces. In the Cas-EKF SLAM simulation, the robot receives odometry data in a different form. Instead of rotation and translation, the odometer records how far the right and left wheel have travelled. This leaves us with the following control vector $\mathbf{u}$ where s_r and s_l are the distances travelled by the right and left wheel, respectively.

$$\mathbf{u}(k+1) = \begin{bmatrix} s_r \\ s_l \end{bmatrix}$$

This can be translated to determine the new position and orientation of the robot by changing the prediction equation for $\mathbf{x}(k+1|k)$ to the following where b is the distance between the two wheels.

$$\begin{aligned} \mathbf{x}(k+1|k) &= \mathbf{x}(k|k) + \begin{bmatrix} \Delta x * \cos\left(\theta(k) + \Delta\theta/2\right) \\ \Delta x * \sin\left(\theta(k) + \Delta\theta/2\right) \\ \Delta\theta \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\ &= \mathbf{x}(k|k) + \begin{bmatrix} \frac{s_r + s_l}{2} * \cos\left(\theta(k) + (s_r - s_l)/2b\right) \\ \frac{s_r + s_l}{2} * \sin\left(\theta(k) + (s_r - s_l)/2b\right) \\ \frac{s_r - s_l}{b} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \end{aligned}$$

Figure 13 depicts the robot's movement model and the involved measurements for determining $\mathbf{x}(k|k+1)$. For furthur explanation see Arras (2003).

Figure 13: Kinematic model of a differential drive robot with approximation $s \approx \Delta x$ from Arras (2003).

Additionally, to account for the errors in the distance measurements for this movement model, the covariance matrix $\mathbf{U}$ is introduced where k_R and k_L are error constants.

$k_R\|s_r\|$	0
0	$k_L\|s_l\|$

Figure 14: Error covariance matrix $\mathbf{U}$ describing the odometer error.

To account for error as a result of this movement model, a new Jacobian $\mathbf{G}$ is added.

$\frac{1}{2}\cos\left(\theta + \frac{s_r - s_l}{2b}\right) - \frac{s_r + s_l}{4b}\sin\left(\theta + \frac{s_r - s_l}{2b}\right)$	$\frac{1}{2}\cos\left(\theta + \frac{s_r - s_l}{2b}\right) + \frac{s_r + s_l}{4b}\sin\left(\theta + \frac{s_r - s_l}{2b}\right)$
$\frac{1}{2}\sin\left(\theta + \frac{s_r - s_l}{2b}\right) + \frac{s_r + s_l}{4b}\cos\left(\theta + \frac{s_r - s_l}{2b}\right)$	$\frac{1}{2}\sin\left(\theta + \frac{s_r - s_l}{2b}\right) - \frac{s_r + s_l}{4b}\cos\left(\theta + \frac{s_r - s_l}{2b}\right)$
$\frac{1}{b}$	$-\frac{1}{b}$

Figure 15: Jacobian matrix $\mathbf{G}$ describing effect of the error on the robot's position.

The Jacobian $\mathbf{J}$ must also be translated to following to receive the new odometer values.

1	0	$-\frac{s_r+s_l}{2} * \sin\left(\theta + \frac{s_r-s_l}{2b}\right)$
0	1	$\frac{s_r+s_l}{2} * \cos\left(\theta + \frac{s_r-s_l}{2b}\right)$
0	0	1

Figure 16: Revised version of Jacobian **J** for differential drive robot.

With these values, the prediction step of the upper cell **A** of the covariance matrix **P** is changed to reflect the new motion model.

$$\mathbf{A}(k+1|k) = \mathbf{J}\mathbf{A}(k|k)\mathbf{J}^T + \mathbf{G}\mathbf{U}(k+1)\mathbf{G}^T$$

When eliminating the Gaussian error, the matrix **W** is replaced with **R**, which is the stacked observation covariance matrix. This changes the equation for **S**.

$$\mathbf{S} = \mathbf{H}\mathbf{P}(k+1|k)\mathbf{H}^T + \mathbf{R}$$

5.2 EKF Additions

In order to make the given EKF SLAM algorithm directly comparable to Fast-SLAM a few changes needed to be made to this code. Since Fast-SLAM relies on the Gaussian noise in order to create particles with different location predictions, an element of Gaussian noise needed to be added into the EKF implementation.

First, the Gaussian error value q was added back into the **x** prediction equation.

$$\mathbf{x}(k+1|k) = \mathbf{x}(k|k) + \begin{bmatrix} \frac{s_r+s_l}{2} * \cos\left(\theta(k) + \Delta\theta/2\right) \\ \frac{s_r+s_l}{2} * \sin\left(\theta(k) + \Delta\theta/2\right) \\ \frac{s_r-s_l}{b} \\ 0 \\ \vdots \\ 0 \end{bmatrix} + \begin{bmatrix} \frac{s_r+s_l}{2} * \cos\left(\theta(k) + \Delta\theta/2\right) * q \\ \frac{s_r+s_l}{2} * \sin\left(\theta(k) + \Delta\theta/2\right) * q \\ \frac{s_r-s_l}{b} * q \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Then, the Gaussian noise matrix **Q** was added back into the prediction equation for **A**.

$$\mathbf{A}(k+1|k) = \mathbf{J}\mathbf{A}(k|k)\mathbf{J}^T + \mathbf{G}\mathbf{U}(k+1)\mathbf{G}^T + \mathbf{Q}$$

Finally, Gaussian noise was added into the observation vector **z** to simulate vision sensor error and integrate noise values into the observation covariance matrix **R**.

5.3 Implementing Fast-SLAM

To implement Fast-SLAM, the particle filter had to be added. This was done using a loop over the prediction and update steps such that each particle performed an

individual prediction and update based off the sensor lines read in from the simulation's file. The noise values used were unique to each particle to simulate the separate readings of the different particles and to allow for a distribution of estimated positions amongst the particles.

Next, the covariance matrix **P** was eliminated from the code. Instead, individual two by two covariance matrices for each of the landmarks were stored. The stacked matrices, such as **H** were separated into parts for each of the landmarks. Then a loop was added around the Extended Kalman algorithm in the code to allow for an individual Extended Kalman Filter to be performed on each landmark for every particle.

Particles were resampled according to weight after every step. The code for weighted resampling was provided by Kaplan (1999).

6 Results

The EKF SLAM and Fast-SLAM implementations were run several times. Several trials were run to determine the time it took for a single step of the SLAM algorithm to run for different numbers of landmarks. Additionally, the entire two hundred and eighty three steps of the algorithm were run and timed all the way through several times for EKF SLAM and Fast-SLAM with several different numbers of particles. The resulting maps for the full-run trials were also saved for comparison.

6.1 EKF SLAM

Each step of the EKF SLAM algorithm includes reading in the simulation data and data association along with the localization and mapping computations previously discussed. The section of code that has the highest order of time complexity is the Extended Kalman Filter which requires matrix multiplication for matrices of large dimensions. I recorded several runs of this section of code for different numbers of landmarks in order to analyze this relationship between the number of landmarks and the time this section of code took to run. Figure 17 shows the different timings for these runs. The function of time per landmarks squared is nearly linear due to the relationship between the squared number of landmarks and the dimensions of the matrices used for the Extended Kalman Filter.

The largest number of landmarks in the simulated map, 20, takes an average time of about 0.0126 seconds. The calculated relationship between the number of landmarks and the time it takes to perform the correction step using the Extended Kalman Filter indicates that an area of 1000 landmarks would take roughly 0.832 seconds. Since this is only one part of one step of an algorithm that would take several hundred steps to produce an accurate map of the environment, this is not a desirable scaling factor.

Next, I timed the entire step of the EKF SLAM algorithm for steps with 20 landmarks. At 20 landmarks, a single step of the EKF SLAM algorithm ran in about 0.137 seconds, which is approximately 10 times the average time it took to perform the

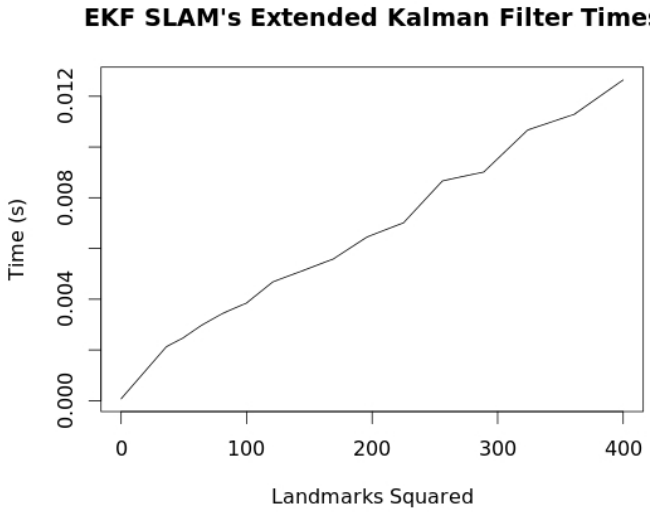


Figure 17: Plot of time for Extended Kalman Filter over the number of landmarks squared for EKF SLAM.

Extended Kalman Filter on the same number of landmarks. Reading in the data for each step of the simulation accounts for the majority of each step and takes an average time of 0.0811 seconds. While, in this case, reading in the data is simply reading in numbers, in a real life simulation it would take time to collect data from the robot's sensors and extract the useful data for landmarks. Therefore, it is reasonable that this section of the algorithm would take significantly more time.

Finally, I ran the EKF SLAM algorithm through the entire simulation of 283 steps several times to calculate the average runtime of the algorithm and average final placements of the landmarks and robot.. On average, it took 33.399 seconds to run. The average final positions of the robot and landmarks are recorded in Appendix A. Figure 18 depicts the map resulting from the average positions of the robot and landmarks.

Next, I looked at the accuracy of my EKF SLAM implementation. The code provided did not include a set of true landmark locations, so I could not compare the results of EKF SLAM to the true positions. Instead, I used the map produced by the Cas-EKF implementation from CAS Robot Navigation Toolbox which produces the same map for every run since it does not include any error values. The Cas-EKF positions are recorded in Appendix G. To get the resulting position of my EKF SLAM algorithm, I ran it several times and calculated an average. With the final positions of both implementations, I produced and error vector which described the differences between the two maps. This error vector normalized to a 2-norm value of 0.445 meters.

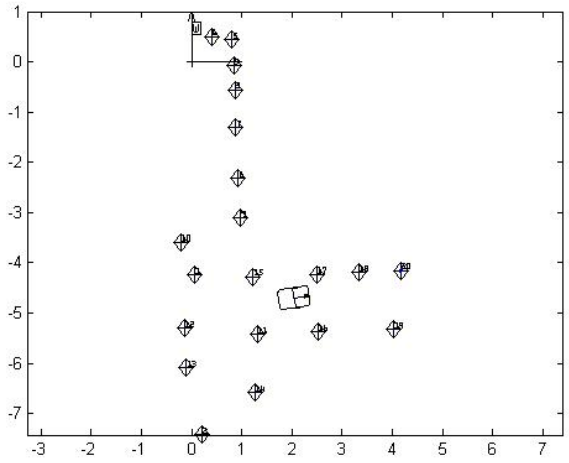


Figure 18: Average positions for EKF SLAM.

To visualize this difference, I made a figure of the algorithm's results displayed in Figure 19. In this figure, the results of several runs of EKF are displayed in grey, while the Cas-EKF results are in black.

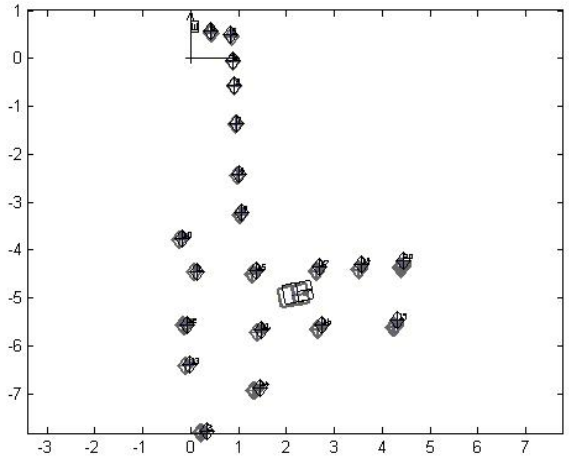


Figure 19: EKF SLAM positions vs. CAS-EKF SLAM positions.

6.2 Fast-SLAM

To compare the time it took for Fast-SLAM to perform the Extended Kalman Filter to the times from EKF SLAM, I timed Fast-SLAM's method for several different numbers of landmarks. Figure 20 shows the average timings for these runs. The relationship between timing and the number of landmarks is roughly linear. For 20 landmarks, the Extended Kalman Filter section of the code took an average of 0.00250 seconds. This is approximately $\frac{1}{5}$ the time it took to perform the Extended Kalman

Filter for EKF SLAM. However, Fast-SLAM requires several runs of the Extended Kalman Filter per step. This number of times depends directly on the number of particles used.

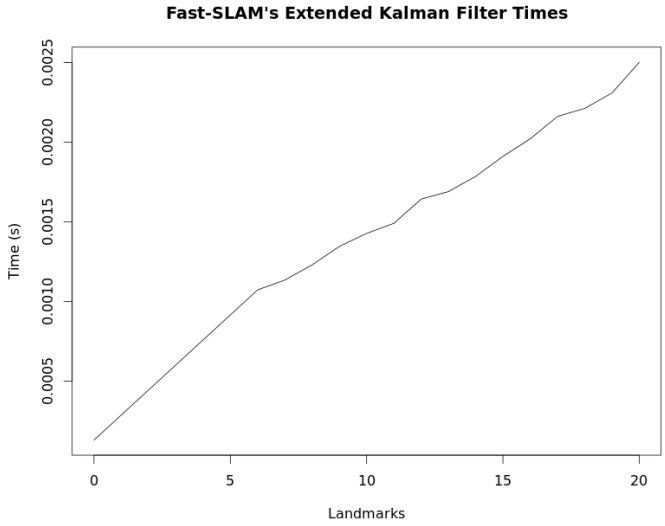


Figure 20: Plot of time for Extended Kalman Filter over the number of landmarks for Fast-SLAM.

Next, I timed the entire step of Fast-SLAM for several different numbers of particles for steps with 20 landmarks. Reading in the data from the file took approximately the same amount of time across all numbers of particles as well as the EKF SLAM algorithm. This is expected since the read step was the same amount of code for both algorithms and only occurs once per step. For one particle, a single step ran in about 0.103 seconds. For five, ten, fifteen, and twenty particles, the average steps took about 0.200, 0.324, 0.445, and 0.560 seconds, respectively. Figure 21 visualizes these results. The relationship between the times and the number of particles is almost perfectly linear.

I found a similar relationship when I timed the entire Fast-SLAM functions for the entire 283 steps several times as shown in Figure 22. A single particle ran this algorithm in about 26.754 seconds. The five, ten, fifteen, and twenty particle runs took and average of 51.841, 81.760, 111.337, and 141.664 seconds, respectively.

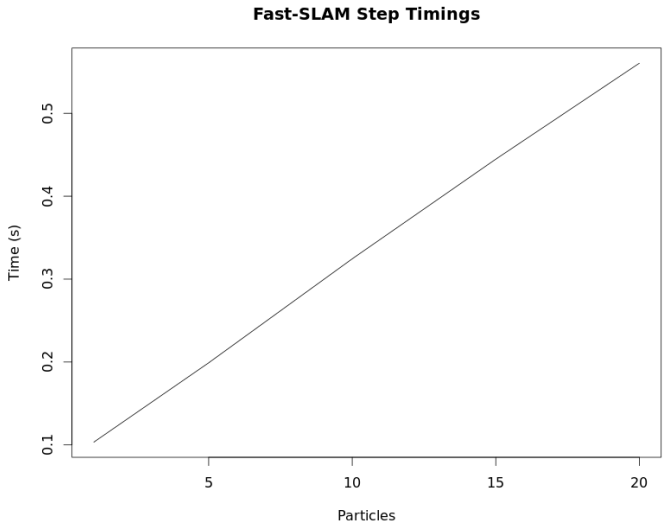


Figure 21: Plot of time for Extended Kalman Filter over the number of landmarks for Fast-SLAM.

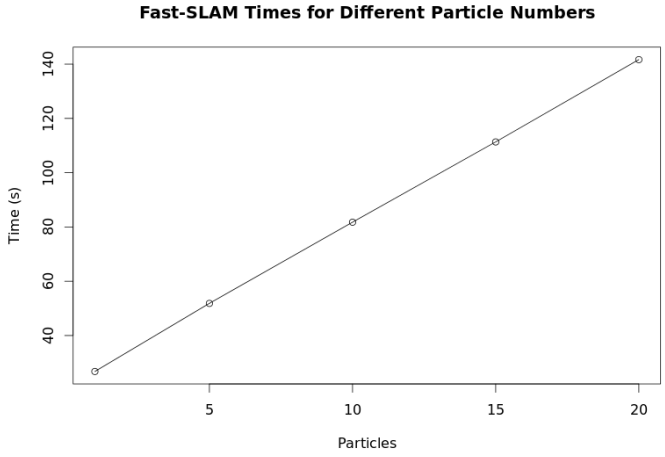
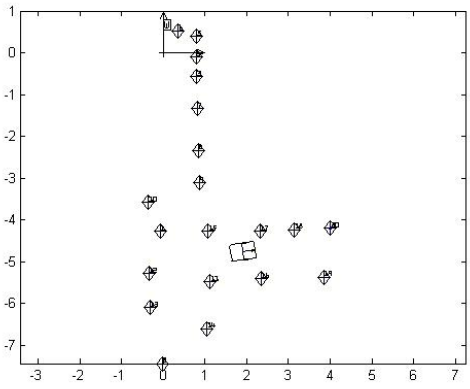
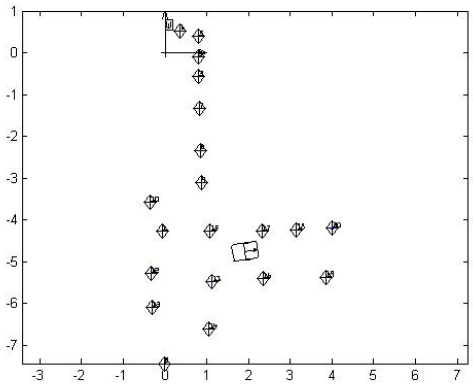


Figure 22: Plot of time for Extended Kalman Filter over the number of landmarks for Fast-SLAM.

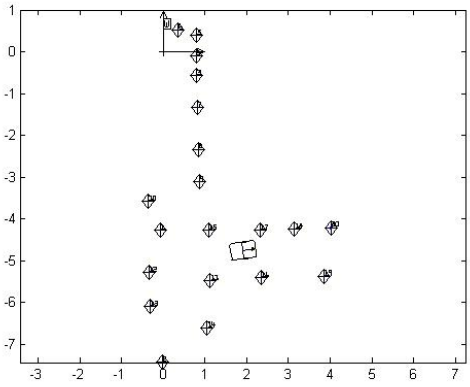
I recorded the final positions for several runs at each of the specified number of particles. The averages of these runs are stored in Appendix B - Appendix F. To visualize these positions, I plotted each average in Figure 23. I also displayed in Figure 24 the results of several runs with each number of particles in gray against the Cas-EKF results in black.



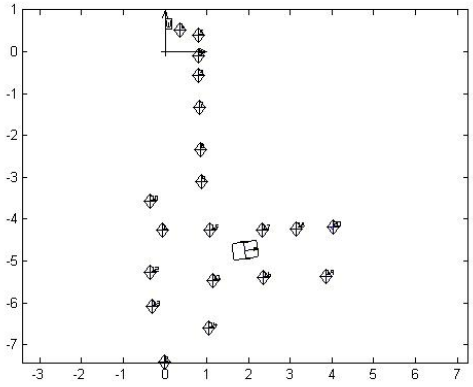
(a) 1 Particle



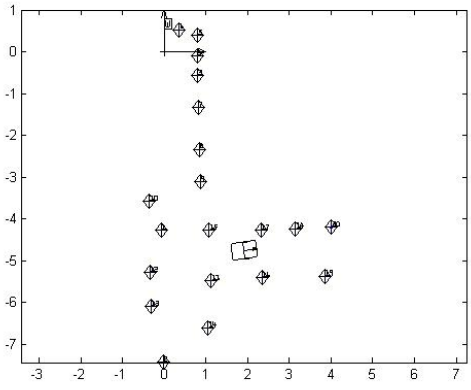
(b) 5 Particles



(c) 10 Particles

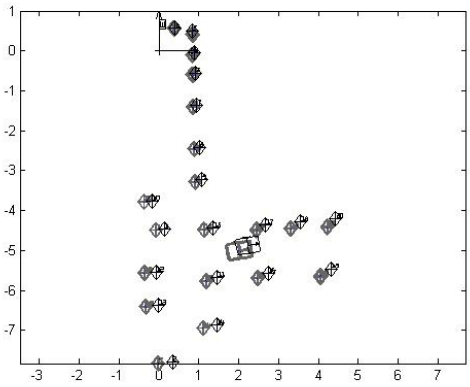


(d) 15 Particles

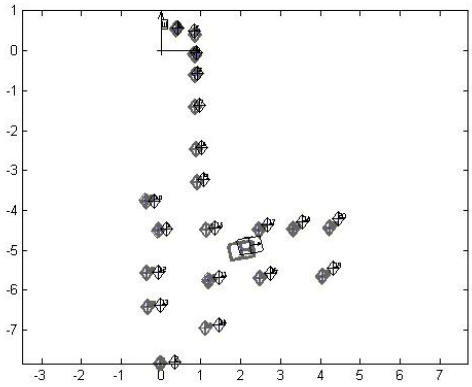


(e) 20 Particles

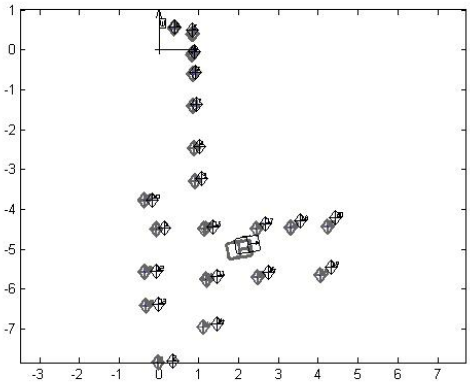
Figure 23: Average positions from Fast-SLAM



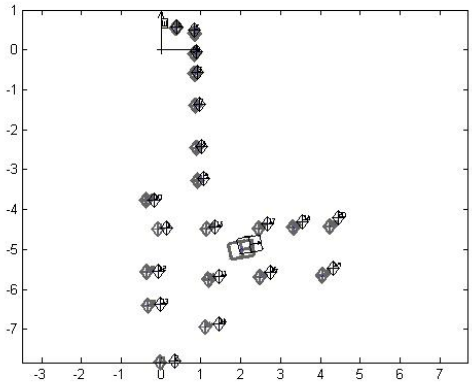
(a) 1 Particle



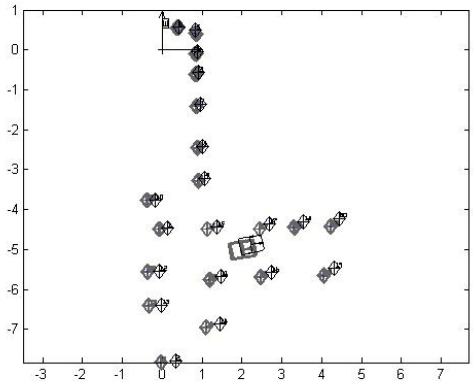
(b) 5 Particles



(c) 10 Particles



(d) 15 Particles



(e) 20 Particles

Figure 24: Fast-SLAM positions vs. CAS-EKF SLAM positions.

6.3 Analysis

6.3.1 EKF SLAM

The timings for the Extended Kalman Filter of the EKF SLAM algorithm behaved fairly predictable and scaled with the number of landmarks. This verifies the time dependence on the number of matrices present in the map. Surprisingly, the Extended Kalman Filter section of code was not the major time consuming section in each step of code. Reading in the data from the file took significantly more time across all numbers of landmarks. Also, matching observed landmarks to those already stored in the map took a comparable amount of time to the Extended Kalman Filter section, and was similarly dependent on the number of landmarks, but also the number of observed landmarks at the robot's current position.

I had expected the Extended Kalman Filter section of code to be the main time-consuming section of code, and was surprised by these results. It is possible that this is a result of using Matlab as the coding medium for this project. Matlab's matrix multiplication functions are heavily optimized for time as a result of the frequent use of Matlab for matrix operations since it is more commonly used for advanced mathematics than for other areas of programming. This would affect the ratio of time spent in the Extended Kalman Filter section compared to other sections that might rely on loops to perform operations across multi-dimensional arrays.

The section that took the most time for both EKF SLAM and Fast-SLAM was reading in the data. Presumably, for SLAM using a real robot where reading in the data does not mean opening a file and reading in distance measurements in the form of numbers, this step could be even more time consuming. Also, the method for extracting beacons would become a larger portion of the time spent because it would require some form of image processing.

With those things in mind, I was surprised that an algorithm such as Fast-SLAM is considered an optimization on EKF SLAM. In Fast-SLAM, hefty steps such as data association are performed once for every particle during the step. I would have to observe timings on a much larger map before I was convinced that getting rid of the large covariance matrix was worth the extra time spent repeating the process of information intake, processing, prediction, and correction for every particle.

6.3.2 Fast-SLAM

For Fast-SLAM, the relationship between the timings for different particles behaved exactly as expected. The number of particles directly affected the timings for both the Extended Kalman Filter section of the code, each step of the algorithm, and the entire run. For the recorded step and run timings, the data fit a linear relationship almost perfectly.

Fast-SLAM surpassed EKF SLAM in overall run-time somewhere between one and five particles. This low number is perhaps due to the Matlab optimizations on matrix operations discussed earlier, but it could also be due to the low number of landmarks available in the simulation. In a larger environment, presumably the covariance matrix of EKF SLAM would become an operational burden due to its

high dimension. Also, it would likely consume large amounts of memory due to its dimensions. Although Fast-SLAM must keep track of a map for each of its particles, it uses up about $\frac{1}{K}$ for the storage of its covariance matrices for a map with K landmarks because it does not keep track of the correlations.

The part of Fast-SLAM I found most surprising was that I did not seem to produce more accurate maps by adding particles. As shown in Figure 24 and in the results data, the resulting positions of the robot and landmarks vary more so than in EKF SLAM, but to not appear to change either in value or variance much with each number of particles used.

This result was contrary to the current literature available for Fast-SLAM. It makes sense that this would be a direct result of using small error values. To test this theory, I increased the Gaussian error varriance for several runs, but received unconvincing results. Over a certain threshold, the map produced by Fast-SLAM would become incorrect enough to interfere with the data association process and result in false negative matches. When this happened, the particle would create a new landmark very close to the landmark it should have been mapped at, which would remain in its map for the rest of the process. The particles' weights were based off of how similar the predicted and stored values were for the landmarks that had been observed and matched successfully. Therefore, the false negative matching did not interfere with the particle's weighting process. Increasing the number of particles did not stop this error from occurring.

Since this algorithm is adapted from the Cas-EKF SLAM algorithm, which does not even include the Gaussian errors necessary to do Fast-SLAM, the code used was not easily adapted to Fast-SLAM. I believe that the matching process and method for calculating landmark uncertainty would have to be drastically changed to accommodate Fast-SLAM. I attribute the relatively accurate maps produced by my Fast-SLAM implementation to relatively accurate sensor information and small amounts of error provided by the simulation. In a real environment, I do not believe the Fast-SLAM implementations would have been this accurate.

Also missing from the Cas-EKF SLAM implementation was a method for loop closing, which allows large portions of a map to be rotated and shifted to match observed and stored landmarks when the robot encounters a loop. Such a process might fix the false negative matching error discussed previously, as often the robot would produce an identical line of landmarks at an offset angle in the map.

7 Conclusion

After implementing both algorithms, I believe that EKF SLAM is a more reliable method of producing an accurate map. Despite the lack of success of my Fast-SLAM implementation, it is clear that the accuracy of Fast-SLAM relies a great deal on random values. The position of the robot in different particles of Fast-SLAM is entirely dependent on random noise. While the weighting does make particles that match up well with the observed landmarks more likely to be selected for resampling, there is no guarantee that the particle with the highest weight will even be resampled,

since it is done randomly. Fast-SLAM does not rely on the large covariance matrix that EKF SLAM needs, but in eliminating this matrix, it gets rid of data about landmark correlations that greatly enhance the accuracy of EKF SLAM.

Beyond the accuracy of Fast-SLAM, the resulting timings of the algorithm were not as optimal as anticipated. For the small simulated map, the time spent doing the actual Extended Kalman Filter method was not a significant enough portion of the step for the overall algorithm to be optimized by switching from EKF SLAM to Fast-SLAM. For small areas, EKF SLAM is clearly the more accurate and faster algorithm to use.

A EKF Final Positions

robot:

x-coord	2.150168±0.01983468
y-coord	-4.947736±0.01185052
θ	0.1538896±0.006545948

landmarks:

1

x-coord	0.0658906±0.01553505
y-coord	-4.462271±0.00430695

2

x-coord	0.219591±0.02524493
y-coord	-7.818379±0.00454633

3

x-coord	0.9072855±0.005185201
y-coord	-0.5823609±0.004563073

4

x-coord	0.8832802±0.004606466
y-coord	-0.0665038±0.004717049

5

x-coord	0.8489899±0.00663825
y-coord	0.4589987±0.00381322

6

x-coord	0.4223314±0.007925283
y-coord	0.5317487±0.004543447

7

x-coord	0.9283679±0.005693819
y-coord	-1.380269±0.005346236

8

x-coord	0.9794889±0.009195064
y-coord	-2.443479±0.00551947

9

x-coord	1.021528±0.01258432
y-coord	-3.261947±0.005479252

10

x-coord	-0.2296379±0.01414995
y-coord	-3.778079±0.004242588

11

x-coord	1.382092±0.01864159
y-coord	-5.714859±0.006666494

12

x-coord	-0.146599±0.01874034
y-coord	-5.580516±0.004522388

13

x-coord	-0.1107091±0.02071819
y-coord	-6.417478±0.0043098

14

x-coord	1.331996±0.02245139
y-coord	-6.929595±0.005137053

15

x-coord	1.291298±0.01669238
y-coord	-4.500734±0.007597451

16

x-coord	2.650251±0.01883532
y-coord	-5.659803±0.01344971

17

x-coord	2.624666±0.01897276
y-coord	-4.452513±0.01429438

18

x-coord	3.507933±0.02127713
y-coord	-4.412083±0.01829519

19

x-coord	4.243449±0.01912411
y-coord	-5.607196±0.02445262

20

x-coord	4.393481±0.0232509
y-coord	-4.371502±0.02640403

B Fast-SLAM Final Positions with 1 Particle

robot:

x-coord	2.011492±0.02821069
y-coord	-5.004015±0.02419571
θ	0.1350971±0.006054444

landmarks:

1

x-coord	-0.0726307±0.01251569
y-coord	-4.488717±0.01202178

2

x-coord	-0.02725145±0.01626886
y-coord	-7.82259±0.0110301

3

x-coord	0.8401273±0.02076425
y-coord	-0.6039244±0.01715057

4

x-coord	0.8364575±0.02505991
y-coord	-0.1009852±0.01791271

5

x-coord	0.8388376±0.02492947
y-coord	0.4051865±0.01781541

6

x-coord	0.3774419±0.02750643
y-coord	0.5517593±0.01842852

7

x-coord	0.8546005±0.01652239
y-coord	-1.404422±0.01461785

8

x-coord	0.8792399±0.01503949
y-coord	-2.454986±0.01508014

9

x-coord	0.9041242±0.01403539
y-coord	-3.278795±0.01442081

10

x-coord	-0.3776308±0.01546177
y-coord	-3.766555±0.01660477

11

x-coord	1.186221±0.01805997
y-coord	-5.757972±0.01801717

12

x-coord	-0.3692551±0.01303151
y-coord	-5.563299±0.0165245

13

x-coord	-0.3363237±0.01287435
y-coord	-6.40503±0.01362745

14

x-coord	1.105217±0.00776473
y-coord	-6.945284±0.007282095

15

x-coord	1.13409±0.0104402
y-coord	-4.483923±0.01216882

16

x-coord	2.467734±0.0162012
y-coord	-5.692416±0.01947901

17

x-coord	2.444523±0.01408941
y-coord	-4.485169±0.02013692

18

x-coord	3.309234±0.01291785
y-coord	-4.453897±0.02349517

19

x-coord	4.050986±0.01214646
y-coord	-5.646511±0.02734382

20

x-coord	4.221325±0.01402204
y-coord	-4.420676±0.02709113

C Fast-SLAM Final Positions with 5 Particles

robot:

x-coord	2.01674±0.02708654
y-coord	-4.99891±0.0273283
θ	0.1346123±0.005286946

landmarks:

1

x-coord	-0.07080915±0.01314042
y-coord	-4.486401±0.01412719

2

x-coord	-0.02611555±0.01594952
y-coord	-7.820933±0.01158092

3

x-coord	0.8440536±0.02006504
y-coord	-0.5994053±0.01972933

4

x-coord	0.8412442±0.02295064
y-coord	-0.09583745±0.02257247

5

x-coord	0.8434365±0.02282394
y-coord	0.4094631±0.02004901

6

x-coord	0.3830352±0.02670794
y-coord	0.5563398±0.02222284

7

x-coord	0.8579303±0.01674416
y-coord	-1.401299±0.0162186

8

x-coord	0.8823241±0.01746826
y-coord	-2.45114±0.01889523

9

x-coord	0.9068959±0.01548947
y-coord	-3.275682±0.01708995

10

x-coord	-0.37489±0.01730449
y-coord	-3.763143±0.01856233

11

x-coord	1.188855±0.01819072
y-coord	-5.754288±0.01887275

12

x-coord	-0.3671148±0.01545349
y-coord	-5.560923±0.01823569

13

x-coord	-0.3347021±0.01547592
y-coord	-6.403716±0.01451064

14

x-coord	1.104947±0.0120714
y-coord	-6.946809±0.006062509

15

x-coord	1.136066±0.01263561
y-coord	-4.484384±0.009760573

16

x-coord	2.470288±0.01591673
y-coord	-5.691199±0.01674308

17

x-coord	2.448078±0.01544146
y-coord	-4.483994±0.01714159

18

x-coord	3.312545±0.01629461
y-coord	-4.454384±0.01859757

19

x-coord	4.053545±0.01579348
y-coord	-5.647648±0.02040863

20

x-coord	4.224726±0.014999
y-coord	-4.421692±0.02130988

D Fast-SLAM Final Positions with 10 Particles

robot:

x-coord	2.011339±0.0243674
y-coord	-5.002299±0.0229648
θ	0.1352967±0.004214557

landmarks:

1

x-coord	-0.072836±0.009816194
y-coord	-4.488274±0.01536093

2

x-coord	-0.0274093±0.01313079
y-coord	-7.822346±0.01352804

3

x-coord	0.8398415±0.01559317
y-coord	-0.601919±0.01948241

4

x-coord	0.8361445±0.01746717
y-coord	-0.0985036±0.0197776

5

x-coord	0.8375508±0.01639406
y-coord	0.4065005±0.018959

6

x-coord	0.3770284±0.01982852
y-coord	0.5540131±0.02164259

7

x-coord	0.8543283±0.01245435
y-coord	-1.403352±0.0174129

8

x-coord	0.8787649±0.01238638
y-coord	-2.453651±0.01876922

9

x-coord	0.903618±0.01247762
y-coord	-3.277405±0.01760583

10

x-coord	-0.3782856±0.01285102
y-coord	-3.765332±0.02112121

11

x-coord	1.185351±0.01681587
y-coord	-5.757619±0.01919561

12

x-coord	-0.3696762±0.01509186
y-coord	-5.563778±0.01978318

13

x-coord	-0.3367895±0.01398403
y-coord	-6.405321±0.01690565

14

x-coord	1.105852±0.00798818
y-coord	-6.9464±0.007848657

15

x-coord	1.135254±0.01666164
y-coord	-4.483909±0.01211002

16

x-coord	2.467531±0.01586785
y-coord	-5.693438±0.01414276

17

x-coord	2.444801±0.01591954
y-coord	-4.487026±0.01441867

18

x-coord	3.309179±0.01531773
y-coord	-4.457128±0.01394334

19

x-coord	4.051163±0.01549462
y-coord	-5.649519±0.01318395

20

x-coord	4.223012±0.0165645
y-coord	-4.423824±0.01501421

E Fast-SLAM Final Positions with 15 Particles

robot:

x-coord	2.020859±0.02769245
y-coord	-4.997112±0.02425127
θ	0.1351612±0.004638845

landmarks:

1

x-coord	-0.0683886±0.01457787
y-coord	-4.486134±0.01125092

2

x-coord	-0.02355365±0.01753593
y-coord	-7.820295±0.01013212

3

x-coord	0.844536±0.02139993
y-coord	-0.5993589±0.01836231

4

x-coord	0.8410576±0.02152201
y-coord	-0.09529275±0.0180306

5

x-coord	0.8428543±0.02164051
y-coord	0.4097006±0.01755446

6

x-coord	0.3822344±0.02548524
y-coord	0.5568083±0.0207873

7

x-coord	0.8590912±0.01778097
y-coord	-1.400842±0.01478389

8

x-coord	0.8838239±0.01841002
y-coord	-2.451014±0.01610406

9

x-coord	0.9089127±0.01830081
y-coord	-3.274915±0.01514474

10

x-coord	-0.3730185±0.01888963
y-coord	-3.762892±0.01670978

11

x-coord	1.191831±0.02294062
y-coord	-5.753556±0.01738512

12

x-coord	-0.3654321±0.0170662
y-coord	-5.560766±0.01482437

13

x-coord	-0.3334429±0.01596545
y-coord	-6.403216±0.01324295

14

x-coord	1.105434±0.008602093
y-coord	-6.945881±0.005625694

15

x-coord	1.136026±0.01071869
y-coord	-4.483029±0.009341818

16

x-coord	2.473369±0.01705927
y-coord	-5.689323±0.01603325

17

x-coord	2.449836±0.01671139
y-coord	-4.482749±0.01599385

18

x-coord	3.313555±0.01611884
y-coord	-4.452533±0.01874409

19

x-coord	4.055891±0.01664316
y-coord	-5.644668±0.02037635

20

x-coord	4.227131±0.0159397
y-coord	-4.419754±0.02030166

F Fast-SLAM Final Positions with 20 Particles

robot:

x-coord	2.016591±0.03263396
y-coord	-4.99972±0.02163271
θ	0.1349555±0.003446475

landmarks:

1

x-coord	-0.07063195±0.01428006
y-coord	-4.487603±0.008627023

2

x-coord	-0.0251012±0.01174992
y-coord	-7.821436±0.007101704

3

x-coord	0.8422814±0.02715051
y-coord	-0.601779±0.01292319

4

x-coord	0.8388169±0.03076046
y-coord	-0.0982437±0.0154949

5

x-coord	0.8406273±0.02883139
y-coord	0.4072412±0.01315559

6

x-coord	0.3800431±0.0325306
y-coord	0.5542139±0.01458139

7

x-coord	0.8565249±0.02165489
y-coord	-1.402641±0.0111247

8

x-coord	0.8810807±0.02169404
y-coord	-2.452777±0.01268292

9

x-coord	0.9059774±0.02013731
y-coord	-3.277093±0.0116487

10

x-coord	-0.3760828±0.02068492
y-coord	-3.764884±0.01133446

11

x-coord	1.188219±0.02030978
y-coord	-5.755993±0.01249105

12

x-coord	-0.3677867±0.01533378
y-coord	-5.56279±0.01205016

13

x-coord	-0.3342065±0.01214873
y-coord	-6.404769±0.009447917

14

x-coord	1.106997±0.00679259
y-coord	-6.945634±0.005585666

15

x-coord	1.135275±0.01048948
y-coord	-4.484386±0.009733115

16

x-coord	2.470168±0.01663974
y-coord	-5.692177±0.01385232

17

x-coord	2.447011±0.01672222
y-coord	-4.484702±0.01380455

18

x-coord	3.311175±0.01487761
y-coord	-4.454559±0.01603023

19

x-coord	4.053264±0.01511823
y-coord	-5.647383±0.01841125

20

x-coord	4.224321±0.01632536
y-coord	-4.421219±0.01934296

G CAS RNT EKF SLAM Final Positions

robot:

x-coord	2.227480
y-coord	-4.876396
θ	0.176675

landmarks:

1

x-coord	0.135581
y-coord	-4.454501

2

x-coord	0.341065
y-coord	-7.794719

3

x-coord	0.915860
y-coord	-0.553527

4

x-coord	0.882833
y-coord	-0.037513

5

x-coord	0.839886
y-coord	0.485805

6

x-coord	0.407606
y-coord	0.570287

7

x-coord	0.951908
y-coord	-1.354704

8

x-coord	1.017316
y-coord	-2.413731

9

x-coord	1.070594
y-coord	-3.228953

10

x-coord	-0.176791
y-coord	-3.757147

11

x-coord	1.462635
y-coord	-5.670841

12

x-coord	-0.072115
y-coord	-5.554269

13

x-coord	-0.010484
y-coord	-6.383051

14

x-coord	1.450673
y-coord	-6.870035

15

x-coord	1.360587
y-coord	-4.431411

16

x-coord	2.737572
y-coord	-5.567522

17

x-coord	2.677667
y-coord	-4.361122

18

x-coord	3.553057
y-coord	-4.293975

19

x-coord	4.324132
y-coord	-5.457760

20

x-coord	4.437676
y-coord	-4.214785

References

Arras, K. O. (2003). *Feature-based Robot Navigation in Known and Unknown Environments*.

Bailey, T. and Durrant-Whyte, H. (2006). Simultaneous localization and mapping (SLAM): part II. *Robotics & Automation Magazine, IEEE*, 13(3):108117.

CAS Robot Navigation Toolbox. The CAS robot navigation toolbox homepage. http://www.cas.kth.se/toolbox/index.html.

Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L. E., and Thrun, S. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press.

DAPRA DRC. DARPA DRC | DARPA robotics challenge home. http://www.theroboticschallenge.org/.

Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localization and mapping: part i. *Robotics & Automation Magazine, IEEE*, 13(2):99110.

Grisetti, G., Stachniss, C., and Burgard, W. (2005). Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling.

Grisetti, G., Stachniss, C., and Burgard, W. (2006). Improved techniques for grid mapping with Rao-Blackwellized particle filters. *IEEE Transactions on Robotics*.

Kaplan, D. T. (1999). *Resampling Statis in MATLAB*. Resampling Stats Inc., Arlington, Virginia.

Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). FastSLAM: a factored solution to simultaneous localization and mapping problem. In *AAAI-02*.

Riisgaard, S. and Blas, M. R. (2003). SLAM for dummies. *A Tutorial Approach to Simultaneous Localization and Mapping*, 22:1127.