Autonomous Robot Navigation based on Simultaneous Localization and Mapping (SLAM), and Particle Filtering

by

Amirhossein Monjazeb

B. Eng.

A thesis submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements for the degree of

Master of Applied Science

Ottawa-Carleton Institute for Mechanical and Aerospace Engineering

Department of Mechanical and Aerospace Engineering

Carleton University

Ottawa, Ontario

Canada

December, 2007

Your file Votre référence
ISBN: 978-0-494-36829-9
Our file Notre référence
ISBN: 978-0-494-36829-9

The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis

submitted by

Amirhossein Monjazeb

B. Eng.

in partial fulfillment of the requirements for
the degree of Master of Applied Science

J. Z. Sasiadek

Thesis Supervisor

Chair, Dept. of Mechanical and Aerospace Engineering

Carleton University
December 2007

To my lovely wife Lida

Abstract

This thesis provides a brief introduction to Simultaneous Localization and Mapping (SLAM) in extensive outdoor environments, a process that recently has made mobile robots truly autonomous. SLAM is a process in which localization and mapping are done simultaneously in an unknown environment without an access to a priori map. To solve the SLAM problem, sensor information and mathematical motion models are extensively used. This research introduces a probabilistic approach to a SLAM problem and under Gaussian and non-Gaussian conditions offers alternative solutions. Due to the non-linearity of the real systems, which are always subject to noise, an Extended Kalman Filter algorithm for the SLAM problem under Gaussian condition will be shown. Then an alternative way of dealing with SLAM problem with assumption of non-Gaussian and called FAST-SLAM will be analyzed. FAST-SLAM is an algorithm that using Rao-Blackwellised method for particle filtering, estimates the path of robot while the landmarks positions which are mutually independent and with no correlation, can be estimated by EKF. This process is doable using a factorization that fits very well in SLAM problem since there will be a representation of SLAM through out a Bayesian network. A solution to SLAM problem can be worthy if a proper representation of motion and sensor of a robot is accessible. In this thesis, a real outdoor autonomous robot is presented and the related equations are derived. Then simulated examples based on both filters are represented and results are discussed and compared.

Acknowledgment

I would like to thank all of the people who encouraged me and supported me during the undertaking of this research. To begin, I would like to thank my thesis supervisor, Professor Jurek Z. Sasiadek, for introducing me to one of the most interesting research fields and for his availability, expertise and advice to fulfill this research. Many thanks to Dr. Vladimir Polotski from Frontline Robotics Inc. for his motivation, and encouragement to move forward.

I want to say thanks to Dr. Michael Montemerlo from Carnegie Mellon University, whose work was the best inspiration to this research and for his supporting words and emails. Special thanks to Professor Hugh Durrant-Whyte, Dr. Tim Bailey, and Professor Eduardo Nebot form Australian Centre for Field Robotics in Sydney, who patiently answered my research questions.

I am deeply thankful to my Mother and Father whose support and care were encouraging me during this work and to my beloved sister Golchehr who was always available for reviewing my work and was with me to the end. Finally, last but not certainly least, thanks to my lovely wife Lida whose love and patience was the most valuable support to me.

Online Support

A free software code for robotics research intentions was used in this particular study. The software in form of both MATLAB and C++ codes is available at http://www.lasmea.univ-bpclermont.fr/ftp/pub/trassou/SLAM/SLAM_Summer_School2002/SLAM%20Summer%20School%202002.htm

Contents

Notation

k	discrete time index
$\mathbf{X}_k^R$	path of the robot upto time step k
$\mathbf{x}_k^R$	state of the robot at time step k
$\mathbf{u}_k$	control vector applied at time
$\mathbf{m}_i$	vector of true location of the i^{th} landmark
$\mathbf{z}_{k,i}$	observation of the i^{th} landmark
$\mathbf{z}_k$	generic observation of one or more landmarks
$\mathbf{m}$	set of all landmarks
$\mathbf{Z}_k$	set of all observations
$N(\mu, \sigma)$	normal distribution with mean μ and covariance σ
$\mathbf{w}_k$	system noise
$\mathbf{Q}_k$	system noise covariance matrix
$\mathbf{v}_k$	observation noise
$\mathbf{R}_k$	observation noise covariance matrix
$f(.)$	non-linear motion function
$h(.)$	non-linear sensor function
$\hat{\mathbf{x}}_k^+$	posterior state

$\mathbf{P}_k^+$	posterior covariance matrix of the system
$\hat{\mathbf{x}}_k^-$	prior state
Φ_k	matrix of the partial derivatives of system function
$\hat{\mathbf{z}}_k^-$	prediction observation
$\tilde{\mathbf{z}}_k$	innovation
$\tilde{\mathbf{Z}}_k$	innovation covariance matrix
Ψ_k	matrix of partial derivatives of the observation function
$\mathbf{K}_k$	Kalman filter gain
V^R	velocity of the middle back axel of the robotic vehicle
V^e	velocity of the vehicle measured by the encoder reader
β	sensor bearing
D	distance between sensor and landmark
ℓ	wheel to wheel distance
$\mathbf{X}^S$	sensor path
$\mathbf{x}^S$	sensor pose
V_R	robot linear velocity
L^S	location of center of range sensor
L^C	location of middle point of the back axel of the robot
$\mathbf{T}_\varphi$	transformation vector
φ	angle between robot's heading and the global refrence system
$\mathbf{w}_k^u$	control noise

w_k^R	motion noise
$\mathbf{x}^{\mathbf{m}}$	vector of all landmarks locations
$\mathbf{d}_k$	set of all data association
${}^{\mathbf{n}}\mathbf{x}_k^R$	state of the robot for particle n
${}^{\mathbf{n}}\mathbf{X}_k^R$.	path of the robot for particle n
$\mathbf{d}_{k,i}$	data association for landmark *i*
$\hat{\mathbf{d}}_{k,j}$	maximum likelihood
Δ	mean square
${}^{\mathrm{n}}\hat{w}_k$	importance weight

Acronyms

Bel	Robot Belief
Bel^-	Robot Prior Belief
Bel^+	Robot Posterior Belief
CPU	Central Processing Unit
GPS	Global Positioning System
SLAM	Simultaneous Localization and Mapping
KF	Kalman Filter
EKF	Extended Kalman Filter
PF	Particle Filter
R-B	Rao-Blackwellization
RBPF	Rao-Blackwellise Particle Filter
MCL	Monte Carlo Localization
ML	Markov Localization
MLE	Maximum Likelihood Estimator
RMS	Root Mean Square
DBN	Dynamic Bayesian Network
SIR	Sampling Importance Resampling

Chapter 1

Introduction

To make a robot truly autonomous, there were many efforts made in the field of robotics and automation. Simultaneous localization and mapping (SLAM) is one of the methods to make this dream come true. SLAM is a process by which a mobile robot, or an autonomous vehicle, in an unknown location and in an unknown environment, incrementally constructs a navigation map of its surroundings. At the same time, the robot simultaneously, and with the observations of features, uses this navigation map for localization. In other words, the autonomous vehicle tries to do both mapping and localization at the same time. For various reasons, this problem is very complex and has to be solved together. This is due to the fact that the vehicle position has an error at every time step and the obtained map is highly correlated with the error of the vehicle.

During the past decades, many approaches and mathematical tools have been implemented to achieve a proper representation of SLAM and to make the SLAM problem computationally efficient. Among all these approaches, SLAM based on Bayesian estimation theory has been the most successful one. Since a real dynamic system is always subject to noise, a good understanding of noise representation is needed. Furthermore, most dynamic systems in the real world are non-linear which means that the mathematical models describing the dynamic system and related observations can not be considered as a linear problem.

To find an optimal solution for SLAM problem, a non-linear system needs to be approximated by a linear system in terms of mathematical equations. So far, various algorithms have been offered to get a reasonable solution. Among all proposed algorithms, Extended Kalman Filter

(EKF) has been one of the most appropriate tools to deal with a non-linear dynamic system model subject to Gaussian noise. Many successful applications have been done by implementing EKF into the SLAM problem. In fact, EKF has been introduced as a classical solution to SLAM problem. Nonetheless, EKF suffers form some problems that makes it difficult to be used for SLAM. The single-hypothesis data association and quadratic complexity due to the high dimensional Gaussian approximations for states of the robot and landmarks locations, makes the off-diagonal elements of the covariance matrix very large. This causes more complexity and cost increase of computation and in most cases, diverges the filter. Moreover, when a large number of landmarks are present in the environment, the computation becomes almost impossible. As an alternative approach, there has been offered another algorithm which makes the problem with multi-hypothesis data association and logarithmic complexity instead of quadratic. This new approach is called FAST-SLAM that utilizes Rao-Blackwellised particle filter to solve the SLAM problem efficiently. Using FAST-SLAM algorithm, the posterior estimation will be over the robot's pose and landmarks locations. The FAST-SLAM algorithm has been implemented successfully over thousands of landmarks and compare to EKF-SLAM that can only handle a few hundreds of landmarks, it has appeared with considerable advantages. This thesis investigates at first the classical method for dealing with SLAM problem under Gaussian condition using EKF. Then, under non-Gaussian condition the new algorithm (FAST-SLAM) will be introduced, the elements will be derived, simulation results will be demonstrated and differences will be investigated and discussed.

1.1 Autonomous Mobile Robots

1.1.1 Mobility

The mobility of a robot is the degree to which that robot is able to freely move through the environment. Manipulators in industry were the first practical use of robots [1]. These robots were usually used for assembly lines and car manufacturing. They are programmed in such a way that they can perform deferent tasks faster, cheaper and more accurate than their human counterparts. A manipulator consists of a movable arm attached to its base fixed to the ground. These types of robots are not able to change their base locations and are only able to move the arm attached to the base.

In present days, we have more expectations from a robot. We want them to be mobile and not attached to the ground so that they can clean our environment, perform different tasks in hazardous areas, be used in planetary exploration, mine and explore underwater. That is why they have to be able to move freely in an environment and to be able to perform their tasks.

1.1.2 Autonomy

Autonomy of a robot depends on to what extend that robot relies on prior knowledge or information from the environment to achieve its task. There can be three different types of robots: *non-autonomous, semi-autonomous,* and *fully-autonomous* [2].

- **Non-autonomous** robots are remotely steered by humans. The intelligence involved in these robots consists of interpreting the commands received from human controllers.
- **Semi-autonomous** robots can either navigate by themselves or be steered by humans. In dangerous situations human takes full control. In normal situations robots can take over the control.
- **Fully-autonomous** robots are steered solely by the robot themselves. The robots do not require human interaction to fulfill their tasks. Fully-autonomous robot vehicles are capable of intelligent motion and action, without requiring any external guidance to follow or control them [2].

1.1.3 Application Areas

A hazardous environment is a good example for the need of using autonomous robots since inherent danger in these environments make humans participation undesireable. Robots can access traverse, maintain and explore these areas with no concerns. In 1993 an eight-legged mobile robot called *Dante* was to reach the *Lava Lake* in *Antarctica* to study the danger of its gases [3].

For the environments that are very difficult for humans to explore, usage of autonomous robots can be very important. Mission to other planets or into space can be done by robots with no level of concerns for humans. Other robots are also used to explore sea surfaces while following reefs or pipes, and study marine creatures. There were many projects on sub-sea implementations of mobile robots to fulfill specific exploring tasks [4].

Finally, a potentially interesting area where autonomous vehicles can be used is the service sector. Intelligent wheel chairs and robotic vacuum cleaners can improve the life quality of many people. Medicine or food delivery robots can relieve the workload of care workers.
In all above application areas the robot requires autonomy and mobility. A mobile robot to fulfill its task needs to move through the environment safely. This moving around is called *robot navigation.*

1.2 Robot Navigation

Navigation is a task by which the robot travels safely from one location to another. The problem of navigation is summarized in four general questions:

- *What does the environment look like?* The robot has to know a map of the environment to find its path among features (Landmarks). Finding out how the environment looks like, is called *Mapping.*
- *Where am I?* Finding out the current position and whereabouts of the robot is called *Robot Localization.*
- *Where am I going?* As the robot moves through the environment, it should know where the goal or destination is. Finding out what the next location is, is called *goal recognition.*
- How do I get there? If the robot knows the starting and ending points, it has to find its way to reach the destination. The way the robot plans how to reach its destination is called *path planning.*

1.2.1 Drive System

Drive system of a robot is a mechanism that steers a vehicle along a path. Obviously the drive system plays an important role for the robot movements. It directly changes the location of a mobile robot. A proper drive system gives robot ability to move around its environment more freely. The more degree-of-freedom in a locomotion mechanism, gives a robot more mobility. If an initial position is known, the subsequent changesin position could be detected. This results in *relative position measurement*, or *dead reckoning* procedure.

1.2.2 Dead Reckoning and Optical Encoders

Dead reckoning [5, 6] is a method based on sensor measurements to determine the current location of an autonomous robot by advancing some previous position through a known path and velocity information over a given length of time. Nowadays, most of land-based autonomous robots (In a two-dimensional environment) use this procedure as the base of their navigation strategies. The simple implementation of dead reckoning is sometimes mentioned as relative position measurement or *odometry*. It simply means that the displacement of an autonomous vehicle can be measured by a device attached to the robot. This device is usually an *optical encoder*. An optical encoder [7] is a device by which the rotation of wheel of a robot can be measured and the linear displacement of a robot can be determined. There are two types of optical encoders, *incremental* and *absolute*. The incremental type measures rotational velocity and can infer relative position. The absolute version directly measure angular position and infer velocity.

1.2.3 Heading sensors

Heading sensors [6] are very important for navigation purposes. In an odometric-based positioning method, any small momentary orientation error will cause a constantly growing lateral position error. For this reason it would be of great benefit if orientation errors could be detected and corrected right away. *Gyroscopes* and *accelerometers* [8] are two widely used sensors for determining the *heading* or *orientation* of a robot.

Gyroscopes detect angular accelerations about three axes [6]. Accelerometers [7] measure small accelerations along x and y axes attached to the coordinate system of the robot. They suffer from extensive drift and are sensitive to uneven ground. This problem can partially be solved by adding a *tilt sensor* to it so that it can cancel the gravity component. However, the drift will still be considerable [4].

1.2.4 Range/Bearing Sensors

A very important part of navigation systems are *range/bearing sensors* [9]. Range/bearing sensors obtain information from the environment that the robot is traveling through. Using range/bearing sensors the *absolute position measurement* (the position and angle of landmarks with respect to the coordinate system of the robot/sensor) can be obtained. The absolute position measurement provides information about the location of landmarks independent of previous location of the robot. Unlike dead reckoning the measurement in this case is not derived from a sequence of measurements, but directly from a single measurement.

The range/bearing sensor in many applications is applied as a laser scanner that sends laser beams to the environment and identifies features. A common laser sensor in outdoor applications is SICK sensor that returns range and bearing. It gets 360 samples of ranges separated by 0.5 degree. There are the two observations of relative distance of the landmark and its bearing to the robot's coordinate system.

The second possibility can be a sonar sensor. Sonars are very cheap compare to laser scanners. Likewise, the measurement is not accurate enough for some applications [10].

The third alternative can be the use of a vision system. Traditionally, it has been very computationally intensive. It is apparent that a room without light makes a big trouble for feature detecting and building a map.

For the experiments in this study, laser scanner sensors have been considered since they match with the most outdoor applications in the real world while different types of sensors can be extensively used in different applications.

1.2.5 Landmarks

For navigation purposes, there is a need of some features in the environment that can be easily re-observed by a robot; any parameterized model can be chosen to represent map of the environment, however it is commonly assumed to be a collection of point features. A collection of points or *Landmarks* relative to some external (Global) coordinate system can be used by a robot to localize itself. One way to imagine how a robot finds its way in an environment full of landmarks is to picture a person finding his way in a crowded room with closed eyes. As he reaches and touches every object or wall, he estimates his path and finds

his way through it. In a similar way, a robot identifies landmarks in an environment using its range sensor. In indoor applications, landmarks can be considered doors, walls, other regular household objects and even artificial signs designed by humans. In outdoor applications they can be trees, bushes, stones or any artificial landmarks such as studs or signs. Whatever application and environment the landmarks are considered for, they must be easily re-observable by sensors, distinguishable from each other. Landmarks are usually stationary in SLAM applications but dynamic landmarks have been recently considered for some specific applications [11].

1.2.6 Errors and Uncertainty

Determining the location of a robot and in the same time constructing a map in an unstructured environment is an important problem in navigating an autonomous vehicle. In a two-dimensional environment, the location of an autonomous robot can be described as (x, y), the coordinates of the robot, and Θ, the orientation of it with respect to the global coordination system. In the navigation problem an autonomous vehicle is meant to build a map of its environment while it is trying to follow its desired path.

The problem arises from the fact that there is always error associated with the motion of the robot. For example, in a four-wheeled robot system, the robot controller uses the information provided by the optical encoders attached to the wheels to command the guidance system mechanism to keep the robot near its desired path at all time. This mechanism is usually referred to *odometry*. However, the robot system mechanism may not follow commands very well and the measurements from the optical encoders contain errors. As the time goes by, the

error from encoder readings accumulates and becomes the major source of uncertainty in the system. The uncertainty could grow to a point that it becomes impossible for a robot to make any meaningful inference about its whereabouts. Beside the encoders error, the slippage due to uneven pathways, inaccuracy of range sensor readings (observation of landmarks), and even the changes of wheel diameter in time, can accumulate a significant amount of error in the system. All the above sources of error can result in incorrect position estimation. Therefore, the robot needs to have some notion of the error from all above sources. This error consideration can be taken into account when the robot is trying to find its way.

1.2.7 Other Difficulties

There are many other problems involved in navigation of an autonomous robot. These problems are: *obstacle avoidance*, *landmark recognition*, and *integration of information from different sensors [12]*.

1.2.7.1 Obstacle Avoidance

Robots must avoid colliding with obstacles in the environment at all cost [13]. The problem becomes more complicated if the obstacles are dynamic.

1.2.7.2 Landmark Recognition

An autonomous robot needs to recognize objects and landmarks in order to keep itself on the desired path [14]. If these objects and landmarks are not known in advance, image processing

may need a lot more computational power than if those features were known. Whether or not these structures are known in advance, recognition of them can come with a significant amount of uncertainty.

1.2.7.3 Fusion of Information from Different Sensors

The robot has to combine all the information from different sources to be able to build a map and find its way among them. Since different sensors have different information about the environment and the position of the robot, they might give inconsistent or incorrect sensor readings. Sensor fusion is the problem of combining data from different sensors into one unified view of the environment [12].

1.3 Simultaneous Localization and Mapping

Simultaneous localization and mapping (SLAM) is "a procedure in which an autonomous mobile robot in an unknown environment, and in and unknown location incrementally builds a map of this environment while simultaneously using this map to compute vehicle location"[14]. SLAM problem is much more complicated in comparison with that of mapping itself or localization only. Since both mapping and localization must be estimated simultaneously, they are highly correlated and there is a need for an appropriate algorithm which can deal with the uncertainty of the system, computational complexity and data association due to landmarks localization while building a map. Furthermore, SLAM problem needs to be expressed in a probabilistic form. There have been several algorithms introduced

so far but *EKF-SLAM* and *FAST-SLAM* are two of most practical ones in real world applications.

1.3.1 SLAM based on Extended Kalman Filtering (EKF-SLAM)

EKF-SLAM algorithm is mostly known as a classical solution to SLAM problem. The EKF algorithm providing an optimal solution for SLAM problem was first introduced by Smith and Cheesman in 1986 [15]. This algorithm is based on standard Kalman filter [16]. It is necessary for the process to be under Gaussian conditions. Furthermore, the system should be linear or at least not very far away from it, so it could be linearized.

Using EKF algorithm, the estimation of the state, and its covariance matrix of the posterior belief $Bel^{+}(\mathbf{x}_k) = P(\mathbf{x}_k | \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0)$, can be computed easily. The procedure estimate is done in three major steps; *prediction, observation,* and *update*. The EKF algorithm can be implemented as a mathematical tool to achieve a Bayesian solution. As we will see in chapter 3, the belief of where a robot is, will be represented as a parameterized continuous function.

Unfortunately, EKF suffers from two major problems for a large scale map of environment; The *computational complexity* and *data association*. There have been several efforts to overcome these problems but it still has some influence on the performance and convergence of the filter.

1.3.2 SLAM based on Rao-Blackwellised Particle Filtering

SLAM can be solved according to Rao-Blackwellised particle filtering called FAST-SLAM. It is based on straight forward factorization. This factorization is based on the observation that if the trough path of a robot is known, the individual landmark localization problems are mutually independent. In other words, if there is no uncertainty about the robot pose and the observations are independent, the estimation of each landmark in the map is independent of the rest of the map. In reality, the path is unknown in advance. However, the conditional independence enables the filter to estimate the general probability distribution in factored term according to equation (1.1).

$$P(\mathbf{X}_k^R, \mathbf{m} \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R) = P(\mathbf{X}_k^R \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R)\, P(\mathbf{m} \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R)$$

$$= P(\mathbf{X}_k^R \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R) \times \prod_{i=1}^{M} P(\mathbf{m}_i \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R) \tag{1.1}$$

The first factor is the estimation of the robot path and the second factor in the above equation is the product of M landmarks estimation given the robot pose is known. This factorization is the fundamental idea behind FAST-SLAM method. In fact FAST-SLAM decomposes the SLAM problem into a localization and M landmark position estimation problems. Furthermore, FAST-SLAM relies on a so called particle filter to estimate the following robot posterior:

$$P(\mathbf{X}_k^R \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R) \tag{1.2}$$

This particle filter can be updated in constant time for each particle in the filter. To update the map, the algorithm relies on *M* independent Kalman filters for the M landmark estimates.

$$P(\mathbf{m} \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R) \tag{1.3}$$

As will be discussed in chapter 5, the entire filter can be updated using logarithmic scale in the number of landmarks *M*. FAST-SLAM method can also handle non-linear robot motion models. Besides this non-linearity of motion advantage, the noise does not have to be Gaussian anymore but it can be any type of noise.

1.4 Background

FAST-SLAM algorithm was introduced by Michael Montemerlo [11] and was further developed by Thrun, Montemerlo, Koller, and Wegbreit [17]. The previous work on landmark based environmental representation with EKF-SLAM was accomplished by Hugh Durrant-Whyte and colleagues [14, 18, 19]. So far, consistency and convergence of both algorithms have been in debate [10, 20, 21, 22]. More specifically, the complexity in terms of computation and the data association with observation of different landmarks has been an open subject for discussion [23, 24, 25].

1.5 Objectives

The objective of this thesis was to develop and evaluate an efficient navigation method based on Rao-Blackwellised particle filtering (RBPF) method called FAST-SLAM. The following considerations have been made:

- The method (SLAM) relies on correct correspondence between data obtained from the mounted sensor and the data currently stored in the map. In a natural environment, the varied distribution of recognizable objects (landmarks), makes the data association algorithm unsuitable for large number of landmarks.
- Real time operation in a very large environment, requires high computational and storage demands that makes the use of EKF-SLAM algorithm inefficient
- The process of building an incremental map of the environment and at the same time localizing, is highly non-linear. Due to the existence of a significant error accumulation during the navigation in a large environment, SLAM algorithm must remain mathematically consistent.

This investigation is based on landmark environmental representation for an outdoor mobile robot vehicle with a mounted range/bearing sensor in a noisy environment. Localizing and mapping aspects of the system will be investigated since they both have to be solved together. Likely, different solutions would be detained using two different algorithms, EKF-SLAM and FAST-SLAM, and the results, are compared and investigated for both Gaussian and non-Gaussian assumptions. The FAST-SLAM algorithm with logarithmic scale will be analyzed

and the results will be investigated and compared with FAST-SLAM method but using linear scale. In this thesis the autonomous robot has been a wheeled robot in all experiments. It should be noted that landmarks are assumed to be stationary in this research.

1.6 Thesis Outline

Chapter 1 of this thesis introduces autonomous robots and their navigation. In chapter 2 the SLAM problem and a probabilistic form of it with a Bayesian representation is introduced. In chapter 3 the Extended Kalman Filter (EKF) as a classical approach to solve the SLAM problem is investigated. As mentioned before, the EKF-SLAM can be an appropriate solution for linear or linearizable systems. Furthermore, an assumption of Gaussian system has been made in this chapter. Chapter 4 presents results from simulation experiments for EKF-SLAM. Chapter 5 describes an alternative approach called FAST-SLAM based on Rao-Blackwellised particle filtering. It also includes results from simulation experiments for logarithmic FAST-SLAM and comparison of it with the linear FAST-SLAM algorithm. Several navigation cases for both EKF-SLAM and FAST-SLAM methods were shown in chapter 6. Chapter 7 includes summary and conclusion.

Chapter 2

Simultaneous Localization and Mapping

2.1 Localization Problem

The problem of robot localization consists of answering the question "*Where am I ?*". In robotics, localization is the process of determination the *pose* or *location* of an autonomous robot [26]. Location of a robot in a two-dimensional environment [27] is referred to (x, y), the coordinates, and Θ, the orientation or heading of the robot. In localization problem, a map and the initial location of the robot are known and the goal is to keep track of the position while following a *priori map* [28]. This map describes the environment and can be a geometric map, map of occupancy or map of *passive* or *active* landmarks [29]. With access to a priori map and a sequence of control actions U_k, the robot can find its location relative to the environment. By applying tracking or local techniques this problem can be solved easily [30].

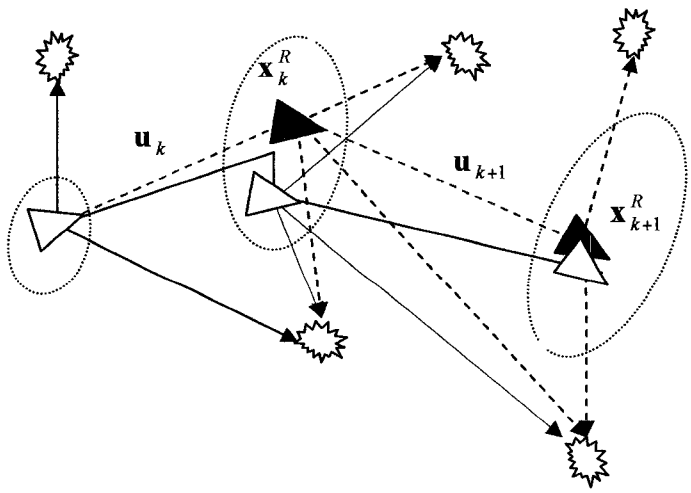


Figure 2.1: A robot that is following a priori map using observations of many landmarks. [31].

One method for localization is the detection of passive landmarks [31]. Passive landmarks are the features in the environment that could be natural such as a tree or artificial such as a bar code. The robot can detect passive landmarks using its sensor. Once the landmarks are detected, the information is matched with a priori map and the position of the robot can be determined. Figure 2.1 indicates the localization problem. The ellipses represent the relative *uncertainty* of the robot with respect to its initial position [15]. Note that the uncertainty grows as the robot keeps traveling along the path. This uncertainty grows unbounded and reaches to a point that it is impossible for robot to localize itself.

2.2 Mapping Problem

Mapping is a process of obtaining a map of the environment. The problem of mapping consists of answering the question "*What does the environment look like?*". The goal of mapping is for a mobile robot to build a map while following a trajectory. This trajectory means that the robotic vehicle path $\mathbf{X}_k^R$ is known in advance and there is no need for path estimates. Then the robot makes inferences to acquire a two dimensional map $\mathbf{m}$ [27]. The output is a map of the environment that represents a view of robot's surroundings. In the same way, this map can be a geometric map, map of occupancy or map of landmarks [28]. To acquire this map, the robot must use its sensor to perceive the environment. Figure 2.2 illustrates the mapping problem. This time, the ellipses represent the relative uncertainty of the landmark observations. As the robots travels on its path, the errors of landmarks observations and consequently the uncertainty of the system grows.

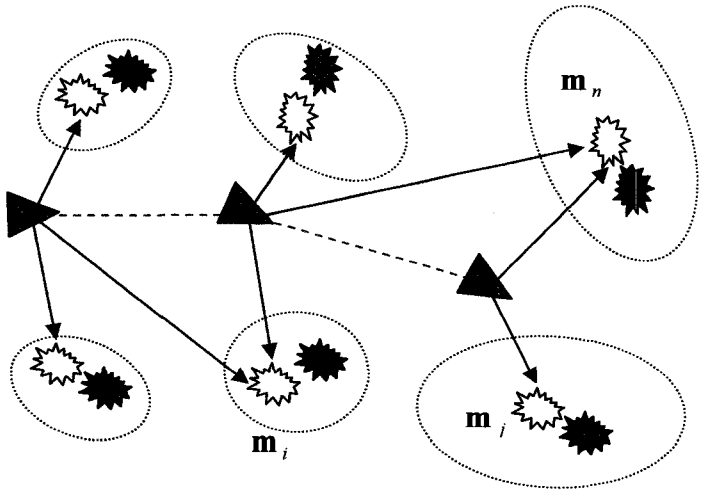


Figure 2.2: A robot is observing the outside world by its range sensor to acquire a map while it is traveling on a priori path [31].

2.3 SLAM Problem

Simultaneous Localization and Mapping (SLAM) [14, 18, 19, 27, 32] describes a situation in which a mobile robot is placed in an unknown environment while there is no access to a priori map and only with sensing the environment and trough some sequential controls, localize itself and at the same time incrementally builds a navigation map of its surroundings. Localizing and mapping are both highly correlated factors when a solution to SLAM problem is acquired. It is important to note that the initial condition of the robot should be given before any attempt for solving the problem. Furthermore, both sensor data and motion of the robot are corrupted by unwanted noise. Figure 2.3 illustrates the SLAM problem form a visual point. The ellipses indicate the uncertainty of both robot's path and landmarks positions that grow through the process. As a matter of fact, the uncertainty over landmarks grows with the

uncertainty of the path unbounded. The reason is that, in SLAM, there always is a high correlation between the robot's pose and a landmark position. Before analyzing the alternative solutions to SLAM problem, some elements and sets are being considered. The elements and sets [14] are defined as follows:

- A discrete time index $k = 1, 2, \ldots$
- $\mathbf{x}_k^R$: Vector of the true location and orientation of the robot at the discrete time *k*.
- $\mathbf{u}_k$: A known control vector applied at time *k-1* to drive the vehicle from $\mathbf{x}_{k-1}^R$ to $\mathbf{x}_k^R$ at time step *k*.
- $\mathbf{m}_i$: The vector of true location or parameterization of the i^{th} landmark.
- $\mathbf{z}_{k,i}$: Measurement or observation of the i^{th} landmark taken from a location $\mathbf{x}_k^R$ at time step *k*.
- $\mathbf{z}_k$: The generic observation of one or more landmarks taken at time step *k*.
- The set of history of states or path of the robot:

$$\mathbf{X}_k^R = \{\mathbf{x}_0^R, \mathbf{x}_1^R, \ldots, \mathbf{x}_k^R\} = \{\mathbf{X}_{k-1}^R, \mathbf{x}_k^R\}$$

- The set of history of control inputs:

$$\mathbf{U}_k = \{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_k\} = \{\mathbf{U}_{k-1}, \mathbf{u}_k\}$$

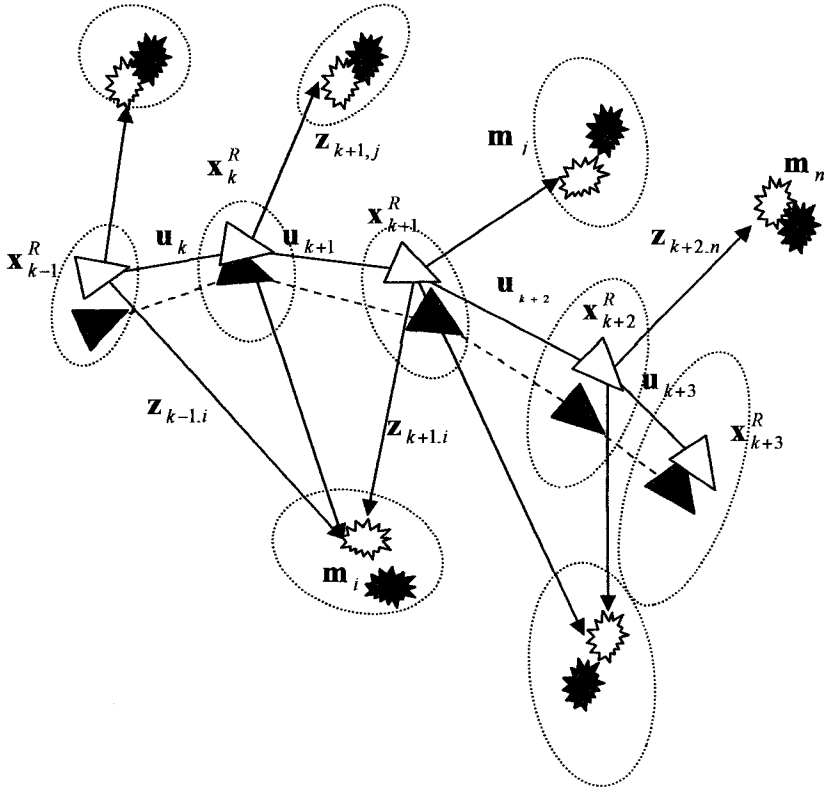
- The set of all landmarks:

$$\mathbf{m} = \{\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_M\}$$

- The set of history of observations (Landmark observations) :

$$\mathbf{Z}_k = \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_k\} = \{\mathbf{Z}_{k-1}, \mathbf{z}_k\}$$

Where $\mathbf{z}_k = \{ \mathbf{z}_{k,i}, \mathbf{z}_{k,j}, \ldots, \mathbf{z}_{k,n} \}$ is a set of all possible landmark observations at time step k. Also the ith observed landmark at time step k is expressed as $\mathbf{m}_{k,i}$ or simply $\mathbf{m}_i$. The reason for skipping sub-script k is that the landmarks are considered static in this representation and therefore they are time independent. So there will be no need of time step for any specific landmark.



	Robotic vehicle	Landmark
Estimated path	▶	✹
Real path	▷	✹

Figure 2.3: A robotic vehicle that moves in an unknown environment and takes observations of many landmarks using a range/bearing sensor [14].

2.4 Probabilistic SLAM

As discussed in chapter 1, different information from different sources and sensors (relative and absolute position information) are being combined so that filters such as EKF and RBPF make the best estimation out of the result. In this chapter we focus on how different position measurements can be combined in a formal probabilistic problem.

2.4.1 Probability Distribution

The general SLAM problem as discussed before can be described in the form of a probability distribution function [30, 33, 34, 35]. If the initial state of a mobile robot $\mathbf{x}_0^R$ (state at the time step zero) and the recorded landmark observations $\mathbf{Z}_k$ and control inputs $\mathbf{U}_k$ up to and including time step k are known, the joint posterior density of the landmark locations and the state of the robot can be recursively described by the following probability distribution for all time steps k [14, 30].

$$P(\mathbf{x}_k^R, \mathbf{m} \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R) \tag{2.1}$$

2.4.2 Process Models

If state of motion of the robot, $\mathbf{x}_k^R$ and also the true locations of landmarks $\mathbf{m}$ are known, observation and motion models describe the probability of making an observation of all

landmarks $\mathbf{z}_k$ and changing of location of a mobile robot simultaneously. Therefore, we can describe the sensing and motion of the robot in a probabilistic form [35]. For simplicity, and throughout this thesis, at all time steps, we will note the motion of the robot and map of land marks as $\mathbf{x}_k$ in the form of following augmented set [26] and it will be denoted as state of the whole system.

$$\mathbf{x}_k = \{\mathbf{x}_k^R, \mathbf{m}\} \tag{2.2}$$

2.4.2.1 Sensor Model

The observational model can also be described in probabilistic terms. This probability [14, 31] can be described the way a range sensor observes $\mathbf{z}_k$ from a certain location $\mathbf{x}_k$ at time step k by the density

$$P(\mathbf{z}_k \mid \mathbf{x}_k) \tag{2.3}$$

This probability is called the *sensor model* or *perceptual model* [36]. This transition density is usually difficult for the computation. The reason for this is sometimes high dimensionality of the measurement. For instance, if we use a camera for the measurement, the probability density becomes very complex due to each possible camera picture at each possible location which needs a large amount of computing power and memory [37].

2.4.2.2 Motion Model

An autonomous robot performs actions [34] in the environment that change the position of the robot. This action is due to a sequence of control actions. If we let $\mathbf{u}_k$ from the set of control actions $U_k = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k\}$, be the action performed by the robot at time step k, we can express the way the location of the robot changes probabilistically by a transition density as:

$$P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) \tag{2.4}$$

This probability density gives the probability that if at time step *k-1* the robot was at location $\mathbf{x}_{k-1}$, and performed a control action $\mathbf{u}_k$, then it ends up at location $\mathbf{x}_k$, at time step k. this transaction density is therefore motion model that describes how the control input $\mathbf{u}_k$ changes the location of the robot. Notice how actions contain relative information about the new state of the robot in this probability density. Given the last location, the robot can estimate its current location based on the performed control action. Once the state of the robot and the landmark locations are known, we can assume that observations are conditionally independent [30].

2.4.3 Bayesian SLAM

SLAM can be described as a Bayesian estimation problem. A Bayesian framework that estimates this density is the *Markov Localization* framework [30]. This framework captures the probabilistic foundations of many currently used localization methods. This framework

combines information from different sensors in form of relative and absolute position measurements to form the *belief of a robot*. Estimation of location of the robot and also landmark locations can then be described with the consideration of noise in all measurement devices.

2.4.4 Belief of Robot

With a probabilistic point of view, it can be said that the robot has a belief of what the position of landmarks and its pose is. At time step k the robot is not considering one specific location but a number of locations for itself and the landmarks. Through the Markov Localization framework [30], this belief can be described very satisfactorily. Markov Localization is an iterative way of using the Bayesian Theorem to update the available information at each time that new sensor reading is available. The Markov Localization framework combines the data from different sensors to form a combined belief regarding the location of a robot and landmarks. The belief can be described by a probability density over all locations $\mathbf{x}_k \in \boldsymbol{S}$ where $\boldsymbol{S}$ is the set of all locations of the robot and the landmarks (The global coordinate system reference). The belief can be described as follows.

$$\boldsymbol{Bel}(\mathbf{x}_k) = P(\mathbf{x}_k \mid \mathbf{X}_{k-1}, \mathbf{U}_k, \mathbf{x}_0) \tag{2.5}$$

The probability distribution in equation (2.5) indicates the robot's belief at state $\mathbf{x}_k^R$ at time step k, given the initial state $\mathbf{x}_0^R$ and all locations of the robot, map of landmarks $\mathbf{m}$, and the

set of control actions up to time step *k*. This probability distribution has the highest possible probability at which the system can be. The goal of SLAM is to make this belief as close as possible to the real distribution of the system. There is only one peak out of this distribution at the true locations and it is zero everywhere else. This specification matches with the definition of a unimodal system.

2.4.4.1 Prior and Posterior Beliefs

While the robot is navigating through the environment, it incorporates all data coming from different sensors. At time step *k*, the system has a *belief* which is called prior belief of the map and its own location. Once the robot incorporates the measurement information from the observation $\mathbf{z}_k$ at time step *k*, a new belief is created in the system. This new belief of robot which is called the *posterior belief* is indicated as $Bel^{+}(\mathbf{x}_k)$,. This posterior belief is the belief that the system has, after it has also included the latest observation information.

As we have probabilistic models for sensor and motion, we can update the belief in a probabilistic term. The initial condition should always be mentioned when the beliefs are updated. The belief of the system after the control action $\mathbf{u}_k$ at time step *k-1* and before incorporating the new observation $\mathbf{z}_k$, is called the prior belief and is described as

$$Bel^{-}(\mathbf{x}_k) = P(\mathbf{x}_k \mid \mathbf{z}_1, \mathbf{u}_0, \mathbf{z}_2, \mathbf{u}_1, \ldots, \mathbf{z}_{k-1}, \mathbf{u}_k, \mathbf{x}_0) = P(\mathbf{x}_k \mid \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0) \tag{2.6}$$

After obtaining observation $\mathbf{z}_k$ and incorporating it to get to the posterior belief, the posterior belief is described by

$$Bel^{+}(\mathbf{x}_k) = P(\mathbf{x}_k \mid \mathbf{z}_1, \mathbf{u}_0, \mathbf{z}_2, \mathbf{u}_1, \ldots, \mathbf{z}_{k-1}, \mathbf{u}_k, \mathbf{z}_k, \mathbf{x}_0) = P(\mathbf{x}_k \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0) \quad (2.7)$$

The probability densities of both prior and posterior beliefs can now be computed.

2.4.4.2 Total Probability and Markov Localization

Using total probability and the Markov localization [29, 35, 38], we can derive an efficient formula for the next step.

$$Bel^{-}(\mathbf{x}_k) = P(\mathbf{x}_k \mid \mathbf{z}_1, \mathbf{u}_0, \mathbf{z}_2, \mathbf{u}_1, \ldots, \mathbf{z}_{k-1}, \mathbf{u}_k, \mathbf{x}_0) = P(\mathbf{x}_k \mid \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0)$$

$$= \int_s P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0) \times P(\mathbf{x}_{k-1} \mid \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0)\, d\, \mathbf{x}_{k-1} \quad (2.8)$$

The location of the robot at time step *k-1* has nothing to do with the control action $\mathbf{u}_k$. Therefore, we do not have to take the control action $\mathbf{u}_k$ into account when expressing the prior belief of the robot. In figure 2.4 this property is illustrated for a dynamic Bayesian network [39]. According to this property and based on the formulation of the posterior belief, equation (2.8) can be written as follows:

$$Bel^{-}(\mathbf{x}_k) = \int_s P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0) \times P(\mathbf{x}_{k-1} \mid \mathbf{Z}_{k-1}, \mathbf{U}_{k-1}, \mathbf{x}_0)\, d\, \mathbf{x}_{k-1}$$

$$= \int_s P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0) \times Bel^{+}(\mathbf{x}_{k-1})\, d\, \mathbf{x}_{k-1} \quad (2.9)$$

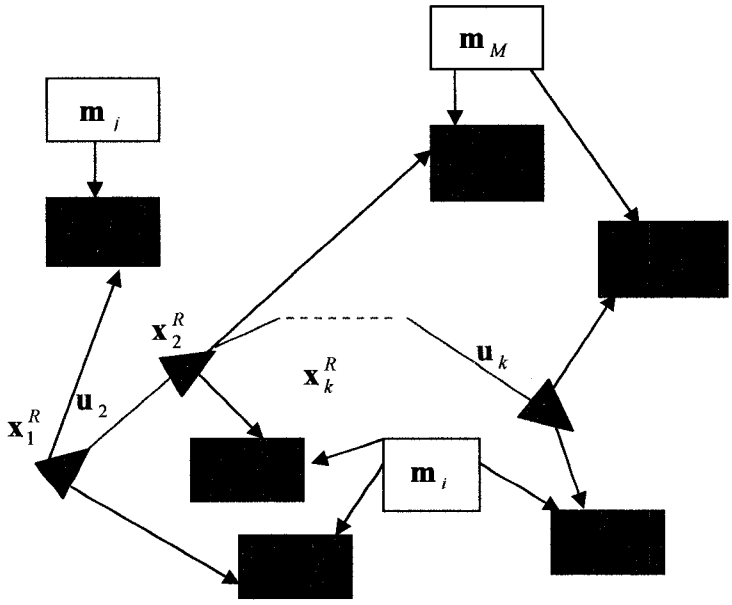


Figure 2.4: A dynamic Bayesian network.

To further simplify equation (2.9), Markov assumption which states that based on information of the current state, the previous state of the system is independent of the current one, can more simplify the prior belief. This assumption simply expresses that it does not matter what the sensor reading was and how the system used the observation information to get to the new state. In fact, the current information of the system is the matter of importance at the current time step. Markov assumption leaves all the previous unnecessary control and observation information behind and will make equation (2.9) much simpler. Therefore, equation (2.9) can be rearranged as

$$Bel^{-}(\mathbf{x}_k) = \int_s P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{U}_k, \mathbf{x}_0) \times Bel^{+}(\mathbf{x}_{k-1})\, \mathrm{d}\,\mathbf{x}_{k-1} \tag{2.10}$$

The observation data can be incorporated into the prior belief to create the posterior belief defined in equation (2.7). Using Bayesian theorem and the definition of the prior belief, equation (2.7) can be rewritten as

$$Bel^{+}(\mathbf{x}_k) \times P(\mathbf{z}_k \mid \mathbf{Z}_{k-1}, \mathbf{U}_k) = P(\mathbf{z}_k \mid \mathbf{x}_k, \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0) \times P(\mathbf{x}_k \mid \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0) \quad (2.11)$$

Using the Markov assumption, equation (2.11) can be more simplified and rewritten as

$$Bel^{+}(\mathbf{x}_k) \times P(\mathbf{z}_k \mid \mathbf{Z}_{k-1}, \mathbf{U}_k) = P(\mathbf{z}_k \mid \mathbf{x}_k) \times P(\mathbf{x}_k \mid \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0) \quad (2.12)$$

Or

$$Bel^{+}(\mathbf{x}_k) \times P(\mathbf{z}_k \mid \mathbf{Z}_{k-1}, \mathbf{U}_k) = P(\mathbf{z}_k \mid \mathbf{x}_k) \times Bel^{-}(\mathbf{x}_k) \quad (2.13)$$

Where

$$Bel^{-}(\mathbf{x}_k) = P(\mathbf{x}_k \mid \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0)$$

And

$$Bel^{+}(\mathbf{x}_k) = P(\mathbf{x}_k \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0)$$

Or simply

$$P(\mathbf{x}_k \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0) \times P(\mathbf{z}_k \mid \mathbf{Z}_{k-1}, \mathbf{U}_k) = P(\mathbf{z}_k \mid \mathbf{x}_k) \times P(\mathbf{x}_k \mid \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0) \quad (2.14)$$

Equation (2.14) provides a recursive procedure for calculating the posterior belief $Bel^{+}(\mathbf{x}_k)$ for the system state $\mathbf{x}_k$ at time step k by incorporating the observation data $\mathbf{Z}_k$ and the control actions $\mathbf{U}_k$ up to and including time step k. This equation is a Bayesian definition of pose of the robot accompanying a sequential control actions and sensor measurements considering with the initial conditions of the robot. Figures 2.5 and 2.6 show three dimensional diagrams of prior and posterior belief update steps in detail. There are many statistical estimation algorithms that are approximations of general Bayes filter, such as Kalman filter and particle filter. In figure 2.5, the green curve depicts the probability of state of the robot at time step k. the pink curve demonstrates the probability of state of the robot at the time step k conditioned on the previous state of the robot, the black curve indicates the function of the motion of the robot, and the orange curve indicates the sum of conditional probabilities of the state of the robot up to time step k.

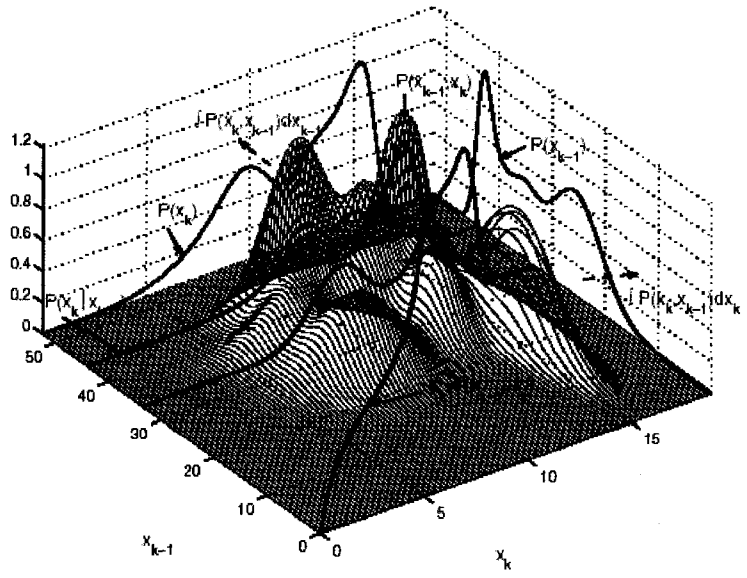


Figure 2.5: The prior belief update step of a mobile robot [31]

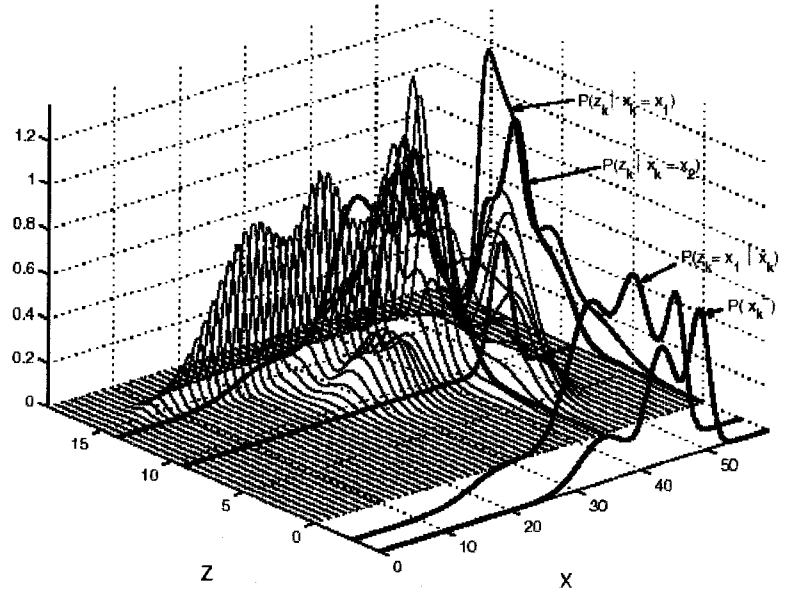


Figure 2.6: The posterior belief update step of a mobile robot [31]

2.5 Summary

The SLAM problem can be approached from a probabilistic point of view. By looking at the problem probabilistically, the concepts of uncertainty and probabilities automatically come into the picture. Instead of being sure its own location in one spot, the robot considers the possibility of a space of locations. Naturally the robot has a belief over the location space. A robot starts with an initial belief. This belief can be a uniform distribution if the robot does not know its location, or it can be a distribution with one pick at the right location if the robot knows where it is.

When the robot is navigating through the environment, it senses the surroundings and receives position information in form of relative and absolute measurements. If the probability distributions of these measurements are known, then they can be incorporated into the location belief of the landmarks and robot itself. We refer to the belief resulting from incorporating a relative measurement as prior belief, and to the belief resulting from incorporating an absolute measurement as posterior belief.

By means of Bayes' rule and Markov Localization, we derive a formula that recursively computes the posterior belief given relative and absolute measurements.

$$P(\mathbf{x}_k \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0) \times P(\mathbf{z}_k \mid \mathbf{Z}_{k-1}, \mathbf{U}_k) = P(\mathbf{z}_k \mid \mathbf{x}_k) \times P(\mathbf{x}_k \mid \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0)$$

To implement the above formula we need to specify three probability density functions; the motion model, the observation model, and the initial conditions. By representing the location space continuously as a parameterized probability density, the computations involved become much easier.

Chapter 3

Extended Kalman Filtering SLAM

3.1 EKF as an Optimal Solution

Following a Bayesian approach to SLAM problem, Kalman filter (KF) [16, 40, 41, 42] can be implemented as a mathematical tool to arrive at a Bayesian solution. As described in the second chapter, the belief was represented as a parameterized continuous function. KF can deal with the computational complexity of beliefs over continuous spaces and is a recursive data processing algorithm that estimates the state of noisy linear dynamic systems. The term state means the location and orientation of the robot and locations of landmarks. The fact that the state of the dynamic system might be noisy and not directly observable makes the state estimation difficult. We must be very careful with regard the term *optimal solution* [43] and should keep in mind that KF can only deal with systems that are represented by Gaussians. This assumption is the only way to make KF a good estimator and to arrive at a reasonable solution. Furthermore, KF deals with linear systems. The limitation of KF based approaches is related to non-linearity lied in the heart of SLAM problem.

Some dynamic systems and sensor models are not completely linear but may not be far from it either. Usually, sensor and motion models are nonlinear but approximately linear for small differences in the values of the state variables. Use of KF as a linear estimator will increase the error in the state estimates so largely. In order to prevent this error to become increasingly large, Extended Kalman Filter (EKF) [16, 37, 41, 42, 44], originally called Kalman-Schmidt

filter [41] is widely used for SLAM solution. EKF is an extended form of basic linear Kalman filter. The EKF algorithm providing an optimal solution for SLAM problem was first introduced by Smith and Cheesman in 1986 [15]. EKF-SLAM algorithm is based on Bayesian theorem as well as KF and has been implemented in many applications both in industry [45] and research [21, 46] fields. Likewise, the conditions of the system must follow some prerequisites such as linearity of the system and a Gaussian representation of the state and also noise by which the system is subject to. Even in cases that the system is not linear, it must be linearizable for the EKF algorithm to be able to achieve a reasonable solution. Only under these conditions the filter converges. When EKF works under above conditions, the solution to SLAM is sufficiently closer to the Bayesian solution and can be considered as an optimal solution. In case of EKF-SLAM, sensor and motion models are linearized and present noises must have Gaussian probability distributions. The process noise and the observation noise are considered independent, white, and with zero-mean. The distribution of error due to noise is considered uni-modal. An overview of KF and EKF can be accomplished from [16, 20, 40, 41].

3.2 Noise Characteristics in EKF-SLAM

As discussed in chapter one, there is always some noise around the sensing devices that is corrupting the process of a mobile robot navigation [47]. If EKF is used to estimate the path of a robot and locations of landmarks in the environment, the corrupting noise must have specific characteristics. Noise that is independent, white and zero mean, is a good candidate to

be matched with the EKF-SLAM algorithm. Furthermore, this noise needs to be Gaussian to make the best estimator out of the EKF-SLAM algorithm.

3.2.1 Independency

The independency assumption makes the computation in the filter much easier. Generally, it is fair to assume that the system noise and the measurement noise are mutually independent; this means that the amount of noise at each model has no influence on the other one.

3.2.2 White Noise

The assumption of noise being white makes the computation in the filter very easy. White noise is the type of noise that has same power at all frequencies in the spectra and is completely uncorrelated with itself at any time but the current times.

3.2.3 Zero Mean

Zero mean assumption implies that the error in system and measurements are random. Noise can be classified into systematic noise and non-systematic or random noise. Systematic noise is a noise that constantly corrupts the system state or measurements in a certain way. It is biased noise often caused by inaccurate parameters. Random noise is a type of noise that is

not systematic in that way. Random noise is sometimes positive and sometimes negative but in the case of zero mean on average zero.

3.3 Gaussian Assumption

3.3.1 Noise Gaussian

The Gaussian assumption of noise usually makes the computation of a filter more tractable. This assumption states that the amount of noise can be modeled as a bell-shaped curve. This assumption is justified by supposing that motion and measurement noise are often caused by multiple small noise sources. No matter how the individual noise sources are distributed, the sum of all these independent sources will be Gaussian distributed [42]. Another reason for which Gaussian noise makes the computation of filter very convenient is that only the first and second order statistics of noise characteristics, mean and variance, are known. Many measurement devices provide only a nominal value of the measurement.

3.3.2 Motion Gaussian

The state of the system consists of the position and orientation of the vehicle and the position of landmarks in map of the environment. As was described before, there is always noise in the real world which affects the performance of the robot motion and if it is ignored, the system comes up with a significant amount of uncertainty [5, 47]. There is a specific way of

describing motion noise to translate it for EKF algorithm. To achieve such a representation of a random noise, some assumptions should be considered in advance. With the zero-mean and Gaussian distribution assumptions for the motion noise, the distribution of noise can be expressed as follows:

$$\mathbf{w}_k \sim N(0, \mathbf{Q}_k) \tag{3.1}$$

Where N (0 , $\mathbf{Q}_k$) denotes the Gaussian function with zero-mean and the system noise covariance matrix $\mathbf{Q}_k$ at time step k. The system noise covariance matrix $\mathbf{Q}_k$ can be described as follows:

$$\mathbf{Q}_k = \mathrm{E}[(\mathbf{w}_k)(\mathbf{w}_k)^{\mathrm{T}}] \tag{3.2}$$

The main diagonal of the covariance matrix $\mathbf{Q}_k$ contains the variance in the state of the system. If the state of the system is expressed by vector $\mathbf{x}_k$,

3.3.3 Observation Gaussian

Similar to motion model, sensors for the observation are always subject to noise. This noise is considered random, independent, white, and with zero-mean for EKF-SLAM algorithm. In the same way of motion model, sensor noise can be expressed as

$$\mathbf{v}_k \sim N(0, \mathbf{R}_k) \tag{3.3}$$

Just like motion noise, $N(0, \mathbf{R}_k)$ describes the Gaussian function. This function has zero-mean with the system noise covariance matrix $\mathbf{R}_k$ at time step k where $\mathbf{R}_k$ can be described as follows:

$$\mathbf{R}_k = \mathrm{E}[(\mathbf{v}_k)(\mathbf{v}_k)^{\mathrm{T}}] \tag{3.4}$$

The main diagonal of the covariance matrix $\mathbf{R}_k$ contains the variance in the observation vector variables $\mathbf{v}_k$.

3.4 Non-linearity of the System

Most dynamic systems in the real world applications are non-linear. As was described before, Kalman Filters can not deal with any non-linearity and since EKF-SLAM algorithm includes non-linear models, we need to describe motion and sensor models in terms of non-linear functions.

3.4.1 Non-linearity of Motion Model

The evolution of motion of a dynamic system can be described in form of a probabilistic model. An adequate non-linear model to describe a real noisy dynamic system is as follows:

$$P(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_k) \Leftrightarrow \mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k \tag{3.5}$$

Where $f(.)$ is a non-linear system function which relates the state of the system in the previous time step $k-1$ to the current time step k. matrix, $\mathbf{u}_k$ is the vector of control actions, $\mathbf{w}_k$ is the system noise with the covariance matrix $\mathbf{Q}_k$, and as discussed before $\mathbf{w}_k$ is a white noise, and with zero-mean Gaussian distributed.

3.4.2 Non-linearity of Sensor Model

The sensor model describes how the observations are related to the states of the system. The EKF-SLAM algorithm needs a sensor model to update the state estimation using the observation readings. If, the sensor is described by a model that given the true state of the system describes what the observation data can be, then this model can compare the real observation with the observation given by the model to correct the state estimation. Due to non-linearity of real systems, the probability density function of sensor model is denoted in general form as

$$P(\mathbf{z}_k \mid \mathbf{x}_k) \Leftrightarrow \mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k \tag{3.6}$$

Where $h(.)$ is a non-linear observation function which relates the state of the system $\mathbf{x}_k$ to the measurement from a range sensor. v_k is the observation noise highly correlated with the motion noise which in the same way of motion noise is considered Gaussian, white, independent and with zero mean. Matrix R_k is the covariance matrix of the observation noise.

3.5 EKF-SLAM Algorithm

With the definition of the standard EKF the estimation of the state and its main covariance matrix of the posterior belief of $Bel^{+}(\mathbf{x}_k) = P(\mathbf{x}_k \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0)$ can be computed as

$$\hat{\mathbf{x}}_k^{+} = \mathrm{E}[\mathbf{x}_k \mid \mathbf{Z}_k] \tag{3.7}$$

$$\mathbf{P}_k^{+} = \begin{bmatrix} \mathbf{P}^{+}_{\mathbf{x}_k\mathbf{x}_k} & \mathbf{P}^{+}_{\mathbf{x}_k\mathbf{m}_k} \\ \mathbf{P}^{+T}_{\mathbf{x}_k\mathbf{m}_k} & \mathbf{P}^{+}_{\mathbf{m}_k\mathbf{m}_k} \end{bmatrix} = \mathrm{E}\left[(\mathbf{x}_k - \hat{\mathbf{x}}_k^{+})(\mathbf{x}_k - \hat{\mathbf{x}}_k^{+})^T \mid \mathbf{Z}_k\right] \tag{3.8}$$

To estimate the state of a non-linear system with non-linear observations subject to Gaussian noise, EKF provides a recursive estimate of both state and covariance matrix. There are three steps for EKF to fulfill the task: *prediction*, *observation* and *update*.

3.5.1 Prediction Step

At time step k, the EKF propagates the state and uncertainty of the system at the previous time step. At this step the algorithm first generates a prediction for the state estimate using equation (3.9), following the state prediction, the predicted observation relative to the i^{th} landmark is done using equation (3.10) and then the state estimate covariance prediction is computed using equation (3.11).

System Model $$\hat{\mathbf{x}}_k^- = \Phi_k \hat{\mathbf{x}}_{k-1}^+ + \mathbf{u}_k \quad (3.9)$$

Sensor Model $$\hat{\mathbf{z}}_k^- = h(\hat{\mathbf{x}}_k^-) \quad (3.10)$$

$$\mathbf{P}_k^- = \Phi_k \mathbf{P}_{k-1}^+ \Phi_k^T + \mathbf{Q}_{k-1} \quad (3.11)$$

The Jacobian matrix Φ_k contains the partial derivatives of system function $f(.)$ with respect to state $\mathbf{x}$ evaluated at the posterior state estimate $\hat{\mathbf{x}}_{k-1}^+$ in time step *k-1*.

$$\Phi_k = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\hat{\mathbf{x}}_{k-1}^-} \quad (3.12)$$

3.5.2 Observation Step

After the prediction step, an observation $\mathbf{z}_k$ of i^{th} landmark is made according to equation (3.6). The difference between the observation $\mathbf{z}_k$ and the prediction observation $\hat{\mathbf{z}}_k^-$ is called the innovation or the residual $\tilde{\mathbf{z}}_k$. The innovation denotes the difference between the predicted observation and the real observation at time step *k*.

$$\tilde{\mathbf{z}}_k = \mathbf{z}_k - \hat{\mathbf{z}}_k^- = \mathbf{z}_k - h(\hat{\mathbf{x}}_k^-) \quad (3.13)$$

If the innovation is negligible (almost zero), the predicted measurement is considered the same as real measurement in equation (3.6). This means that the measurement of the state

estimate at time step k, was very close to the true state in time step k. The innovation covariance matrix is then described as:

$$\tilde{\mathbf{Z}}_k = \Psi_k \mathbf{P}_k^- \Psi_k^T + \mathbf{R}_k \tag{3.14}$$

Ψ_k is the Jacobian matrix with partial derivatives of the observation function $h(.)$ with respect to the state $\mathbf{x}_k$ evaluated at the prior state estimate $\hat{\mathbf{x}}_k^-$.

$$\Psi_k = \frac{\partial h(\mathbf{x})}{\partial \mathbf{x}}\bigg|_{\mathbf{x}=\hat{\mathbf{x}}_k^-} \tag{3.15}$$

3.5.3 Update Step

This step is called the *observation update step* or *correction step* or simply *update step*. The following correction equations correct the most recent belief of the system $Bel^+(\mathbf{x}_k)$ as described in equation (2.7) from chapter 2.

This step is the final state estimate that the algorithm needs to be completed and is very important to the SLAM problem solution. The equations are as follows:

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k \tilde{\mathbf{z}}_k \tag{3.16}$$

$$\mathbf{P}_k^+ = \mathbf{P}_k^- - \mathbf{K}_k (\Psi_k \mathbf{P}_k^- \Psi_k^T + \mathbf{R}_k) \mathbf{K}_k^T \tag{3.17}$$

The factor $\mathbf{K}_k$ is called *Kalman filter gain* which determines the extent that the innovation should be taken into account in the system posterior state estimate. Kalman gain determines this extent by looking at the relative uncertainty between the prior state estimate and the observation innovation [40]. Figure 3.1 shows the EKF algorithm.

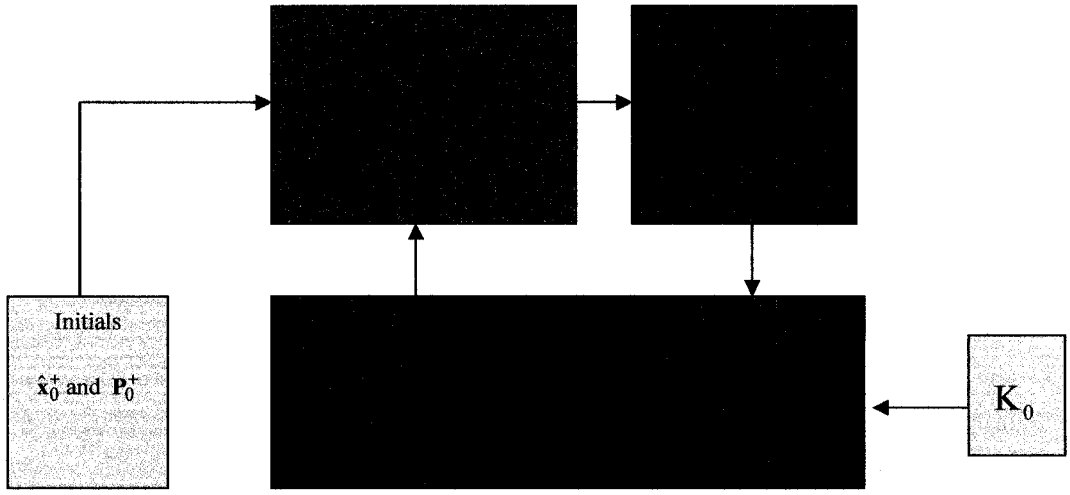
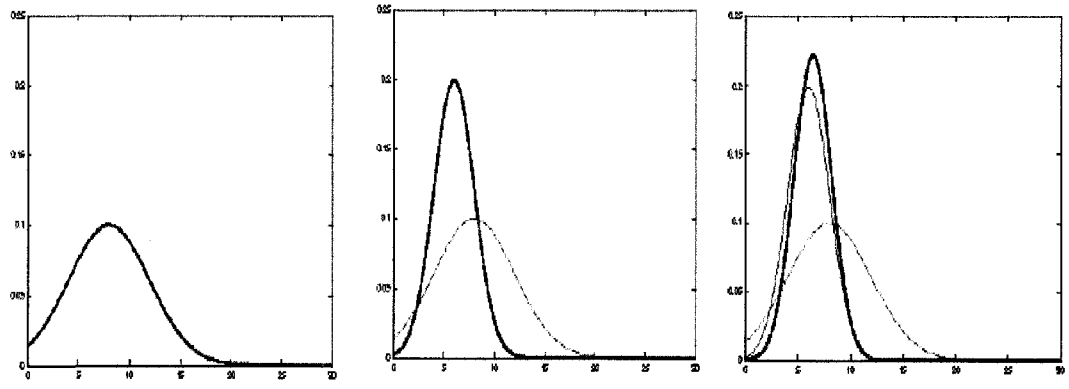


Figure 3.1: The EKF algorithm as a classical solution for SLAM problem when the system is represented with Gaussian implication. $\hat{x}_k^-$ *is the prior state of the robot,* $\hat{x}_k^+$ *indicates the posterior state of the robot, and* $\mathbf{K}_k$ *is the Kalman gain.*

To compare the prior state estimate uncertainty in the state space with the innovation uncertainty in the observation space, the Kalman gain converts the uncertainty in the observation space to the state space by means of the matrix Ψ^T. The Kalman gain is described as follows:

$$\mathbf{K}_k = \mathbf{P}_k^- \Psi_k^T (\Psi_k \mathbf{P}_k^- \Psi_k^T + \mathbf{R}_k)^{-1} \tag{3.18}$$

Figure 3.2 illustrates a posterior belief of the location of a landmark (in x direction) following by prior belief and observation steps. After incorporating the new observation, the EKF optimizes the prior belief of the landmark location. The posterior belief is shown by Figure (3.2-c). Unfortunately, the above restricted assumptions are not applicable in so many situations in the real world. The Gaussian implication of system and sensor noise is not a proper way to get into a reasonable solution for so many practical cases. Furthermore, EKF algorithm suffers form two basic problems that make it unreliable and hard to utilize for so many applications. Followings are two major issues regarding EKF-SLAM algorithm that diverge the filter and prevent it to work properly. These issues are the *computational complexity*, and the *data association*.



(a) *(b)* *(c)*

Figure 3.2: Prior belief, observation, and posterior belief of location of a landmark in x direction [29]. (a) Prior belief (Prediction Step) of location of a landmark. (b) An observation of location of a landmark. c) Posterior belief of location of a landmark after incorporating the observation.

3.6 Computational Complexity

One major issue of EKF-SLAM solution is the computational complexity which is in the heart of the algorithm. The more landmarks are observed and used to build a map, the more time the algorithm consumes. Moreover, EKF algorithm demands a huge memory while the landmarks are added to the map. In fact the Gaussian representation of the system and noise employed by EKF algorithm makes complexity of the computation quadratic. This is due to the fact that for a two-dimensional environment (which is considerable in most SLAM cases), the system covariance matrix **P**, contains $2M+3$ by $2M+3$ arrays. The number 3 is for the variables of the pose of the robot including (x,y) coordinates and Θ the orientation and M is for the number of landmarks in the map. Therefore a huge memory is needed for a quadratic grown covariance matrix as of M^2. Leonard and Durrant-Whyte [10] have shown that the EKF-SLAM algorithm is an inappropriate solution for a large scaled environment with a significant amount of landmarks even though the noise is considered Gaussian. For more details on this issue one can accomplish a deep review in [10, 11, 18].

For more clarification, figure 3.3 depicts the system covariance matrix **P**. this matrix representation is called the absolute representation of the main matrix of the system [9]. It contains the covariance in the robot position, the covariance on the landmarks, the covariance between robot's position and landmarks, and finally it contains the covariance between landmarks mutually. The first cell A_{11} contains the covariance on the robot's position which is a 3×3 matrix (x, y, Θ), A_{22} is the covariance on the first landmark which is 2x2 since landmarks are considered static and do not need any orientation. This continues down to A_{MM} which is the covariance of last observed landmark. The cell A_{21} contains the covariance

between the robot's state and the first landmark. The cell A_{12} contains the covariance between the first landmark and the robot's state. A_{12} can be deduced from A_{21} by transposing the sub-matrix A_{21}. A_{M2} contains the covariance between the last landmark and the first one, while A_{2M} contains the covariance between the first landmark and the last one, which again can be deduced by transposing A_{M2}. To see how the data accumulates in the main covariance matrix while the robot is traveling through the environment, a simple situation is exemplified in the next section.

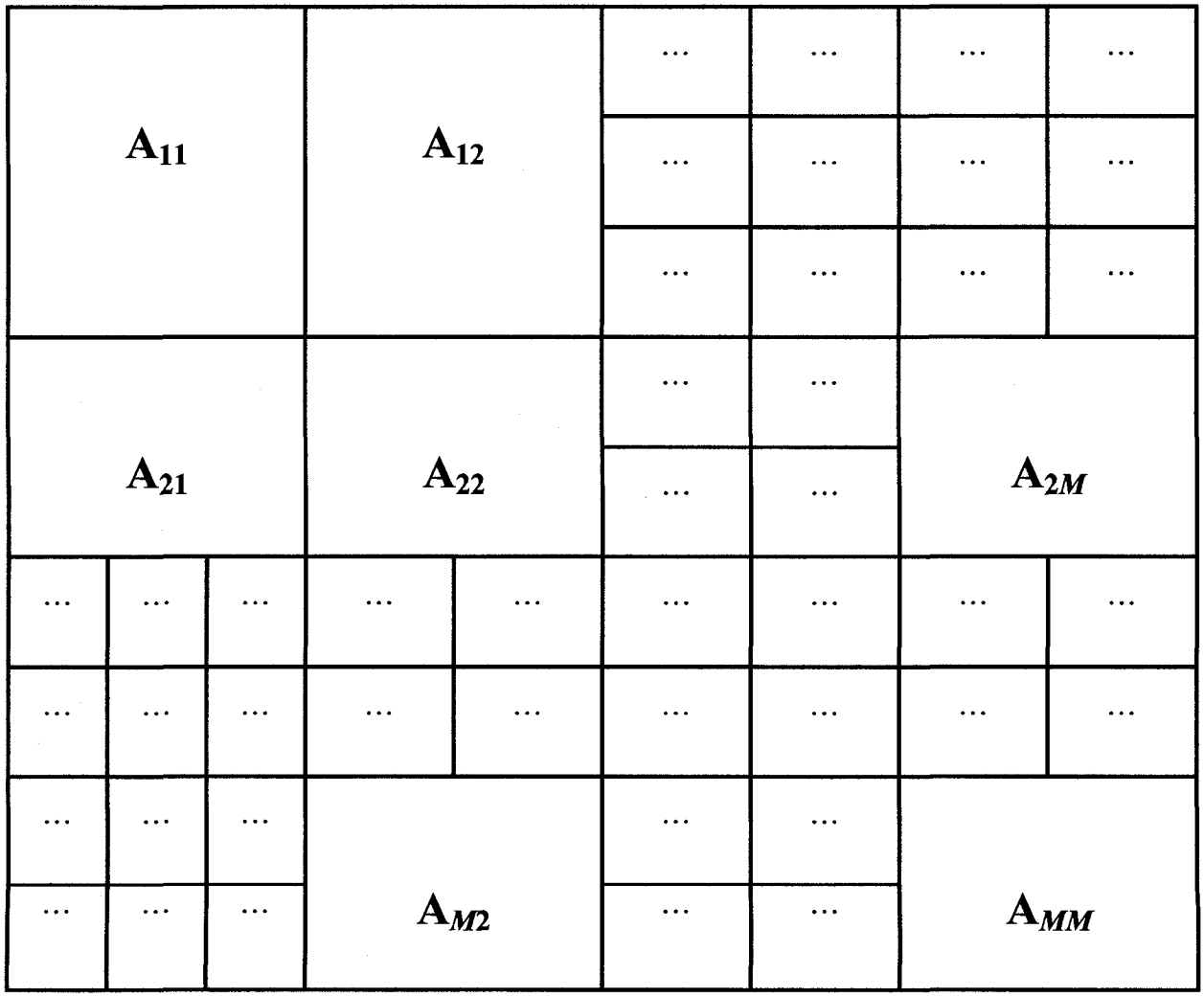


Figure 3.3: Absolute representation of covariance matrix of the system ***P****. This matrix contains 2M+3 by 2M+3 arrays of matrices.*

3.6.1 An Example of the Main Covariance Matrix of the System

Suppose that an autonomous robot is in the initial position P_1 with respect to a reference coordinate system as illustrated in figure 3.4. While the robot is moving to the position P_2, observes the landmark m_1. So far the only landmark the robot has observed is only landmark m_1. Therefore, the only information which is incorporated to the map will be the coordinates of the first landmark with respect to the robot's coordinate system and the coordinates and heading of the robot with respect to the reference coordinate system. The main covariance matrix of the filter is then reperesented as follows:

$$\mathbf{P} = \begin{bmatrix} x_{p1}^{R} & 0 & 0 & x_{p1}^{R} - x^{m1} & 0 \\ 0 & y_{p1}^{R} & 0 & 0 & y_{p1}^{R} - y^{m1} \\ 0 & 0 & \theta_{p1}^{R} & 0 & 0 \\ x_{p1}^{R} - x^{m1} & 0 & 0 & x^{m1} & 0 \\ 0 & y_{p1}^{R} - y^{m1} & 0 & 0 & y^{m1} \end{bmatrix} \tag{3.19}$$

As we can see this matrix is [2(1)+3] × [2(1)+3] = 5×5. The main diagonal of the matrix is the coordinate and heading of the robot and coordinate of the first landmark with respect to the global coordinate system. The off-diagonal elements are the coordinates of the landmark with respect to the robots reference system. Now suppose that the robot reaches to point P_2 and observes the landmarks m_1 and m_2. This time the main covariance matrix has more elements than the previous time step. This matrix is a [2(2)+3] × [2(2)+3] = 7×7 matrix which contains covariance in the robot position, the covariance on the landmarks, the covariance between

robot's position and landmarks, and the covariance between any pairs of landmarks as follows:

$$\mathbf{P}=\begin{bmatrix} x_{p2}^{R} & 0 & 0 & x_{p2}^{R}-x^{m1} & 0 & x_{p2}^{R}-x^{m2} & 0 \\ 0 & y_{p2}^{R} & 0 & 0 & y_{p2}^{R}-y^{m1} & 0 & y_{p2}^{R}-y^{m2} \\ 0 & 0 & \theta_{p2}^{R} & 0 & 0 & 0 & 0 \\ x_{p2}^{R}-x^{m1} & 0 & 0 & x^{m1} & 0 & x^{m1}-x^{m2} & 0 \\ 0 & y_{p2}^{R}-x^{m1} & 0 & 0 & y^{m1} & 0 & y^{m1}-y^{m2} \\ x_{p2}^{R}-x^{m2} & 0 & 0 & x^{m1}-x^{m2} & 0 & x^{m2} & 0 \\ 0 & y_{p2}^{R}-y^{m2} & 0 & 0 & y^{m1}-y^{m2} & 0 & y^{m2} \end{bmatrix} \tag{3.20}$$

As we can see the main covariance matrix grows as the robot observes more landmarks. If there are unlimited landmarks in the environment, this matrix grows unbounded and eventually for M observed landmarks by the robot the covariance matrix $\mathbf{P}$ becomes so large that requires a huge amount of memory and computer power.

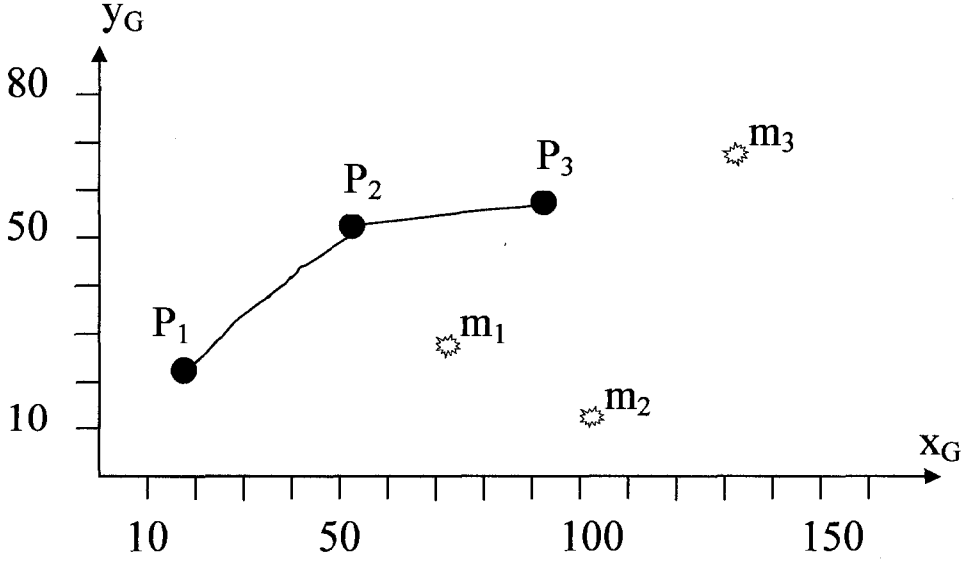


Figure 3.4: A simple example for three landmarks

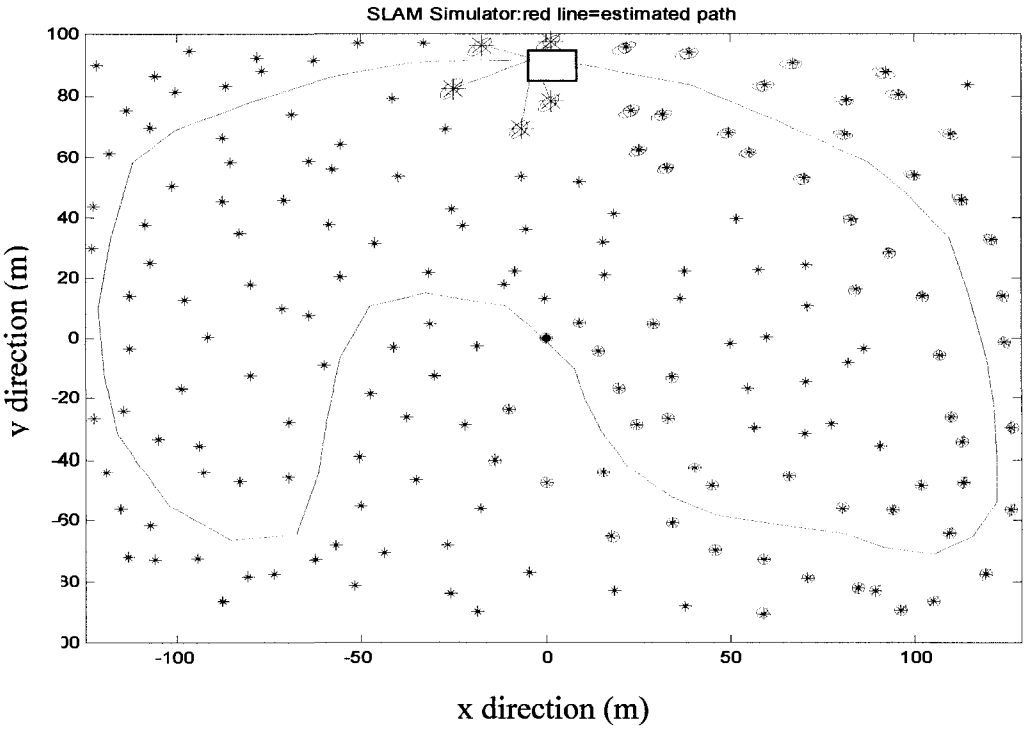
A simulated robot path estimate in two different positions in a closing loop case has been shown in figure 3.5; when the robot is in the middle of the loop, and when it has already completed the loop and is in the middle of the second round. The simulated covariance matrices of both situations are shown in figure 3.6. As we can see the EKF approximate the SLAM posterior as a high dimensional Gaussian over all features in the map and the robot pose. The off diagonal elements of the covariance matrix of this multi variant Goussian represent the correlation between all pairs of state valuables. As a result the EKF is expressive enough to represent the correlated errors that characterize the SLAM problem. This correlation between all pairs of the system grows as the robot keeps moving through the environment and observes landmarks and if there are unlimited landmarks, the main covariance matrix of the system grows unbounded which is not the case in the simulated loop closing example. The darker the matrix elements are, the higher the correlation between the state variables corresponding to the element's row and column is.

In case of the second round, the map appears to be completely correlated and the cross correlation between different constellations is very much larger than the first case where the robot has not completed the loop yet. These correlations are updated but all the information stays in the memory used for the computation. Consequently, for a large number of landmarks, the covariance matrix will become extremely large so that even the most powerful computers can not handle the computation. This case is not practical in real world especially in outdoor applications that the number of landmarks is usually large. Consider the case if an autonomous robot is sent to another planet. In these situations, determining number of landmarks is not a possible task to do before starting the robot mission on surface of the planet.

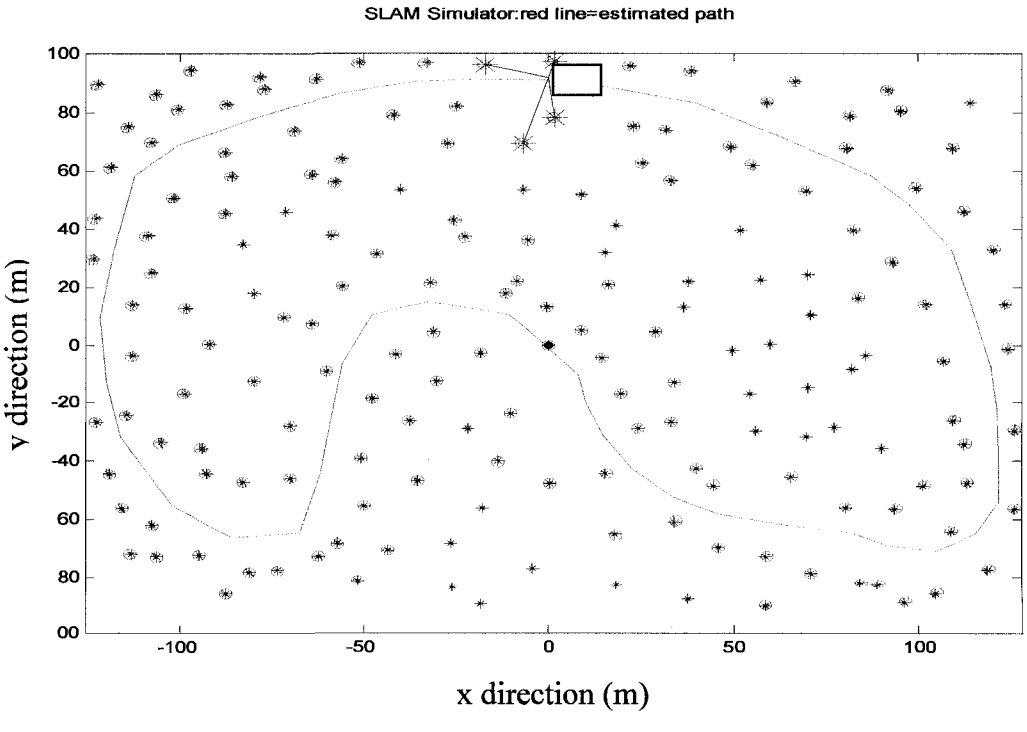
3.7 Single-Hypothesis Data Association

The mapping problem which is solvable by landmarks observations is always influenced by data association. Using the maximum likelihood rule, every observation is assigned to a landmark. Before the data is fused to the map, a new observation of landmark is assigned to it. If for any specific reason, this observation is wrong, which means the probability of the observation is not high enough, a new landmark with no accuracy is added to the map. Since EKF algorithm does not include data association uncertainty, wrong information is added to the map and EKF can not revise this wrong data. If more inaccurate information is fused to the map at the next time steps, the EKF algorithm diverges. Closing the loop will be a major problem specifically when the robot goes back to the first landmark after accomplishing one loop. In figure 3.4 if the number of landmarks increases 5 times, the robot faces a big problem for the loop closing. For more information on data association one can refer to [18].

There have been several attempts to solve the above issues and bring the EKF-SLAM to a better performance but still under Gaussian assumption. These attempts are *scaling SLAM algorithms*, and *robust data association* [11, 48]. Likewise, *Montemerlo* [11] has proposed an alternative way of dealing with SLAM problem that is based on *Monte Carlo Localization* (MCL) by implementation of *Rao-Blackwellised particle filter* (RBPF). This Alternative solution, called *Fast-SLAM,* contains the same promising results of EKF-SLAM with non-Gaussian implication of the system. In chapter 5 we will discuss the FAST-SLAM algorithm in detail and will investigate the advantages and disadvantages of this method.

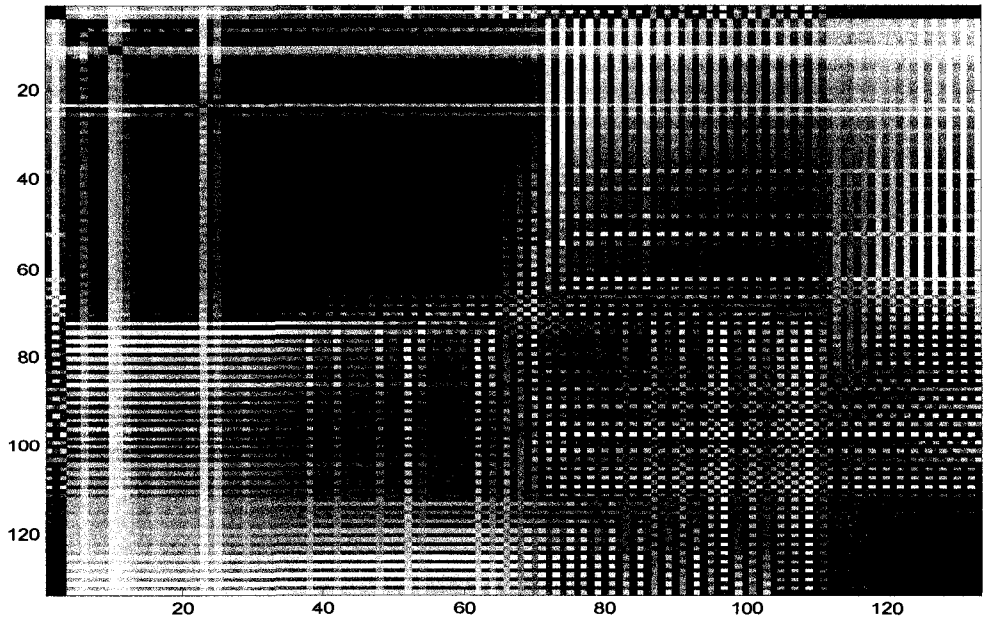


(a)

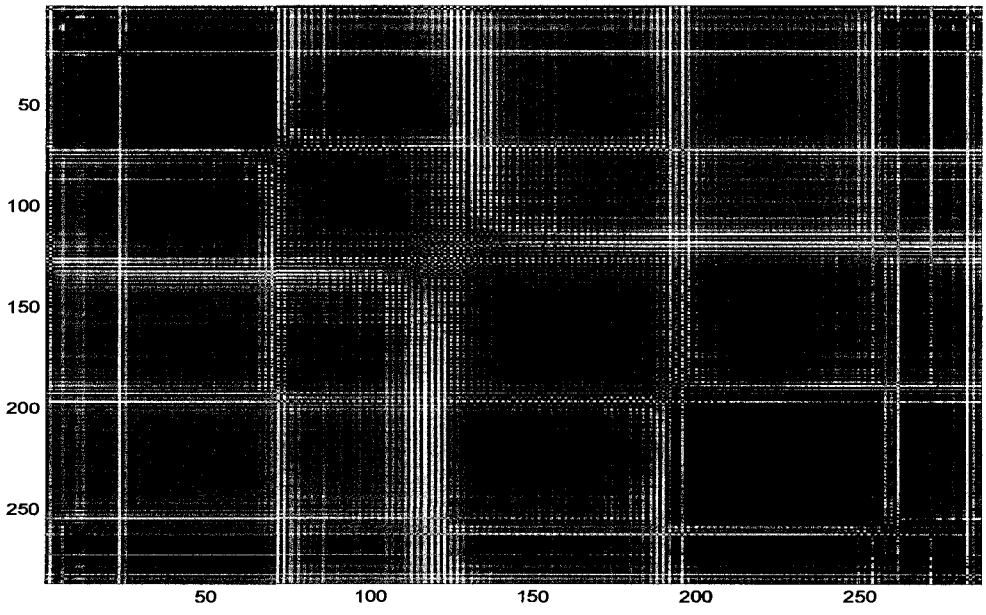


(b)

Figure 3.5: simulated robot path (a) loop not completed (b) loop completed



(a)



(b)

Figure 3.6: Simulated covariance matrix (a) loop not completed (b) loop completed

3.8 Summary

The Extended Kalman Filter (EKF) is a recursive data processing algorithm that has successfully been used in some applications. The EKF estimates the state of a noisy system using noisy measurements. More precisely, it calculates the conditional probability distribution over the space of states given measurements, the belief. It does this in a prediction-correction way, where it first predicts the state of the system based on system dynamics and then corrects its prediction based on measurements.

The EKF makes a number of assumptions on the system, measurements and different noises that are involved in the estimation problem. The EKF assumes that the system and measurements are adequately modeled by a non-linear dynamic system which has been linearized, and that noises are independent, white, Gaussian and with zero mean. To deal with the complexity in the sensor and action models, the models must be assumed Gaussian distributed. Moreover, this filter assumes that the initial state of the system is also independent and Gaussian distributed.

EKF for small scale environment and Gaussian conditions is an appropriate algorithm for SLAM problem nonetheless in case of outdoor applications where number of landmarks becomes extremely large it suffers from two major problems; computation complexity and single hypothesis data association. These two issues makes the usage of memory, and computational power and very costly while in case of loop closing the filter diverges and may not do the estimation correctly.

Chapter 4

A Realistic Model of an Outdoor Mobile Robot

4.1 A Realistic Outdoor Mobile Robot

As we discussed in previous chapters, there is a need of motion and measurement models for a filter to get them fit into the algorithm so that the algorithm can make the best estimation out of relative and absolute measurements. Depending on the application area, these models may differ. Though, we should be very careful of deriving a model since a very perfect model might result in slow evaluation of the system. In this chapter we will introduce a standard outdoor car like robot with a mounted range/bearing sensor. Afterward, we will develop formulas that develop different aspects of a perfect navigator robot, in particular motion and sensing aspects. In this and the following chapters we will use these models in both EKF and RBPF instances. All simulation results will be according to equations that fit with the specifications of the outdoor mobile robot that will be represented in this chapter. Such a standard model of an outdoor autonomous robot with wheels [9] is shown in figure 4.1.

4.1.1 Wheeled Mobile Robot System

As we discussed in chapter 1, mobile robots have a system to be able to have a motion through an environment. In most cases, the locomotion system consists of wheels rather than legs. In nowadays applications, most wheeled autonomous robots look like regular vehicles

with four or six wheels. The reason is that, vehicle mobile robots with wheels are less complex in terms of derivation of motion models and very much suitable for navigational purposes.

4.1.2 Encoder Reading Translation

The robotic vehicle is equipped with optical *wheel encoders* [7] as demonstrated in figure 4.2. By means of wheel encoders the number of wheel rotation of a specific wheel can be calculated. If the diameter of the wheel is known in advance, the displacement of the robot can be estimated. As was described in chapter 1, this displacement is usually called dead reckoning or *encoder sensor reading* [49]. The velocity of the vehicle measured by the encoder reader is denoted as V^e. This velocity is used to translate the velocity of the middle back axel of the robot V^R. The equation that is used for this translation purpose is described as

$$V^R = \frac{V^e}{1 - \tan(\alpha) \cdot \frac{W}{\ell}} \tag{4.1}$$

4.1.3 External Range/Bearing Sensor

The external range/bearing sensor [32] is installed in front of the robotic vehicle. This is the sensor which was modeled by equation (3.6) in chapter 3. This sensor returns range and bearing information to landmark $\mathbf{m}_i$. The position of the i^{th} landmark, $z_{k,i} = (d\ ,\ \beta)$, is being

read. d is the distance of on-board sensor from the landmark and β is the sensor bearing measured with respect to coordinate system of the back axel of the robotic vehicle (x^C , y^C). The steering angle of the vehicle is noted by α. In figures 4.1 and 4.2 all parameters and variables of the system have been shown.

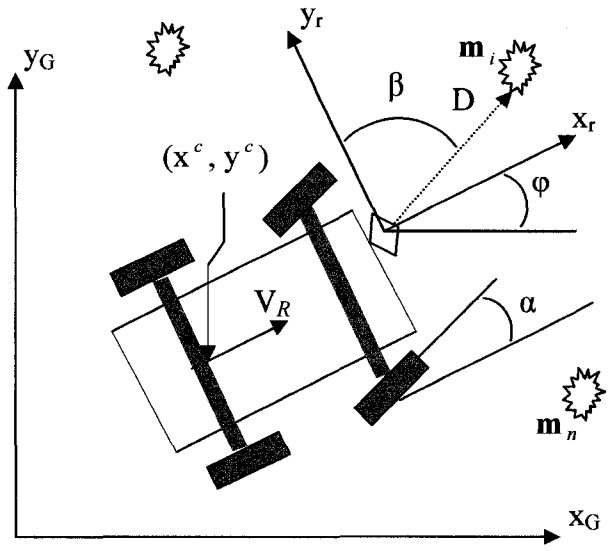


Figure 4.1: The vehicle coordinate system and the position of i^{th} landmark with respect to the robot coordinate system.

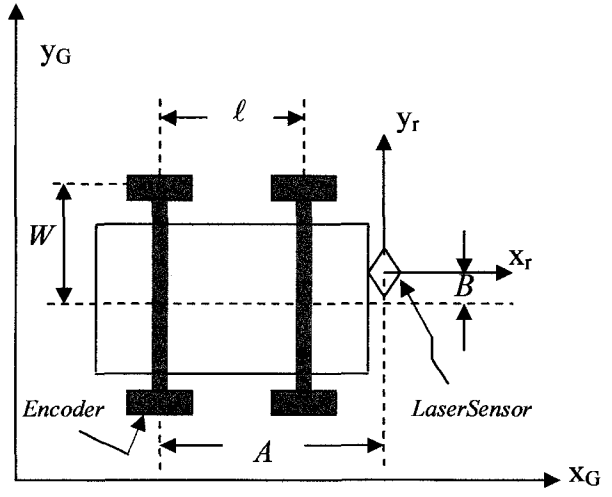


Figure 4.2: Kinematics parameters of the standard robotic vehicle

4.2 Linearization

In practice, the prediction model for the robot trajectory and the model that relates observation to the states are not linear. The SLAM can be formulated if motion models can be linearized [26]. For the linearization purpose, the Jacobian of BOTH models can be used to propagate the main covariance matrix of the system **P**.

4.2.1 Motion Model Linearization

Here, first it should be described how the model of the vehicle can be achieved. In theses cases usually the trajectory of one specific point of the robot body is analyzed [32]. In the robot showed in figure 4.1, the trajectory of the middle of the back axel is used to show the trajectory of center of the laser sensor and is given by equation (4.2). For clarity of presentation $\mathbf{X}_k^R$ (path of the robot) is considered instead of $\mathbf{X}_k^S$ (path of the sensor). In fact the state of the laser sensor mounted on the robot is the matter of importance but to simplify the signs and adapt it with the defined sets in chapter 2, character *S* is replaced with character *R*.

$$\mathrm{V}_R = \begin{bmatrix} \dot{\mathrm{x}}^S \\ \dot{\mathrm{y}}^S \\ \dot{\varphi}^S \end{bmatrix} = \begin{bmatrix} \dot{\mathrm{x}}^R \\ \dot{\mathrm{y}}^R \\ \dot{\varphi}^R \end{bmatrix} = \begin{bmatrix} \mathrm{V}^R.\cos(\varphi) \\ \mathrm{V}^R.\sin(\varphi) \\ \dfrac{\mathrm{V}^R}{\ell}.\tan(\alpha) \end{bmatrix} + \mathrm{w}_k^R \tag{4.2}$$

$$\mathbf{X}_k^S = \mathbf{X}_k^R = \{\mathbf{x}_0^R, \mathbf{x}_1^R, \ldots, \mathbf{x}_k^R\} = \{\mathbf{X}_{k-1}^R, \mathbf{x}_k^R\} \tag{4.3}$$

The translation of the middle point of the back axel with respect to the global coordinate system (x_G , y_G)is denoted as

$$L^S = L^C + A.\mathbf{t}_\varphi + B.\mathbf{t}_{\varphi+\pi/2} \tag{4.4}$$

Where L^S is the location of center of the laser sensor with respect to the global coordinate system and L^C is the location of middle point of the back axel of the robot with respect to the global coordinate system.

The transformation vector is defined by the orientation of the robot as

$$\mathbf{t}_\varphi = (\cos(\varphi) , \sin(\varphi)) \tag{4.5}$$

The location of the sensor can be represented as following scalar equations:

$$x^R = x^c + A \cdot \cos(\varphi) + B \cdot \cos(\varphi + \frac{\pi}{2}) \tag{4.6}$$

$$y^R = y^c + A \cdot \sin(\varphi) + B \cdot \sin(\varphi + \frac{\pi}{2}) \tag{4.7}$$

Where (x^S, y^S) is the coordinate of the laser sensor and (x^R, y^R) is the coordinate of the robot with respect to the global coordinate system. From now on, in this paper and in the MATLAB code, (x^S, y^S) will be considered as coordinate of the robotic vehicle. The full state representation can be written as:

$$\begin{bmatrix} \dot{x}^R \\ \dot{y}^R \\ \dot{\varphi}^R \end{bmatrix} = \begin{bmatrix} V^R.\cos(\varphi) - \frac{V^R}{\ell} \cdot N_1 \\ V^R.\sin(\varphi) + \frac{V^R}{\ell} \cdot N_2 \\ \frac{V^R}{\ell}.\tan(\alpha) \end{bmatrix} + w_k^R \tag{4.8}$$

Where $N_1 = \left(A\cos(\varphi) + B\sin(\varphi)\right)\cdot\tan(\alpha)$, and $N_2 = \left(A\cos(\varphi) - B\sin(\varphi)\right)\cdot\tan(\alpha)$

The whole state can be modeled as following

$$\mathbf{x}_k^R \begin{bmatrix} x_k^R \\ y_k^R \\ \varphi_k^R \end{bmatrix} = \begin{bmatrix} x_{k-1}^R + \Theta_{k-1}^x - \frac{V_{k-1}^R}{\ell}\Omega_{k-1}^x \\ y_{k-1}^R + \Theta_{k-1}^y - \frac{V_{k-1}^R}{\ell}\Omega_{k-1}^y \\ \frac{V_{k-1}^R}{\ell}.\tan(\alpha_{k-1}) \end{bmatrix} + w_k^R \tag{4.9}$$

where

$$\Theta_{k-1}^x = \Delta t \cdot V_{k-1}^R \cdot \cos(\varphi_{k-1}^R)$$

$$\Omega_{k-1}^x = \left(A\sin(\varphi_{k-1}^R) + B\cos(\varphi_{k-1}^R)\right)\cdot\tan(\alpha_{k-1})$$

$$\Theta_{k-1}^y = \Delta t \cdot V_{k-1}^R \cdot \sin(\varphi_{k-1}^R)$$

$$\Omega_{k-1}^y = \left(A\cos(\varphi_{k-1}^R) - B\sin(\varphi_{k-1}^R)\right)\cdot\tan(\alpha_{k-1})$$

w_k is the motion noise and Δt is the sampling time which in this case is not considered constant.

4.2.2 Sensor Model Linearization

The equation that relates the robot state to the sensor observations is as follows:

$$\mathbf{z}_k = h(\mathbf{x}) + \mathbf{v}_k = \begin{bmatrix} \mathbf{z}_{k,i}^{D} \\ \mathbf{z}_{k,i}^{\beta} \end{bmatrix} + \mathbf{v}_k = \begin{bmatrix} \sqrt{(\mathrm{x}_k^{\mathrm{m}_i} - \mathrm{x}_k^R)^2 + (\mathrm{y}_k^{\mathrm{m}_i} - \mathrm{y}_k^R)^2} \\ \tan^{-1}\left(\dfrac{(\mathrm{y}_k^{\mathrm{m}_i} - \mathrm{y}_k^R)}{(\mathrm{x}_k^{\mathrm{m}_i} - \mathrm{x}_k^R)}\right) - \varphi_k^R + \dfrac{\pi}{2}) \end{bmatrix} + \mathbf{v}_k \tag{4.10}$$

Where $(\mathrm{x}_k^{\mathrm{m}_i}, \mathrm{y}_k^{\mathrm{m}_i})$ is the coordinate of i^{th} landmark. As all landmarks are static, which means that they do not move and at any time step k, have the same locations with respect to the global coordinate system, the following equation can be considered:

$$(\mathrm{x}_{k+1}^{\mathrm{m}_i}, \mathrm{y}_{k+1}^{\mathrm{m}_i}) = (\mathrm{x}_k^{\mathrm{m}_i}, \mathrm{y}_k^{\mathrm{m}_i}) = (\mathrm{x}^{\mathrm{m}_i}, \mathrm{y}^{\mathrm{m}_i}) \tag{4.11}$$

4.2.3 Control System with Noise

If the state of the control system of the robot includes noisy signals (which in most cases does), a noise vector $\mathbf{w}_k^u$ is added to the control vector and the complete no-linear motion model can be expressed as following

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k + \mathbf{w}_k^u) + \mathbf{w}_k^R \tag{4.12}$$

In most cases the sum of control noise and noise of the state of the robot can be added. With Jacobian of control vector the total noise of the motion is described as:

$$\mathbf{w}_k = \boldsymbol{J}_{\mathbf{u}} \mathbf{w}_k^u + \mathbf{w}_k^R \tag{4.13}$$

Where

$$\boldsymbol{J}_{\mathbf{u}} = \frac{\partial f(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}}\Big|_{\mathbf{x}=\mathbf{x}_k, \mathbf{u}=\mathbf{u}_k} \tag{4.14}$$

Now it can be seen that equation (4.15) can be equivalent with equation (3.5) from the third chapter.

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k + \mathbf{w}_k^u) + \mathbf{w}_k^R \cong f(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k \tag{4.15}$$

4.2.4 Noise Characteristics Matrices

All noise characteristics are assumed to be white, zero-mean, and independent.

$$\mathrm{E}[\mathbf{w}_k] = \mathrm{E}[\mathbf{w}_k^R] = \mathrm{E}[\mathbf{w}_k^u] = \mathrm{E}[\mathbf{v}_k] = 0 \tag{4.16}$$

$$\mathrm{E}[(\mathbf{w}_k)(\mathbf{w}_k)^{\mathrm{T}}] = \delta_{i,j} \cdot \mathbf{Q}_k \tag{4.17}$$

$$\mathrm{E}[(\mathbf{w}_k^R)(\mathbf{w}_k^R)^{\mathrm{T}}] = \delta_{i,j} \cdot \mathbf{Q}_k^R \tag{4.18}$$

$$E[(\mathbf{w}_k^{\mathbf{u}})(\mathbf{w}_k^{u})^{\mathbf{T}}] = \delta_{i,j}.\mathbf{Q}_k^{\mathbf{u}} \tag{4.19}$$

$$E[(\mathbf{v}_k)(\mathbf{v}_k)^{\mathbf{T}}] = \delta_{i,j}.\mathbf{R}_k \tag{4.20}$$

$$\delta_{i,j} = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

$$E[(\mathbf{w}_k)(\mathbf{w}_k)^{\mathbf{T}}] = \delta_{i,j}\mathbf{Q}_k = \delta_{i,j}(\boldsymbol{J}_{\mathbf{u}}.\mathbf{Q}_k^{\mathbf{u}}.\boldsymbol{J}_{\mathbf{u}}^{T} + \delta_{i,j}.\mathbf{Q}_k^{R}) \tag{4.21}$$

where

$$\mathbf{w}_k \sim N(0, \mathbf{Q}_k) \qquad \mathbf{v}_k \sim N(0, \mathbf{R}_k)$$

$$\mathbf{w}_k^{R} \sim N(0, \mathbf{Q}_k^{R}) \qquad \mathbf{w}_k^{u} \sim N(0, \mathbf{Q}_k^{\mathbf{u}})$$

Now equation (2.2) can be rewritten in form of following matrix:

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_k^{R} \\ \mathbf{m} \end{bmatrix} \tag{4.22}$$

Where

$$\mathbf{x}_k^{R} = (\mathrm{x}_k^{R}, \mathrm{y}_k^{R}, \varphi_k^{R})^{T} \in \boldsymbol{S}^3 \tag{4.23}$$

$$\mathbf{x}^{\mathbf{m}} = (\mathrm{x}^{\mathrm{m}_1}, \mathrm{y}^{\mathrm{m}_1}, \mathrm{x}^{\mathrm{m}_2}, \mathrm{y}^{\mathrm{m}_2}, \ldots, \mathrm{x}^{\mathrm{m}_M}, \mathrm{y}^{\mathrm{m}_M}) \in \boldsymbol{S}^{2M} \tag{4.24}$$

and the model can be written in form of

$$\mathbf{x}_k^R = f(\mathbf{x}_{k-1}^R, \mathbf{u}_k) + \mathbf{w}_k \tag{4.25}$$

$$\mathbf{x}^{\mathbf{m}_i} = \begin{bmatrix} \mathrm{x}^{\mathrm{m}_i} \\ \mathrm{y}^{\mathrm{m}_i} \end{bmatrix} \tag{4.26}$$

4.2.5 Jacobian Matrices

The Jacobian matrix for the function of extended system $f(\mathbf{x}_{k-1}, \mathbf{u}_k)$ is

$$\Phi_k = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\hat{\mathbf{x}}_{k-1}^-} = \begin{bmatrix} \frac{\partial f(\mathbf{x}^R)}{\partial \mathbf{x}^R}\Big|_{\mathbf{x}^R=\hat{\mathbf{x}}^R{}_{k-1}^-} & 0 \\ 0 & \mathbf{I} \end{bmatrix} \tag{4.27}$$

Where $\mathrm{I} \in \boldsymbol{S}^{2M} \times \boldsymbol{S}^{2M}$ and the Jacobian matrix of the observation function $h(\mathbf{x}_k)$ is

$$\Psi_k = \frac{\partial h(\mathbf{x})}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\hat{\mathbf{x}}_k^-} = \begin{bmatrix} \frac{\partial \mathbf{z}_i^D}{\partial \mathbf{x}} \\ \frac{\partial \mathbf{z}_i^\beta}{\partial \mathbf{x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathbf{z}_i^D}{\partial (\mathbf{x}_k^R, \mathbf{m})} \\ \frac{\partial \mathbf{z}_i^\beta}{\partial (\mathbf{x}_k^R, \mathbf{m})} \end{bmatrix} \tag{4.28}$$

Equation (4.28) always has a large number of null elements since only a few landmarks are observed and validated by the sensor at each time step. For instance if only one landmark is observed the Jacobian becomes:

$$\Psi_k = \begin{bmatrix} \frac{\Delta x}{\Delta y} & \frac{\Delta y}{\Delta^2} & 0 & 0 & \cdots & -\frac{\Delta x}{\Delta^2} & -\frac{\Delta y}{\Delta^2} & 0 & \cdots & 0 \\ -\frac{\Delta y}{\Delta^2} & \frac{\Delta x}{\Delta^2} & -1 & 0 & \cdots & \frac{\Delta y}{\Delta^2} & -\frac{\Delta x}{\Delta^2} & 0 & \cdots & 0 \end{bmatrix} \tag{4.29}$$

where

$$\Delta x = x_k^R - x_k^{m_i}$$

$$\Delta y = y_k^R - y_k^{m_i}$$

$$\Delta = \sqrt{(\Delta x)^2 + (\Delta y)^2}$$

4.3 Applying EKF for the Standard Autonomous Robot

As we have all model equations for the described standard robotic vehicle, we are now able to fit these equations into the EKF algorithm. This way, an optimal solution to SLAM can be found. The intention of this section is to show some simulation results specific to EKF-SLAM algorithm. We want to briefly analyze the result of simulations and discuss the performance of EKF-SLAM. To do this, an estimation error of each state element has been plotted and shown in figures 4.3, 4.4, and 4.5. This system is considered with Gaussian assumptions. Furthermore the system is subject to Gaussian, white, independent with zero-mean noise. Only under these certain circumstances the EKF provides an optimal solution to SLAM. Figures 4.3, 4.4, and 4.5 demonstrate the true and the estimated vehicle position errors in x and y direction and for the robot orientation. It should be noted that the true error is not normally available in a real system. Figure 4.6 shows the true locations of the robotic vehicle and landmarks and figure 4.7 shows the observations by the laser sensor. In figure 4.8 a plot

of different location estimations of landmark $\mathbf{m}_i$ is demonstrated. In figures 4.3 and 4.4 true error of the motion in both x and y directions have been plotted. While the condition has been set for the least amount of noise in different sensors of the robot, the average true error is still around 0.3 meters in both directions. Applying EKF algorithm to the SLAM problem reduces the error down to 0.07 meters in average. In figure 4.5, the value of true error of the vehicle orientation gives an average of 0.04 radians while the estimated error is less than 10% of the true orientation error after EKF algorithm is applied. This indicates that EKF-SLAM algorithm may be useful for navigation of a real outdoor mobile robot. Though, we should recall that the condition must follow Gaussian assumptions.

An estimation error, is the difference between true and estimated states. It should be noted that the variation in the estimation error seems to remain constant. The EKF knows that the uncertainty in the system will decrease the chance that the predicted state estimates equal the true states as the time goes by. Due to the structure of its algorithm, the EKF does not justify this difference and naturally tries to compensate this difference at some point. Ideally, the variance in the state estimates that the EKF computations should at all time include the true state as a possible state. The deviation that the EKF calculates indicates how certain it is that the true state lies within a certain distance from the estimated state. The EKF is about 66% sure that the the true state element lies within one deviation from the estimated elements; this is called 1-σ confidence interval. Figure 4.9 shows the innovation and innovation standard deviation along and cross the track direction with the 1-σ confidence interval. In this particular run, the error that the EKF makes, remains quite small within the 1-σ uncertainty region. The green shaded area shows 1-σ confidence interval in figure 4.9.

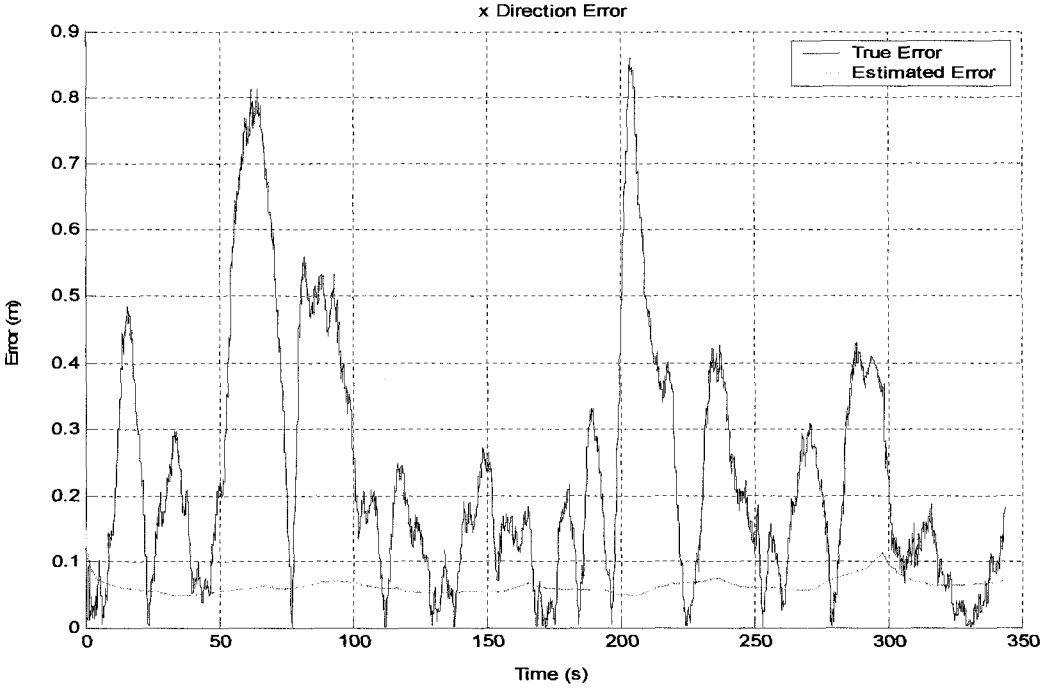


Figure 4.3: True error versus estimated error along x direction

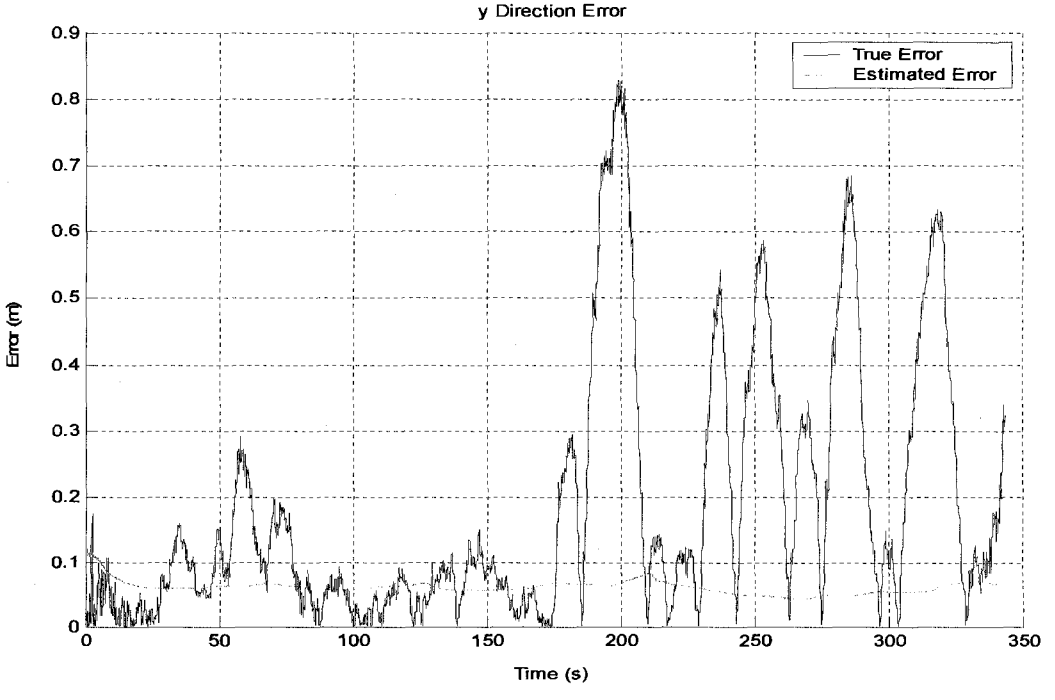


Figure 4.4: True error versus estimated error along y direction

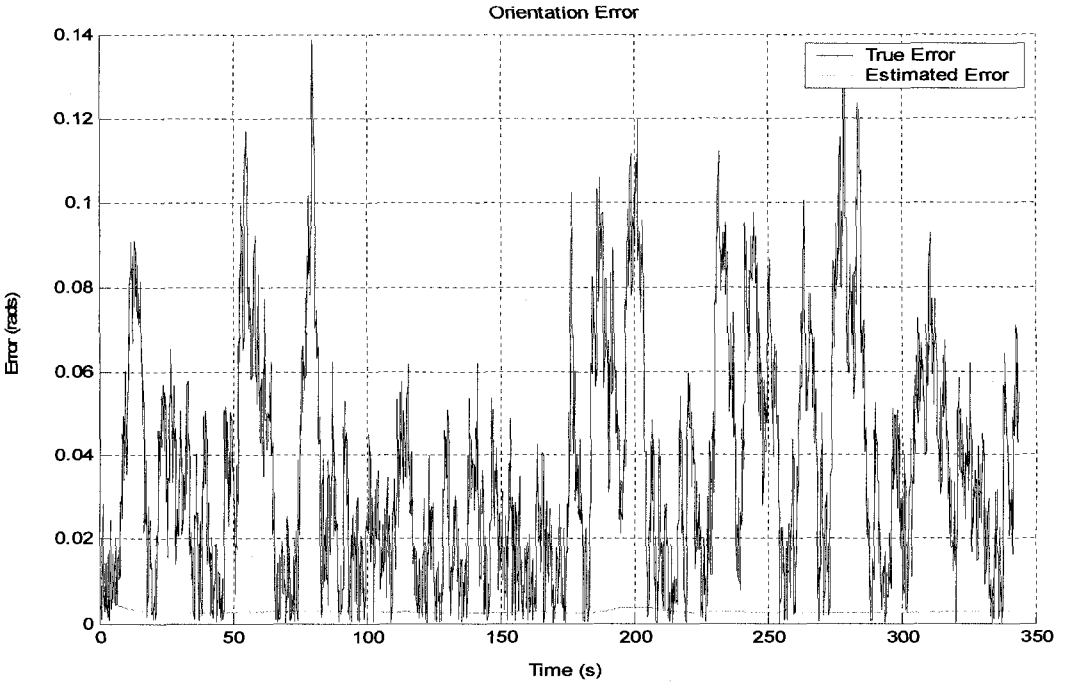


Figure 4.5: True orientation error versus estimated orientation error

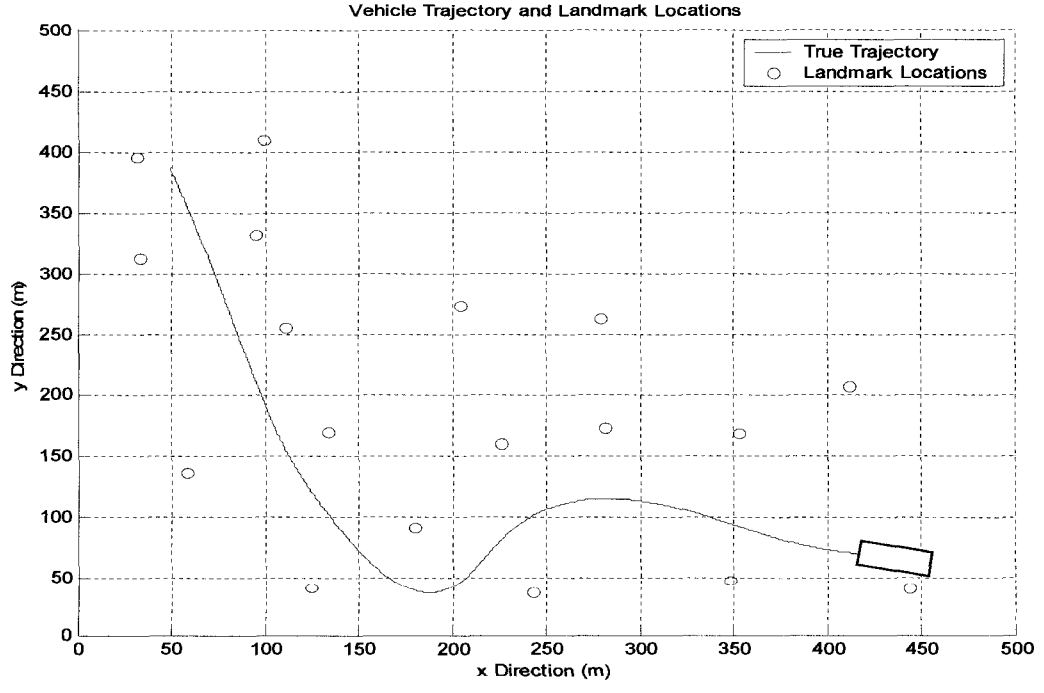


Figure 4.6: The robot is traveling in and environment among landmarks

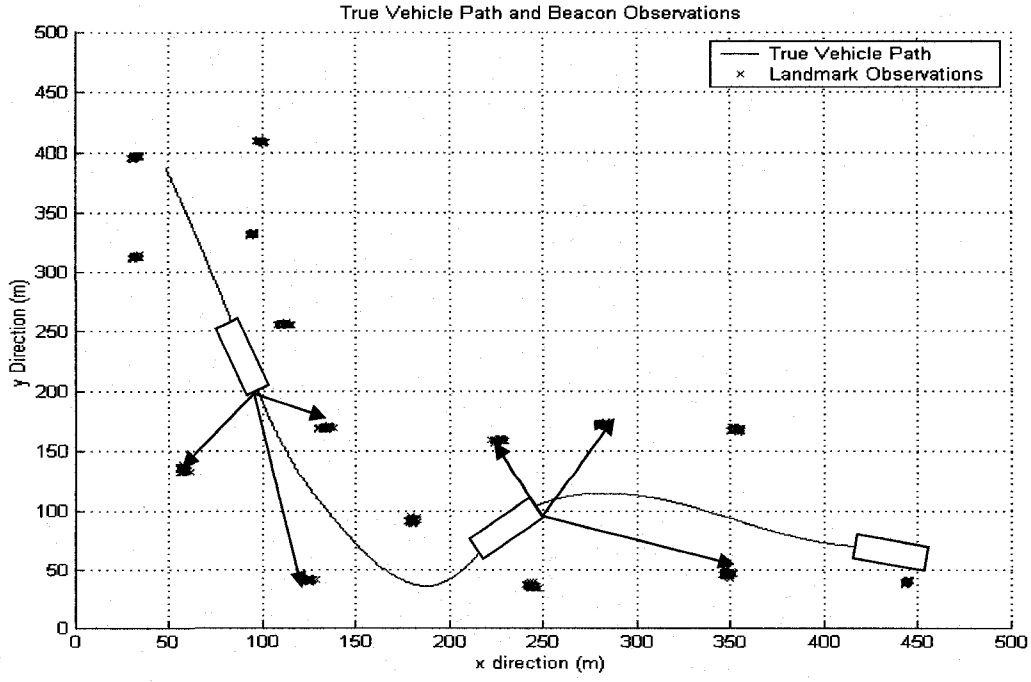


Figure 4.7: Localization and mapping simultaneously

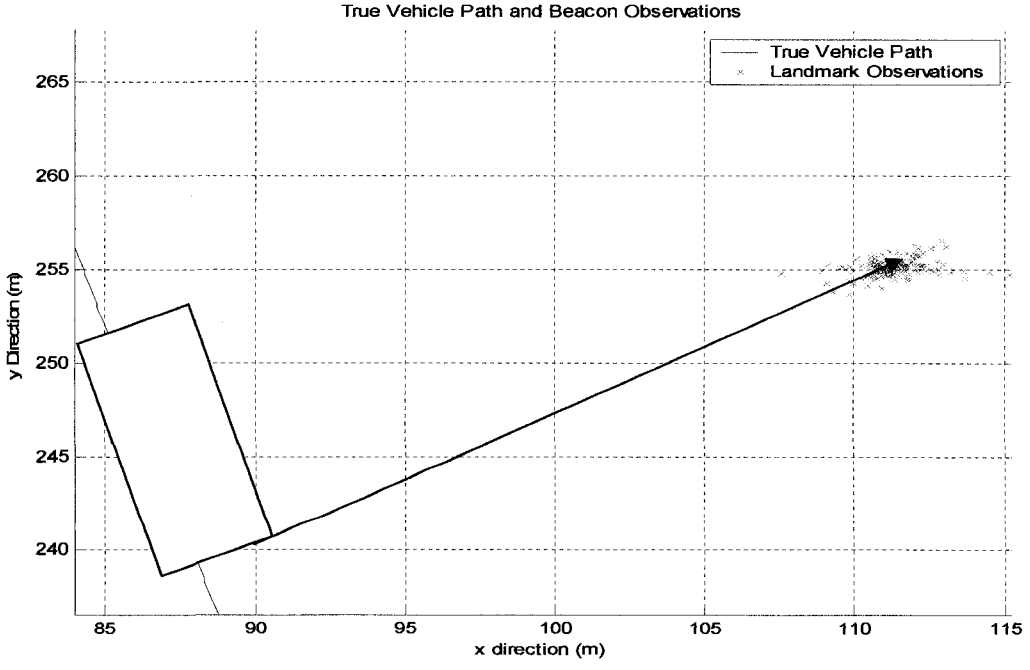
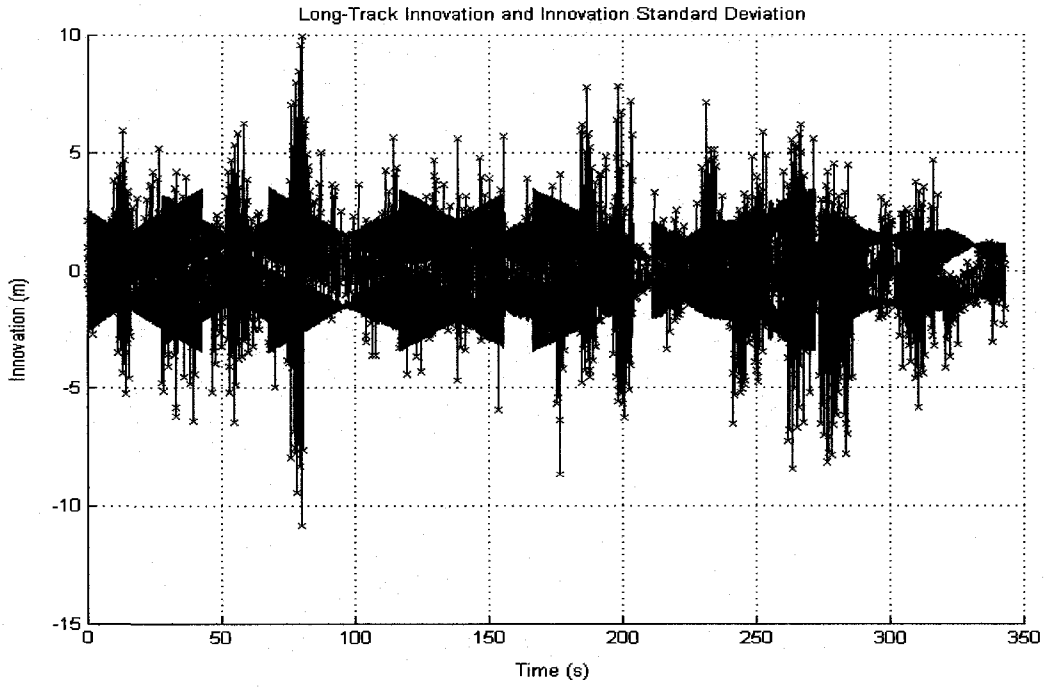
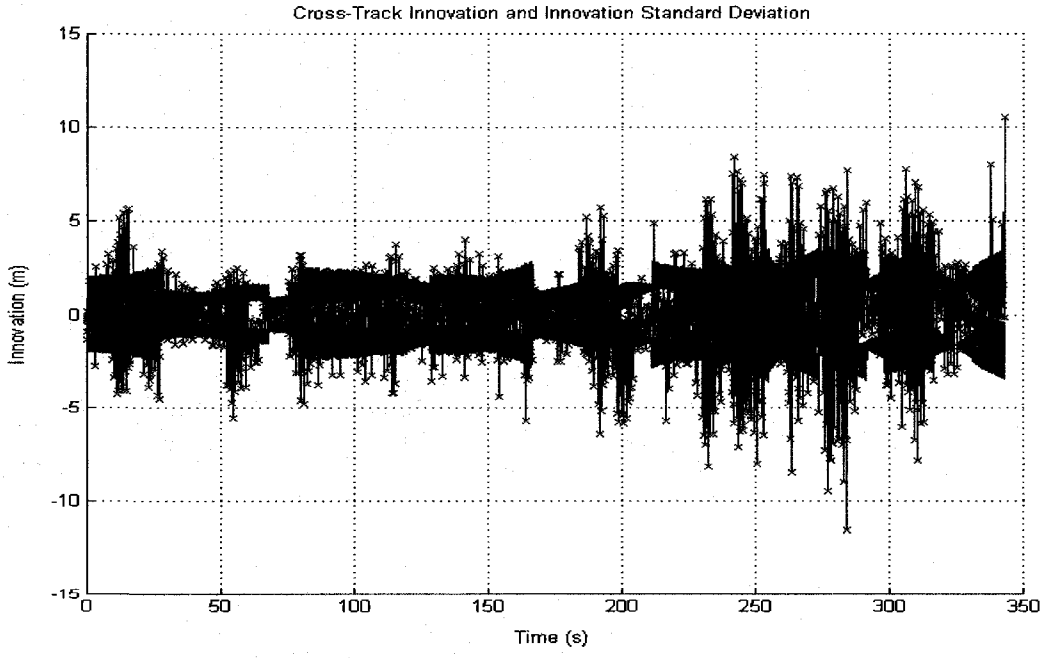


Figure 4.8: An enlarged diagram of landmark observations



(a)



(b)

Figure 4.9: Extended Kalman Filter innovations and innovation standard deviations in (a) Along-track direction and (b) Cross-track directions Green shaded area shows 1-σ confidence interval.

4.4 Summary

As presented in this chapter, a realistic outdoor mobile robot model was shown. The car like mobile robot has a mounted range sensor to detect landmarks and a control system equipped with optical encoder attached to wheels. The optical encoder makes a monitoring of robot's motion. Modeling procedure of motion and sensor of such a robot, is a necessary step to make the robot ready for real world applications. It was also assumed that the corrupting noises in both sensor and motion models are Gaussian, with zero-mean, independent and white.

The derived models, must be linearized since the real dynamic systems are mostly non-linear. These models can be linearized if they are linearizable. Usually, wheeled vehicles are good candidates for this purpose since their motion models are very suitable for linearization. Jacobian matrices are used to linearize both motion and sensor models. Once motion and sensor models are linearized, they can be fit into EKF algorithm to let it to make the best estimation out of robot's path. Some simulation results were shown in terms of true and estimated errors along x and y directions and for the robot orientation to better understand EKF-SLAM efficiency on a standard outdoor robotic vehicle. EKF innovations and innovation standard deviations along-track direction and cross-track directions were discussed in examples for the same robot on the same path to show the performance of EKF-SLAM under Gaussian conditions.

Chapter 5

FAST-SLAM

5.1 Rao-Blackwellised Particle Filtering

In chapter 3, EKF-SLAM was introduced and it was explained that only under Gaussian conditions the filter converges. We saw that the EKF approximated the SLAM posterior as a high dimensional Gaussian over all features in the map and the robot pose. As discussed in chapter 3, there were some concerns regarding the performance of EKF when the situation changed to confront unlimited landmarks in a large environment. The off-diagonal elements of the covariance matrix **P** of this multivariate Gaussian, constrain the correlation between all pairs of state variables. In some real world applications there are some issues such as computational complexity and single hypothesis data association that makes the use of EKF-SLAM absolutely inappropriate. While there have been so many efforts [19, 26, 32, 50, 51] to improve the performance of EKF-SLAM for such situations, Introducing FAST-SLAM by Montemerlo [11] made a revolutionary improvement in the design of recursive probabilistic SLAM and use of the filter with less concerns regarding the mentioned issues.

FAST-SLAM with its basis in Recursive Monte Carlo sampling, or particle filtering, directly represents a non-linear process model which is not necessarily under Gaussian conditions. This method represents distributions using a finite set of sample states or particles. Michael Montemerlo [11] used this approach on the basis of earlier work of Thrun and colleagues [27, 33]. A particle filter is a good tool to overcome discussed issues in chapter 3. Early

applications of particle filter in the SLAM problem go back to the work of Thrun, Fox, and Burgard [52] as *Monte Carlo Localization* where a set of particles were used to reperesent the distribution of possible states of a robot relative to a fixed map. This approach (particle filter) can be however reduced to the sample-space by applying Rao-Blackwellisaion (R-B). Whereby a joint state is partitioned according to the product rule $P(x_1,x_2) = P(x_2 \mid x_1)P(x_1)$. And if $P(x_2 \mid x_1)$ can be represented analytically, only $P(x_1)$ needs to be sampled. According to figure 2.4, the map is represented as a set of independent Gaussians, with a linear complexity, rather than a joint map covariance with quadratic complexity [23, 53].

5.1.1 Associated Data from Observation of a Landmark

As was described in chapter 3, the EKF estimates the posterior of state of the system based on equation (2.1). This means that the estimation is done sequentially over landmark location and robot's pose at every time step *k*. In FAST-SLAM algorithm [11, 22, 23, 24, 25, 54], the posterior is estimated over the landmark location as well but instead of the single state of the robot at time step *k*, the path is estimated. Probabilistically, FAST-SLAM is employing equation (5.1) instead of equation (2.1) from chapter 2.

$$P(\mathbf{X}_k^R, \mathbf{m} \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) \tag{5.1}$$

or simply

$$P(\mathbf{X}_k \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) \tag{5.2}$$

The difference between equations (5.1) and (2.2) is that in equation (2.2), only a single state $\mathbf{x}_k^R$ is being considered in the posterior estimation process, while in equation (5.1) the whole path of the robot $\mathbf{X}_k^R$ up to time step k is being involved.

In SLAM, every individual landmark has some specific information that is obtained from the range\bearing sensor observation and is unique, which means, it is dedicated only to that individual landmark's bearing and range. This unique information of a landmark can be independently incorporated into the filter calculation. Therefore, the assumption is that each observation gives the location data of only one individual landmark $\mathbf{m}_i$ relative to the current state of robot $\mathbf{x}_k^R$ at time step k. In a set of observation of different landmarks at time step k, which is $\mathbf{z}_k = \{ \mathbf{z}_{k,i}, \mathbf{z}_{k,j}, \ldots, \mathbf{z}_{k,n} \}$, every member of this set is the observation of one individual landmark at that time step. For instance, if at time step k, the robot observed landmark i, the observation regarding that landmark will be indicated as $\mathbf{z}_{k,i}$. The information or data related to that individual landmark observation which is incorporated to the map calculation, is identified by character $\mathbf{d}_{k,i}$. Here, we will consider a set of information or data obtained from observation of many landmarks at time step k. This set specifies that what information about what landmark and in what time step is being incorporated to the filter calculation. The set of all data associated with the map according to the set of observations can be presented as

$$\mathbf{d}_k = \{\mathbf{d}_{k,i}, \mathbf{d}_{k,j}, \ldots, \mathbf{d}_{k,n}\} \tag{5.3}$$

For simplicity in the calculation we consider one device for observation of landmarks $\mathbf{z}_k$ and consequently one data association $\mathbf{d}_k$ and with execution of one control $\mathbf{u}_k$ per each time step to avoid a cumbersome calculation.

5.1.2 Independent Landmarks in a Bayesian Network

Here, we recall figure 2.4 from the second chapter. In that picture, which is an example of a *Dynamic Bayesian Network* (DBN), $\mathbf{m}_j$ is observed at the first time step, $\mathbf{m}_i$ is observed at the first time step, time step *2* and time step *k* and finally, landmark $\mathbf{m}_n$ is observed at both time steps *2* and *k*. from this figure it is obvious that if the true path of the robot is known, landmarks $\mathbf{m}_i$, $\mathbf{m}_j$, and $\mathbf{m}_n$, are mutually independent. In other words, there will be no correlation between any two landmarks (two nodes m_i and m_j) of the map in the above DBN and this property is called "d-separation" [55] in a DBN. This property of DBN makes the SLAM problem low-dimensional because an observation of a landmark will not provide any information about the position any other landmark. In other words, landmarks in the map are mutually independent. This leads the SLAM problem to this important fact that the information of location of a landmark does not affect the other ones in the map. Therefore, the path of robot is known and consequently the location of the observed landmark can be estimated without need of any information regarding any other landmarks. Since correlation between elements of the map only arise through robot pose uncertainty, if the robot's true path is known, the landmark positions can be estimated independently. As a result, the SLAM posterior can be re-written as equation (1.1) which is a factorization of particle filter for

robot's pose estimation from a set of landmark estimation products. This factorization will be more described in section 5.1.3.

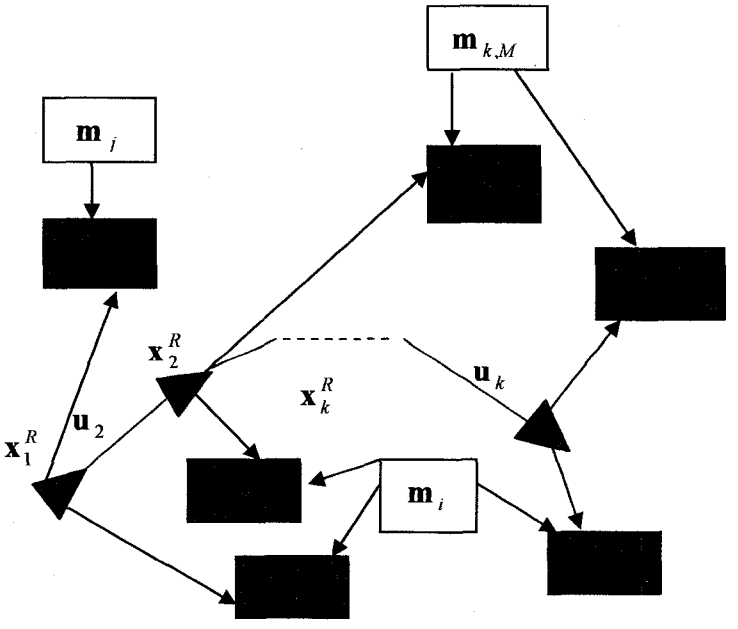


Figure 5.1: A dynamic Bayesian network.

5.1.3 Factorization of Particle Filter

From Figure 5.1, and using the definition of conditional probability, equation (5.1) can be expressed as follows:

$$P(\mathbf{X}_k^R, \mathbf{m} \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k)$$

$$= P(\mathbf{X}_k^R \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k)\, P(\mathbf{m} \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) \qquad (5.4)$$

Since we can write the second term of right hand side of equation (5.4) as follows

$$P(\mathbf{m} \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) = \prod_{i=1}^{M} P(\mathbf{m}_i \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) \tag{5.5}$$

Therefore, we can rewrite equation (5.4) as

$$P(\mathbf{X}_k^R, \mathbf{m} \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) =$$

$$\underbrace{P(\mathbf{X}_k^R \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k)}_{\text{Path Posterior}} \times \underbrace{\prod_{i=1}^{M} P(\mathbf{m}_i \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k)}_{\text{Landmark Estimators}} \tag{5.6}$$

A proof of equation (5.6) can be reviewed in Appendix A. This equation indicates that with the notification of robot's path, a landmark position is conditional to path of the robot and independent of the other landmarks. Concequently, there will be *M+1* filters; One *particle filter* for path of the robot (path posterior) and *M* Extended Kalman Filters for landmarks positions estimation (landmark estimators) according to the path of the robot. This factorization [54] is absolutely exact and fits very well for any SLAM application. When the filter estimates path of the robot using particles, every landmark position is being tracked by EKF. Number of EKFs depends on the number of particles P. We can benefit from this advantage that there will be M×P EKFs In total to estimate landmarks locations using path of the robot at each time step and this makes the filter with multiple hypothesis data association. Figure 5.2 is a simulation of particles shaping around the path for estimation of it. Figure 5.3 illustrates how many particles deal with path of the robot and landmarks positions at the same time. In this figure, we can figure out that the "structure combines sampling over a small set

of variables with closed for calculation of certain marginals" [11] and that is an example of RBPF. The super script on the left hand side of the states in figure 5.3 shows the number of the particle that is used to estimate every individual state of the robot in time step *k*. Character μ indicates the mean of a normal distribution and σ is the covariance.

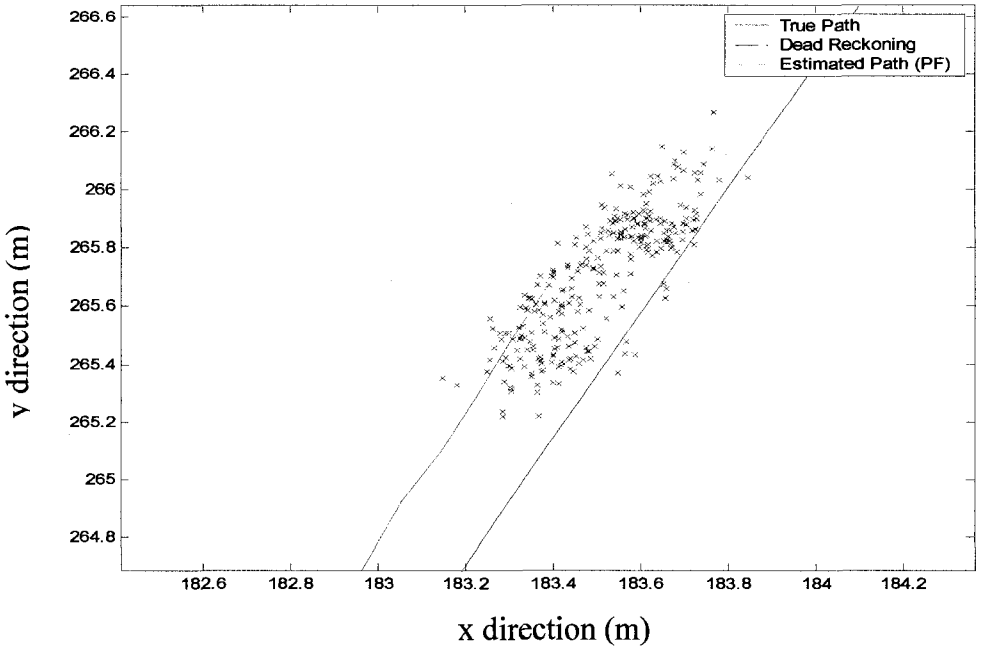


Figure 5.2: Enlarged simulation of how sample particles are shaped while robot is trying to stay on the path.

	State of robot	Landmark 1	⋯	Landmark M
Particle 1	${}^{1}x_k^R$	${}^{1}\mu_1$, ${}^{1}\sigma_1$	⋯	${}^{1}\mu_M$, ${}^{1}\sigma_M$
Particle 2	${}^{2}x_k^R$	${}^{2}\mu_1$, ${}^{2}\sigma_1$	⋯	${}^{2}\mu_M$, ${}^{2}\sigma_M$
⋮	⋮	⋮	⋮	⋮
Particle P	${}^{P}x_k^R$	${}^{P}\mu_1$, ${}^{P}\sigma_1$	⋯	${}^{P}\mu_M$, ${}^{P}\sigma_M$

Figure 5.3: States of the robot with regards to "P" particles while there are M landmarks estimated by M×P EKFs.

5.1.4 State Samples

From figure 5.3 we can see that pose of the robot for a specific particle 'n' can be expressed as follows:

$$ {}^{n}x_k^R = \{ {}^{n}X_k^R, {}^{n}\mu_{k,1}, {}^{n}\sigma_{k,1}, \ldots, {}^{n}\mu_{k,M}, {}^{n}\sigma_{k,M} \} \tag{5.7} $$

where, ${}^{n}X_k^R$ is the n^{th} particle's estimation of the robot's path. A sample of all particles at time step k is expressed as

$$ \text{Sample of } x_k^R = \{ {}^{1}x_k^R, {}^{2}x_k^R, \ldots, {}^{n}x_k^R \} \tag{5.8} $$

The superscript on the left hand side of each term shows the involving of the n^{th} particle at time step k for estimating the path and M landmarks locations in the map using EKF and according to the estimated path of the robot using RBPF.

5.2 Data Association Assumptions

In this chapter, we will first consider mapping between observation and landmarks as known data. Known data association simply means that information of observation of each landmark is being incorporated to the map without being mixed with any other data observation of other landmarks. In other words, the data from each observation is absolutely known and related to only one landmark.

The classical way to deal with the data association problem in SLAM is to choose $\mathbf{d}_{k,i}$ such that it maximizes the likelihood of sensor measurement $\mathbf{z}_k$ given all available data.

$$\hat{\mathbf{d}}_{k,j} = \text{argmax}\, P(\mathbf{z}_k \mid \mathbf{d}_{k,i}, \hat{\mathbf{d}}_{k-1}, \mathbf{X}_k^R, \mathbf{Z}_{k-1}, \mathbf{U}_k) \tag{5.9}$$

The term $P(\mathbf{z}_k \mid \mathbf{d}_{k,i}, \hat{\mathbf{d}}_{k-1}, \mathbf{X}_k^R, \mathbf{Z}_{k-1}, \mathbf{U}_k)$ is called as a likelihood and when the data association picks the maximum value of it, it is called as the *Maximum Likelihood Estimator* (MLE). The assumption of known data association simplifies the computations and is easy and straightforward to understand. While this is not the case in the most real world application, with the assumptions of known data association in FAST-SLAM, we can avoid the cumbersome calculations and later add the parameters of unknown data association to our computation. Thrun and colleagues [25] have derived an improved algorithm regarding the unknown data association for the current version of FAST-SLAM. Nonetheless, for simplicity we will consider the data association as known for the calculation and later on at the end of this chapter, the equations for the unknown data association case will be developed.

5.3 FAST-SLAM Algorithm

FAST-SLAM employs four steps to draw an estimated pose out of a series of samples from equation (5.8). These steps are *state sampling, landmark estimation update, importance weight,* and *importance sampling update*. Followings are four steps discussed in details.

5.3.1 State Sampling

State sampling is the first step of FAST-SLAM algorithm. Before final prediction of the path, pose or state of the robot must be estimated for the time step *k* and added to the path upto one time step earlier (time step *k*-1). This estimation is done for each particle that belongs to the set of particles in equation (5.8), and is up to time step *k-1*, where the pose has already been predicted successfully.

Now a guess of pose of the robot must be done for the current time step *k*. To do so, a probabilistic motion model based on state of the robot at time step *k* is employed, while can be expressed as:

$$ {}^{n}x_k^R \sim P\left(x_k^R \mid \mathbf{u}_k, {}^{n}x_{k-1}^R\right) \tag{5.10} $$

Estimation at this level is added to a temporary set of particles along with the estimated path at time step *k*-1 which is ${}^{n}X_{k-1}^R$. Assuming that the sample of particles in equation (5.8) at time step *k-1*, is distributed as $P\left(\mathbf{X}_{k-1}^R \mid \mathbf{Z}_{k-1}, \mathbf{U}_{k-1}, \mathbf{x}_0^R\right)$, the new particles distribution can be expressed as

$$ P\left(\mathbf{X}_k^R \mid \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0^R\right) \tag{5.11} $$

Equation (5.10) is according to the distribution of particle filtering. The size of map does not affect the particles sampling time consuming. No matter how big the map of environment is,

drawing the new pose of each particle is a constant time operation. Furthermore, the system does not have to be linearized like in the case of EKF-SLAM. In fact equation (5.9) can deal with any non-linearity of the system and this can be a significant advantage of FAST-SLAM compare to EKF-SLAM. The distribution according to equation (5.11) is also referred to proposal distribution of particle filtering.

5.3.2 Landmark Estimation Update

If the data association $\mathbf{d}_{k,i}$ at time step k regarding the observation of the i^{th} landmark $\mathbf{m}_i$ is known, the whole data association of observation of n landmarks at this time step can be expressed as the following set:

$$\mathbf{z}_k = \{ \mathbf{z}_{k,i}, \mathbf{z}_{k,j}, \ldots, \mathbf{z}_{k,n} \} \tag{5.12}$$

While the observation of all landmarks up to time step k can be expressed as:

$$\mathbf{Z}_k = \{ \mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_k \} = \{ \mathbf{Z}_{k-1}, \mathbf{z}_k \} \tag{5.13}$$

The known data association assumption is just for simplicity to derive equations of observation. Montemerlo has derived a modified version of FAST-SLAM called FAST-SLAM 2.0 [17] that deals with unknown data association as well. If there are M landmarks in the environment that are already observable, then M low dimensional EKFs are considered for

each of P particles. In total, there will be M×P Extended Kalman Filters involved in the estimation of landmarks in FAST-SLAM algorithm. In fact, each particle carries M Extended Kalman Filters that estimate the location of M landmarks based on path estimation of the robot. At this step, the independency of landmarks observations makes M independent estimation of landmarks based on the estimated trajectory. If a landmark is not observed at the current time step, the posterior data at the previous time step will be substituted according to the following equation.

$$P(\mathbf{m}_i \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R) = P(\mathbf{m}_i \mid \mathbf{X}_{k-1}^R, \mathbf{Z}_{k-1}, \mathbf{U}_{k-1}, \mathbf{x}_0^R) \tag{5.14}$$

If a landmark is observed then the following equation that is simplified according to Bayes rule and Markov localization can estimate the location of i^{th} landmark.

$$P(\mathbf{m}_i \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R)\, P(\mathbf{z}_k \mid \mathbf{X}_k^R, \mathbf{Z}_{k-1}, \mathbf{U}_k)$$
$$= P(\mathbf{z}_k \mid \mathbf{m}_i, \mathbf{x}_k^R)\, P(\mathbf{m}_i \mid \mathbf{X}_{k-1}^R, \mathbf{Z}_{k-1}, \mathbf{U}_{k-1}, \mathbf{x}_0^R) \tag{5.15}$$

Equation (5.15) is called the landmark update equation. To better express this equation with the consideration of the non-linearity of the measurement which is expressed as equation (5.16), equations (5.18), (5.19), and (5.20) are expressed as follows:

$$P(\mathbf{z}_k \mid \mathbf{x}_k) \Leftrightarrow \mathbf{z}_k = h(\mathbf{x}_k^R, \mathbf{m}_{k,i}) + \mathbf{v}_k \tag{5.16}$$

$$\mathbf{v}_k \sim N(0, \mathbf{R}_k) \tag{5.17}$$

$$\hat{\mathbf{z}}_k = h\,({}^{n}\mathbf{x}_k^R, \mu_{k-1,i}) \tag{5.18}$$

$$\Lambda_k = \left.\frac{\partial h(\mathbf{x})}{\partial \mathbf{z}_{k,i}}\right|_{\mathbf{x}={}^{n}\mathbf{x}_k^R, \mathbf{z}_{k,i}={}^{n}\mu_{k-1,i}} \tag{5.19}$$

$$h\,(\mathbf{x}_k^R, \mathbf{z}_{k,i}) \approx \hat{\mathbf{z}}_k + \Lambda_k(\,\mathbf{z}_{k,i} - {}^{n}\mu_{k-1,i}) \tag{5.20}$$

where $\mu_{k-1,i}$ is the mean of i^{th} landmark at time step k, and Λ_k is the matrix of partial derivatives observation function over the bservation vector of i^{th} landmark at time step k.Using equation (5.20), both measurement and probability density function of the map will be Gaussian.

$$\mathbf{z}_k \sim N(\hat{\mathbf{z}}_k + \Lambda_k(\,\mathbf{z}_{k,i} - {}^{n}\mu_{k-1,i}), \mathbf{R}_k) \tag{5.21}$$

$$P(\mathbf{m}_i \mid \mathbf{X}_{k-1}^R, \mathbf{Z}_{k-1}, \mathbf{U}_{k-1}, \mathbf{x}_0^R) \sim N({}^{n}\mu_{k-1,i}, {}^{n}\Sigma_{k-1,i}) \tag{5.22}$$

Then we will have the EKF algorithm to update a landmark location at time step k:

$$\widetilde{\mathbf{Z}}_k = \Lambda_k\,{}^{n}\Sigma_{k-1,i}\,\Lambda_k^T + \mathbf{R}_k \tag{5.23}$$

$$\mathbf{K}_k = {}^{n}\Sigma_{k-1,i}\,\Lambda_k^T\,\widetilde{\mathbf{Z}}_k^{-1} \tag{5.24}$$

$$ {}^{n}\mu_{k,i} = {}^{n}\mu_{k-1,i} + \mathbf{K}_k(\mathbf{z}_k - \hat{\mathbf{z}}_k) \tag{5.25} $$

$$ {}^{n}\Sigma_{k,i} = [I - \mathbf{K}_k \Lambda_k] {}^{n}\Sigma_{k-1,i} \tag{5.26} $$

where ${}^{n}\Sigma_{k,i}$ is the covariance of i^{th} landmark. When a landmark is in the range of the robot's sensor, there is a distance D between the sensor and the landmark, and there is an angle β expressing the orientation of the landmark with respect to the sensor as shown in figure (4.1) in chapter 4. The range/bearing sensor reads the coordinate and orientation of the robot and accordingly, the observation can be expressed as:

$$ \mathbf{z}_k = h(\mathbf{x}) + \mathbf{v}_k = \begin{bmatrix} \mathbf{z}^{D}_{k,i} \\ \mathbf{z}^{\beta}_{k,i} \end{bmatrix} + \mathbf{v}_k = \begin{bmatrix} \sqrt{(x_k^{m_i} - x_k^R)^2 + (y_k^{m_i} - y_k^R)^2} \\ \tan^{-1}\left(\frac{(y_k^{m_i} - y_k^R)}{(x_k^{m_i} - x_k^R)}\right) - \varphi_k^R + \frac{\pi}{2}) \end{bmatrix} + \mathbf{v}_k \tag{5.27} $$

where $(x_k^{m_i}, y_k^{m_i})$ is the coordinate of the observed landmark with respect to the global reference system and φ is the angle that the coordinate system of the sensor/robot (The sensor coordinate system is attached to the robot) makes with the global reference system. The Jacobian Λ_k is then expressed as follows:

$$ \Lambda_k = \begin{bmatrix} \frac{x_k^{m_i} - x_k^R}{\sqrt{\Delta}} & \frac{y_k^{m_i} - y_k^R}{\sqrt{\Delta}} \\ -\frac{y_k^{m_i} - y_k^R}{\Delta} & \frac{x_k^{m_i} - x_k^R}{\Delta} \end{bmatrix} \tag{5.28} $$

where

$$\Delta = (x_k^{m_i} - x_k^R)^2 + (y_k^{m_i} - y_k^R)^2 \tag{5.29}$$

5.3.3 Importance Weights

Since particles that are used to predict the path of the robot are distributed as equation (5.11), they need to be matched with equation (5.30).

$$P(\mathbf{X}_k^R \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R) \tag{5.30}$$

By so called importance sampling, this matching can be done. If there is no way to take direct sample out of a function, a technique called importance sampling can be used. Importance sampling draw samples from a proposal function. A weight is given to each sample that is as follows:

$$^{n}\hat{w}_k = \frac{\textbf{Target distribution}}{\textbf{Proposal distribution}} = \frac{\mathbf{P}(^{n}\mathbf{X}_k^R \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R)}{\mathbf{P}(^{n}\mathbf{X}_k^R \mid \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0^R)} \tag{5.31}$$

This process which is called SIR algorithm is an example of Rubin's sampling importance resampling [53]. According to this process, a new set of unweighted samples is drawn from

the weighted set with probabilities in proportion to the weight. Using Bayes rule and Markov assumption, the weighted samples can be expressed as:

$$ {}^{n}\hat{w}_k = P(\mathbf{Z}_k \mid {}^{n}\mathbf{X}_k^R, \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0^R) \tag{5.32} $$

As the landmark estimator is an EKF, this observation likelihood can be computed in closed form. This probability can be calculated using the innovation and its covariance matrix. The importance weight can be then written as

$$ {}^{n}\hat{w}_k = \frac{1}{\sqrt{|2\pi\tilde{\mathbf{Z}}_{k,i}|}} \exp\left(-\frac{1}{2}\tilde{\mathbf{z}}_{k,i}^T \tilde{\mathbf{Z}}_{k,i}^{-1} \tilde{\mathbf{z}}_{k,i}\right) \tag{5.33} $$

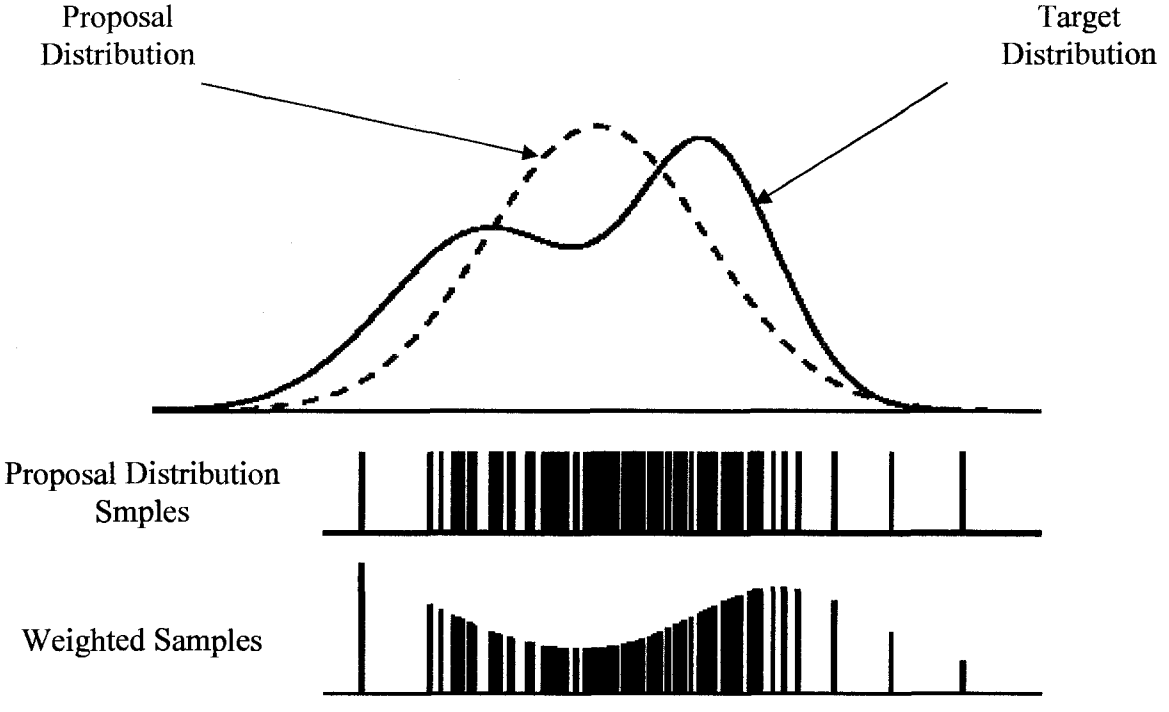


Figure 5.4: Drawing samples from the proposal distribution [11].

Figure 5.4 depicts the drawing of samples from the proposal distribution (dashed curve) in one dimension. The solid curve indicates the target distribution. This figure shows how hard it is to draw samples from the target distribution. It is obvious that the samples that have higher weights in the regions of the proposal distribution are larger than the target distribution and vise versa.

5.3.4 Importance Sampling Update

After weighting temporary particles, a new set of samples $\mathbf{X}_k$ is drawn with replacement from this set with probabilities in proportion to the weights. This step is called importance re-sampling in which FAST-SLAM corrects the pose sample $\mathbf{X}_k$ since there has not been any most recent measurement involved in the process. Different re-sampling techniques can be found in [56].

5.4 Sample Impoverishment

As discussed in section 5.3.3, the proposal and target distributions are getting matched to draw samples for the estimation of path. The better these distribution functions fit, the more samples are incorporated to the path estimation. If for some reason, target and proposal distributions do not match, many particles will be ignored in the resampling step.

If the motion is noisy and the sensor is very accurate, which means the amount of noise in sensor readings is near to zero, the particle throwing away, happens in the third step of FAST-SLAM algorithm. This effect will be demonstrated based on the simulation data in the next

chapter. Obviously, when the measurement noise goes towards zero, the proposal and target distributions are getting less matched since lots of particles (samples) are thrown away. Since many of samples will be thrown away and since there will not be enough samples to estimate the state of the robot at each time step, the filter diverges. This phenomenia is called sample impovrishement and is one disadvantage of this particular FAST-SLAM. Montemerlo and Thrun have improved a version of FAST-SLAM called FAST-SLAM 2.0 [17] that very well overcomes this issue.

5.5 Computational Complexity in FAST-SLAM

5.5.1 Linear Computational Complexity

According to the FAST-SLAM representation, it seems that the computational complexity requires time linear in the number of landmarks times the number of particles associated in the calculation of path of the robot. In other words, the computational complexity seems to be $M \times P$. Since P particles are used for every update, we may have to live with this fact that P (Number of particles) will continue to be in the linear complexity of the rest of calculation for the path estimation. The linearity in number of landmarks M, is due to the fact that any considered particle in the weighted particle set may be duplicated several times. This duplication is the result of sampling which is done by the replacement act. At each time step, every particle set, up to the current time step, is replaced by the new set of particles and this replacement happens continuously for the rest of the path. Since the length of particles linearly depends on number of landmarks in the map, this replacement is linear in the size of

the map. Likewise, most of landmark filters remain unchanged and identical at every time step.

5.5.2 Logarithmic Computational Complexity

For more efficiency and to make the linear replacement of landmarks to a logarithmic form a new representation of computational complexity may be considered. By this representation, the linearity of M landmark filters that are duplicated at each time step will be shared among particles and consequently the FAST-SLAM algorithm will be more efficient. To convert the linearity of M filters to a logarithmic form we can change the particle representation form as an array of landmark filters to a binary tree as indicated in figure 5.5. In this figure, 'i' identifies a randomly chosen landmark. In this figure a binary tree for eight landmark filters is shown [25]. From this figure it is obvious that any sub-tree can be shared among many landmarks. While this representation is more complicated, alot of memory for the filter computation will be saved. On average, the result of such tree will end up with a logarithmic complexity in computation and consequently the computation will be less complex as of the linear procedure. In fact the computation complexity will be performed as $P \times \log(M)$.

Figure 5.6 depicts a specific situation in which the third landmark at time step k is observed and updated. In this incomplete tree, only the third landmark Gaussian parameters are updated and instead of entire tree, a single path from the root to the third landmark Gaussian parameters is duplicated. The tree is completed by copying the missing pointers from the tree of the generating particles. Thus branches that leave the modified path will point to the unmodified sub-trees of the generating particle.

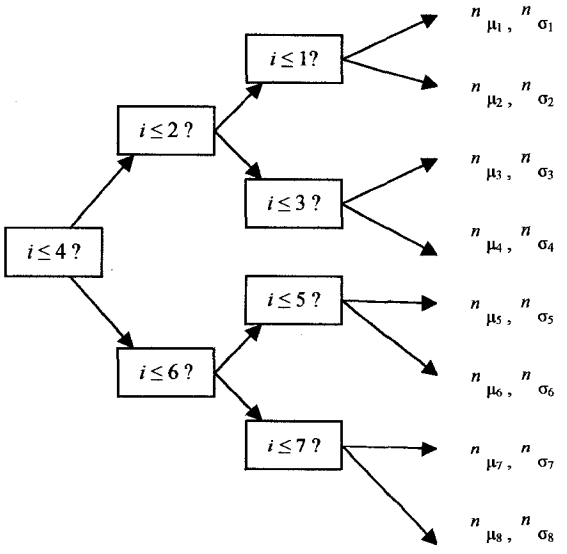


Figure 5.5: The binary tree of landmark filters [25]

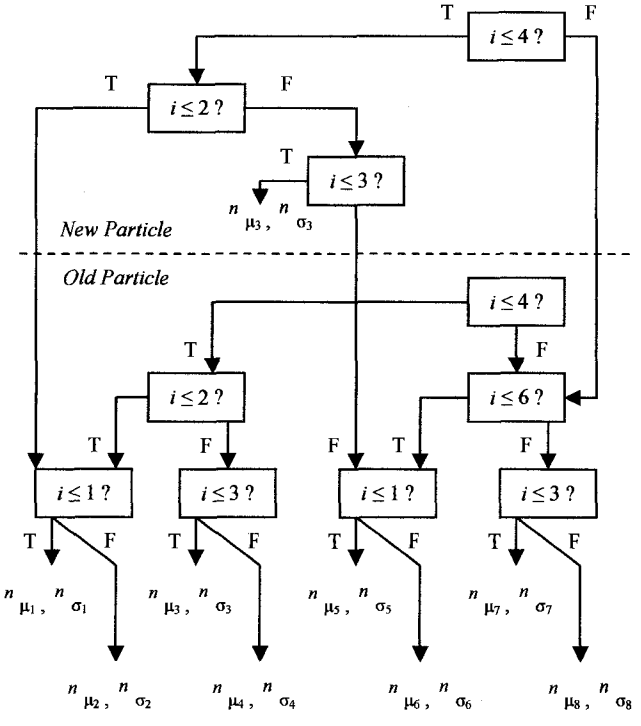


Figure 5.6: The landmark tree update procedure [25]

Clearly, generating the modified tree takes time logarithmic in *M*. Moreover accessing a Gaussian also takes time logarithmic in *M*, since the number of steps needed for navigating to a leaf of the tree is equivalent to the length of the path. Therefore, both generating and accessing a partial tree can be done logarithmic in *M*. Since at every update step *M* new particles are generated, the FAST-SLAM algorithm requires time P×log(*M*).

5.5.3 Computational Complexity Simulation

Figure 5.7 illustrates the simulated result of scaling performance evaluation. In this simulation, for 200 iterations number of landmarks with the variation of up to ten thousand for the linear time FAST-SLAM while this number varies from 1 to more than one million landmarks for the logarithmic time fast slam. 300 particles have been considered for this simulation. The diagrams show the time consuming for computing 500 sensor updates with all observable landmarks during 7 different runs. It is obvious that with the logarithmic time, FAST-SLAM performance increases substantially compare to that of FAST-SLAM with linear time consideration. Logarithmic FAST-SLAM also saves a huge amount of memory for the calculation of path and the environment map. Figure 5.8 indicates memory needed for FAST-SLAM calculation for both logarithmic and linear time FAST-SLAM. In this diagram the advantage of logarithmic FAST-SLAM is validated compare to the linear one. The logarithmic FAST-SLAM requires only a few Mega-bytes even though the incorporated landmarks to the map increases up to 1,000,000 while the linear FAST-SLAM needs more than 20 Mega-bytes for only a few of landmarks.

5.6 Data Association in FAST-SLAM

In practice, the number of landmarks in the map can not be obtained trivially such as what was shown in figure 5.6. Instead, the data association has to be solved between the observation of landmarks at time step k and set of landmarks in the map. This problem can be solved by maximum likelihood rule as described in section 5.2. If $d_{k,i}$ is the data association where $i \in \{1, 2, ..., M\}$ set of landmarks, the probability of $d_{k,i}$ can be described by

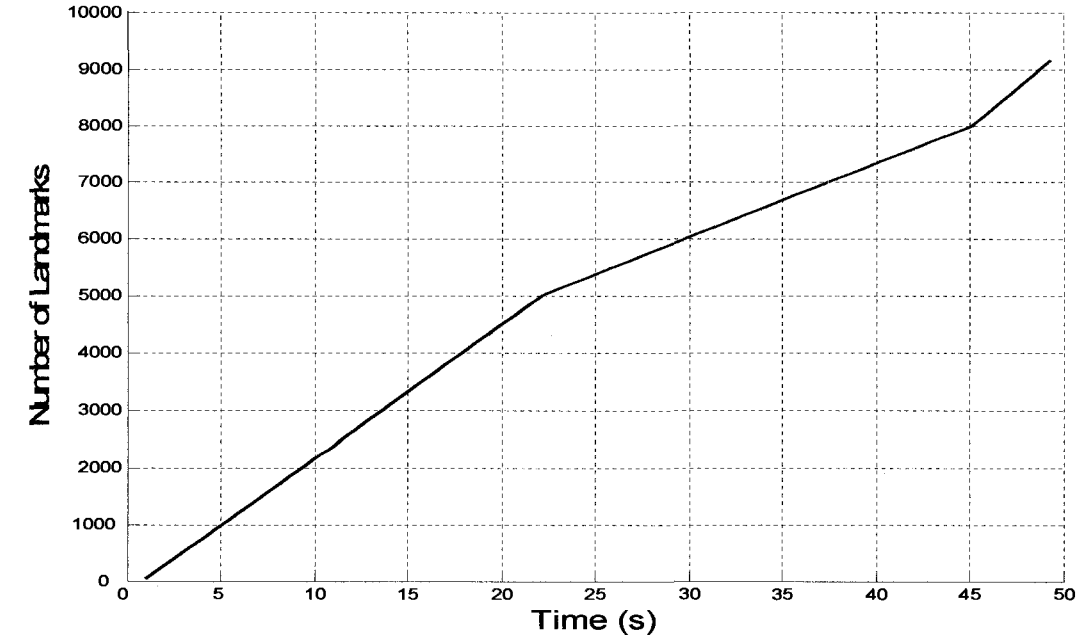
$$P(d_{k,i} \mid \mathbf{Z}_k, \mathbf{U}_k) = \int P(d_{k,i} \mid \mathbf{X}_k, \mathbf{Z}_k, \mathbf{U}_k) P(\mathbf{X}_k \mid \mathbf{Z}_k, \mathbf{U}_k)\, d\mathbf{X}_k \tag{5.34}$$

$$\overset{RBPF}{\approx} \sum_{n} P(d_{k,i} \mid {}^{n}\mathbf{X}_k, \mathbf{Z}_k, \mathbf{U}_k) \tag{5.35}$$

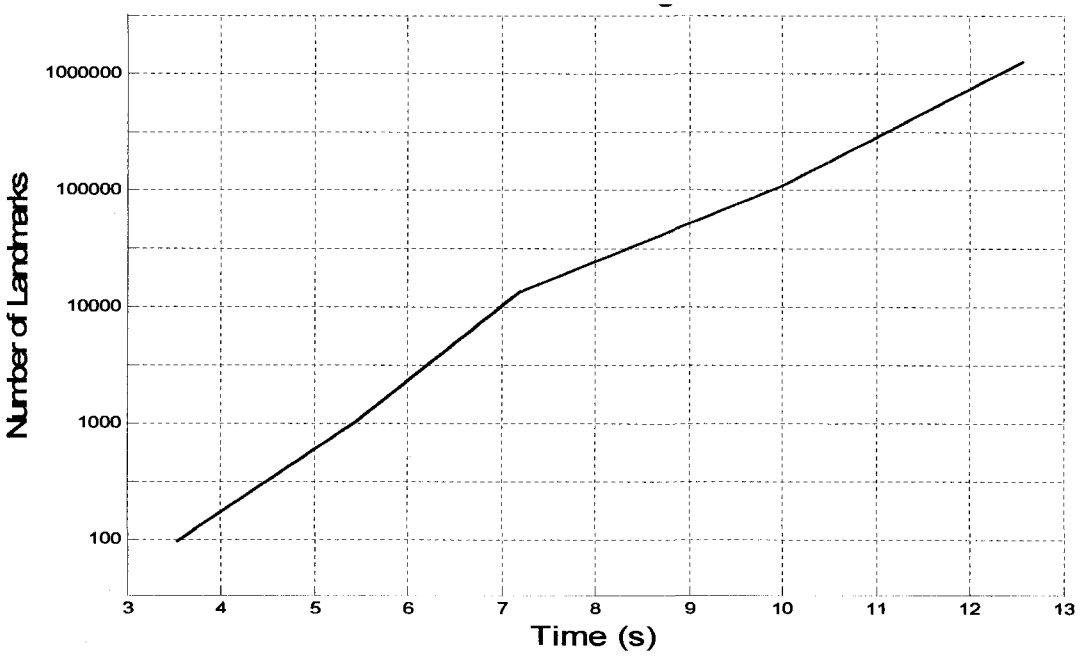
$$\overset{Markov}{=} \sum_{n} P(d_{k,i} \mid {}^{n}\mathbf{x}_k, \mathbf{z}_k) \tag{5.36}$$

$$\overset{Bayes}{\propto} \sum_{n} P(\mathbf{z}_k \mid {}^{n}\mathbf{x}_k, d_{k,i}) \tag{5.37}$$

When $d_{k,i}$ in equation (5.37) reaches its maximum value, it means the maximum likelihood data association. When the maximum value of probability of the data association in equation (5.34) is below a threshold α, the landmark is considered previously unseen and the map is augmented accordingly.



(a)



(b)

Figure 5.7: (a) run time of 500 iterations of linear FAST-SLAM (b) run time of 500 iterations of logarithmic FAST-SLAM [11]

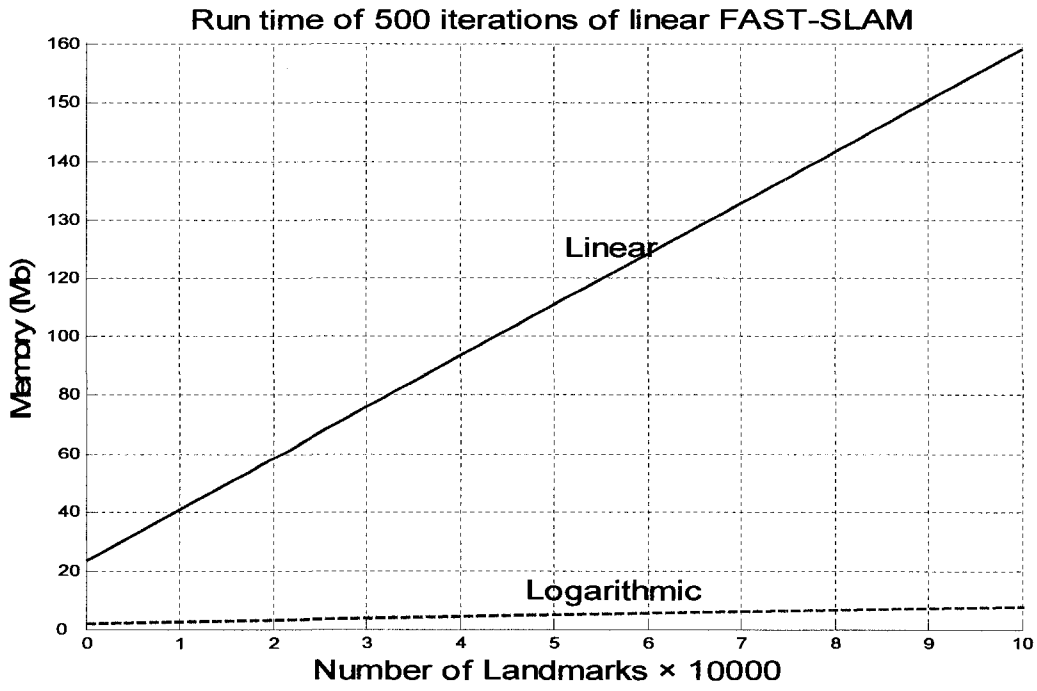


Figure 5.8: Memory usage of logarithmic FAST-SLAM compare to that of linear FAST-SLAM [11]

Unlike EKF-SLAM which is based on single-hypothesis data association, FAST-SLAM does not determine the data association once for each observation. This property of FAST-SLAM is called multiple-hypothesis data association. Consequently, if a wrong observation of a landmark is not accurate enough, the data is simply ignored and is not fused to the map since the particle carrying it could not achieve enough weight in the resampling process and has already been ignored. FAST-SLAM has multiple data association property and only the most accurate data is being incorporated and fused to the map. The particles with the most correct data association that have enough weight can pass the resampling process and the most accurate data association which are carried by those particles are filtered from the resampling process and considered to be fused to the map. Multiple-hypothesis data association is a very

important property of FAST-SLAM that makes it much more accurate compare to EKF-SLAM and prevents the filter to diverge.

5.7 FAST-SLAM with Unknown Data Association

In the previous sections of this chapter we assumed $\mathbf{d}_k$ as known data association. All calculation sofar has been according to known data association assumption since it makes the calculation of FAST-SLAM easier. Since in real world, data association is rarely known, we have to extend the FAST-SLAM algorithm to domains in which the mapping between observations and landmarks is unknown. As was descried in section 5.2 and according to equation (5.9), $\mathbf{d}_k$ is chosen the way it maximize the likelihood of sensor measurement $\mathbf{z}_k$ given all available data. This approach is called Maximum Likelihood Estimator (MLE) or *Nearest Neighbor* data association.

In chapter 3, we described that EKF-SLAM, posses a single hypothesis data association property. Consquently, EKF algorithm is fragile to ambiguity of data association and fails to estimate the path reasonabaly. This ambiguity is due to the fact that uncertainty in SLAM posterior generates uncertainty in observing nearby landmarks. To better understand the data association uncertainty problem, the following section discusses the matter in detail.

5.7.1 Uncertainty in Data Association

As was described in section 1.2.6 of chapter 1, both motion and measurement are subject to noise. As measurement becomes noisy, the distribution of possible observations of each

landmark becomes more uncertain. "If measurement is very noisy, the distributions of observation of close landmarks will begin to overlap substantially. This overlap causes the ambiquity in landmarks indentification by the algorithm. Montemerlo [11] has refered to data association ambiguity caused by measurement noise as *measurement ambiguity*. Figure 5.9 indicates an example of measurement ambiguity. In this figure, ellipses illustrate the range of probable observations from two nearby landmarks. The dark circle depicts possible observation from either landmark.

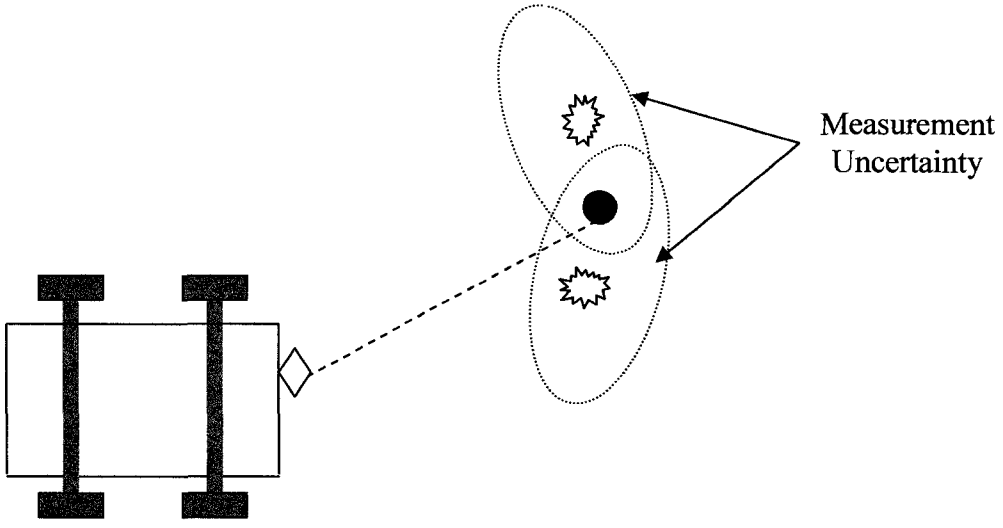


Figure 5.9: Measurement Ambiguity

Attributing an observation to the wrong landmark due to measurement and ambiguity, will increase the error of the map and robot pose, but its impact will be relatively minor. Since the observation could have been generated by either landmark with high probability, the effect of observation of the landmark positions and the robot pose will be small. The covariance of one landmark will be slightly overestimated while the covariance of the second landmark will be slightly underestimated. If multiple observations are incorporated per control a data

association mistake due to measurement ambiguity of one observation will have relatively little impact on the data association decisions for the other observations.

Ambiguity in data association caused by motion noise can have much more severe consequences on estimation accuracy. High emotion noise will lead to higher pose ucertainty after incorporating a control. If this pose of uncertainty is high enough, assuming different robot poses in this distribution will imply drastically different ML data association hypothesis for the subsequent observations. According to figure 5.10, this motion ambiguity is easily induced if there is significant rotational error in the motion of robot. Moreover, if multiple observations are incorporated per control the pose of the robot will correlate the data association decisions of all of the observations. If the SLAM algorithm chooses the wrong data associations for the single observation due to motion ambiguity the rest of the data associations also will be wrong by high probability. Choosing a wrong number of incorrect data associations will typically lead to divergence in an EKF" [11].

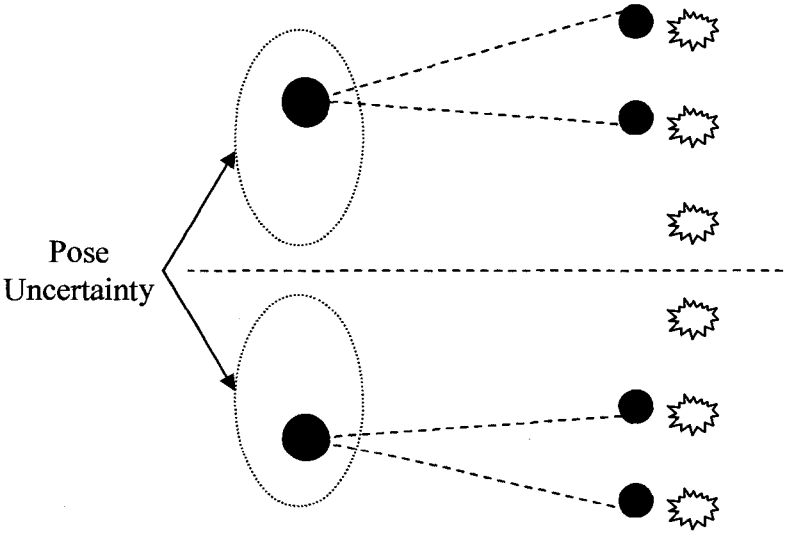


Figure 5.10: Motion ambiguity-Uncertainty due to the robot's.

5.7.2 Per-Particle Data Association

Unlike most EKF-based Approaches, FAST-SLAM takes a multi hypothesis approach to the data association problem. Each particle represents a different hypothesized path of the robot, so data association decisions can be made on a per-particle basis. Particles that pick the correct data association will receive high weights because they explain the observations well. Particles that pick wrong associations will receive low weights and be removed in a future resampling step.

Per particle-data association has several important advantages over standard MLE data association. First, it factors robot's pose uncertainty out of the data association problem. Since the motion ambiguity is the more severe form of data associationambiguity, conditioning the data association decisions on hypothesized robot paths, seems like a logical choice. According to figure 5.10, some of the particles would draw new robot poses consistent with the data association on the top, while others would draw poses consistent with the data association hypothesis on the bottom.

Doing data association on a per-particle basis also makes the data association problem easier. In the EKF, the uncertainty of a landmark position is due to uncertainty in the pose of the robot as well as uncertainty due to measurement error. In FAST-SLAM, Unncertainty of the robot pose is represented by the entire particle set. The landmark filters in a single particle are not affected by motion noise because they are conditioned on a specific robot path. This is especially useful if the robot has noisy motion and an accurate sensor.

The simplest approach to per-particle data association is to apply MLE data association heuristic, only on a per-particle basis. Again, because the landmark estimators are EKFs, the

likelihood in equation (5.38), can be calculated using innovations. This likelihood is exactly the same as the importance weight calculated in equation (5.9) for the original FAST-SLAM algorithm. If the value of this likelihood falls below some threshold P_0, a new landmark is added to the particle.

$$P(\mathbf{z}_k \mid {}^n\mathbf{X}_k^R, \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{d}_k) = \frac{1}{\sqrt{|2\pi\tilde{\mathbf{Z}}_{k,i}|}} \exp\left(-\frac{1}{2}\tilde{\mathbf{z}}_{k,i}^T \tilde{\mathbf{Z}}_{k,i}^{-1} \tilde{\mathbf{z}}_{k,i}\right) \tag{5.38}$$

MLE data association tends to work much better in FAST-SLAM than it does in EF-based approaches. The main reason for this success is that the most severe component of data association ambiguity comes from uncertainty in the robot's pose. Some fraction of the particles will draw new poses that are consistent with the true pose of the robot. These poses will receive correct data associations and explain the observations well. Particles that draw poses far from the true pose will receive wrong data associations that explain the data poorly.

5.7.3 Adding New Landmarks

"Adding a new landmark to FAST-SLAM can be difficult decision to make just as EKF-based algorithms. This is especially true when an individual measurement is insufficient to constrain the new landmark in all dimensions. In the measurement function $h({}^n\mathbf{x}_k^R, \mathbf{z}_k)$ is invertible, however, a single measurement is sufficient to initialize a new landmark. Each observation defines a Gaussian:

$$\mathbf{z}_k \sim N(\hat{\mathbf{z}}_k + \Lambda_k(\mathbf{z}_{k,i} - {}^{n}\mu_{k-1,i}), \mathbf{R}_k) \tag{5.39}$$

This Gaussian can be written expilicitly as:

$$\frac{1}{\sqrt{|2\pi\tilde{\mathbf{Z}}_{k,i}|}} \exp[-\frac{1}{2}(\mathbf{z}_k - \hat{\mathbf{z}}_k - \Lambda_k(\mathbf{z}_{k,i} - {}^{n}\mu_{k-1,i}))^{\mathrm{T}} \mathbf{R}_k^{-1}(\mathbf{z}_k - \hat{\mathbf{z}}_k - \Lambda_k(\mathbf{z}_{k,i} - {}^{n}\mu_{k-1,i}))] \tag{5.40}$$

We define a function **Q** to be equal to the negative of the exponent of this Gaussian:

$$\mathbf{Q} = \frac{1}{2}(\mathbf{z}_k - \hat{\mathbf{z}}_k - \Lambda_k(\mathbf{z}_{k,i} - {}^{n}\mu_{k-1,i}))^{\mathrm{T}} \mathbf{R}_k^{-1}(\mathbf{z}_k - \hat{\mathbf{z}}_k - \Lambda_k(\mathbf{z}_{k,i} - {}^{n}\mu_{k-1,i})) \tag{5.41}$$

The second derivative of **Q** with respect to $\mathbf{z}_{k,i}$ will be the inverse of the covariance matrix of the Gaussian in landmark coordinates.

$$\frac{\partial Q}{\partial z_{k,i}} = -(\mathbf{z}_k - \hat{\mathbf{z}}_k - \Lambda_k(\mathbf{z}_{k,i} - {}^{n}\mu_{k-1,i}))^{\mathrm{T}} \mathbf{R}_k^{-1} \Lambda_k \tag{5.42}$$

$$\frac{\partial^2 Q}{\partial z_{k,i}^2} = \Lambda_k^T \mathbf{R}_k^{-1} \Lambda_k \tag{5.43}$$

Consequently, an invertible observation can be used to create a new landmark as follows.

$${}^{n}\mu_{k-1,i} = h^{-1}({}^{n}\mathbf{x}_k^R, \mathbf{z}_k) \tag{5.44}$$

$$^{n}\Sigma_{k,i} = \Lambda_k^T \mathbf{R}_k^{-1} \Lambda_k \tag{5.45}$$

$$^{n}\hat{w}_k = P_0 \tag{5.46}$$

In practice, a simpler initialization procedure also works well. Instead of computing the correct initial covariance, the covariance can be computed by setting the variance of each landmark parameter to a high value and incorporating the first observation. Higher values of K lead to closer approximations of the true covariance, but can also lead to numerical instability.

$$^{n}\mu_{k-1,i} = h^{-1}(^{n}\mathbf{x}_k^R, \mathbf{z}_k) \tag{5.47}$$

$$^{n}\Sigma_{k,i} = \mathbf{K}\,\mathbf{I} \tag{5.48}$$

In appendix B, the FAST-SLAM algorithm with unknown data association is represented. This algorithm incorporates a single observation for every control. This choice is for notational simplicity only. Multiple readings can be incorporated per time step by processing each observation sequentially. The weight for each particle is equal to the product of weight s due to each observation considered alone. Incorporating multiple observations per time step will increase both the accuracy of data association and the accuracy of the resulting map" [11].

5.8 Summary

In this chapter we discussed an alternative algorithm to deal with SLAM problem. This algorithm is called FAST-SLAM. FAST-SLAM is a method based on Rao-Blackwellised Particle filtering in a Bayesian network that was first introduced by Montemerlo. Since landmarks are mutually independent, observation of each landmark can be incorporated to the map with no correlation with other landmarks in the map. A factorization of particle filter can make the FAST-SLAM algorithm very straight forward since the problem become low dimensional. This factorization lets the path of robot be estimated by particles while there will be simple EKF filters as many as landmarks in the map. The landmark position will be estimated by EKFs according to the path of rbot which was estimated by particles in advance. The result of factorization can be shows as follows.

$$P(\mathbf{X}_k^R, \mathbf{m} \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R) = P(\mathbf{X}_k^R \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R)\, P(\mathbf{m} \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R)$$

$$= P(\mathbf{X}_k^R \mid \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R) \times \prod_{i=1}^{M} P(\mathbf{m}_i \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R)$$

RBPF in FAST-SLAM fulfills the task in four major steps; *State sampling, Landmark estimation update, Importance weight,* and *updating importance sampling.* This process is basd on drawing samples from the proposal distribution that is bing matched with a target distribution. Samples have higher weights in the regions that the proposal distribution is larger than the target distribution. Drawing samples from the proposal distribution simplifies the process rather than drawing samples form the target distribution.

Computational complexity is not at the level of concern as of EKF-SLAM any more since it is not quadratic. Instead, it appears to be linear while it is even simpler with the logarithmic representation of FAST-SLAM. With the structure of FAST-SLAM algorithm, the data association is not a major concern since it posses multiple hypothesis data association property. Unlike EKF-SLAM, an observation data is not determined for each observation and is corrected many times before being incorporated to the map.

Chapter 6

Simulations and Results

6.1 A comparison between EKF-SLAM and FAST-SLAM

This chapter includes number of original simulation experiments to show characteristic features of SLAM and FAST-SLAM methods. As it was described before there are two different filters EKF and RBPF considered for the same trajectory and landmarks locations for a particular SLAM problem for standard outdoor robotic vehicle. According to figure 6.1 the specifications of the vehicle have been considered as A=4.0m, λ=3.0m, W=0.75m, and B=0.4m. Vehicle velocity is 2.0 m/s and the maximum range for the mounted sensor is 100.00 m. In this simulation, the noise in which the system is subject to, is at first assumed Gaussian, white with zero-mean and independent. Then a simulation according to this assumption is made using EKF algorithm.

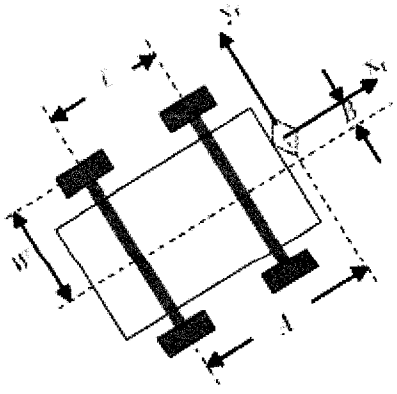


Figure 6.1: Dimentions of the standard robotic vehicle

The result in figure 6.2 shows that the vehicle stays near the path with a highly acceptable estimation. The dotted line shows the odometer reading. Figure 6.3 illustrates the same trajectory in which the noise is not Gaussian anymore but instead a mixture of exponential and Gaussian implication has been applied for the case. This figure clearly shows how fragile the EKF algorithm can be due to non-Gaussian implication. Since the EKF is based on first order Taylor approximation, it will be no more accurate for an optimal solution to SLAM problem in this situation. Notice that almost in the middle of the trajectory, EKF can no longer manage the non-Gaussian error of the system. The catastrophic result is obvious when the vehicle is almost at point (x=325, y=200) with respect to the reference coordinate system (global coordinate). The true path is shown by the dashed line. It can be seen that how much difference along both x and y-axes are made after the robot arrives to the end of the trajectory. Another reason of this failure might be due to the fact that EKF can not deal with the non-linearity of the observations. If, for some reason the observation information is wrong, it will be embedded to the filter before fusing the data to the map and of course with no more considerations into it. Since EKF algorithm does not do any correction at this step, the wrong data will be one part of the map and can be never corrected. If more wrong observations in the next steps are obtained, that the possibility is high, the error highly accumulates and the EKF diverges. Figure 6.4 illustrates the same situation using RBPF. In this case, the non-Gaussian assumption is considered as well. Likewise, the filter performs a good estimation of the path and landmark readings. This diagram is the result of 100 particles in the MATLAB code for observing each individual landmark. For the robot position tracking, 100 particles are considered as well. This figure is a proof of RBPF success for a situation in which the system noise is not Gaussian and at the same time the motion model is non-linear. It is obvious that

the data association is in no more concern. The result shows how well the RBPF deals with a situation in which EKF could not as indicated in figure 6.3.

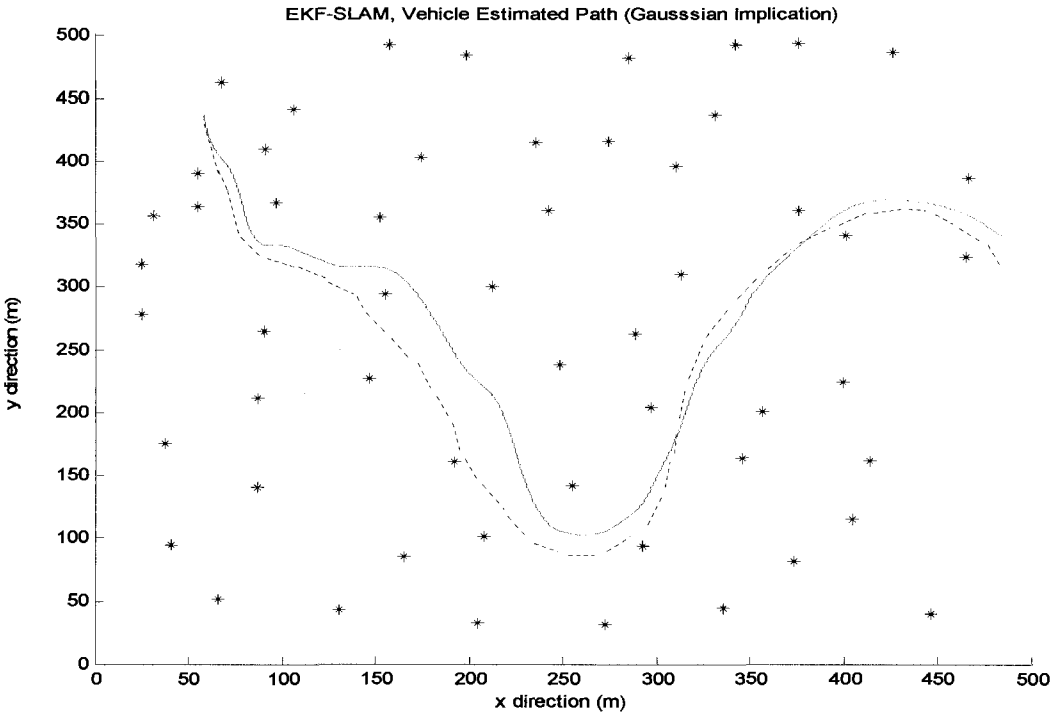


Figure 6.2: EKF-SLAM under Gaussian conditions

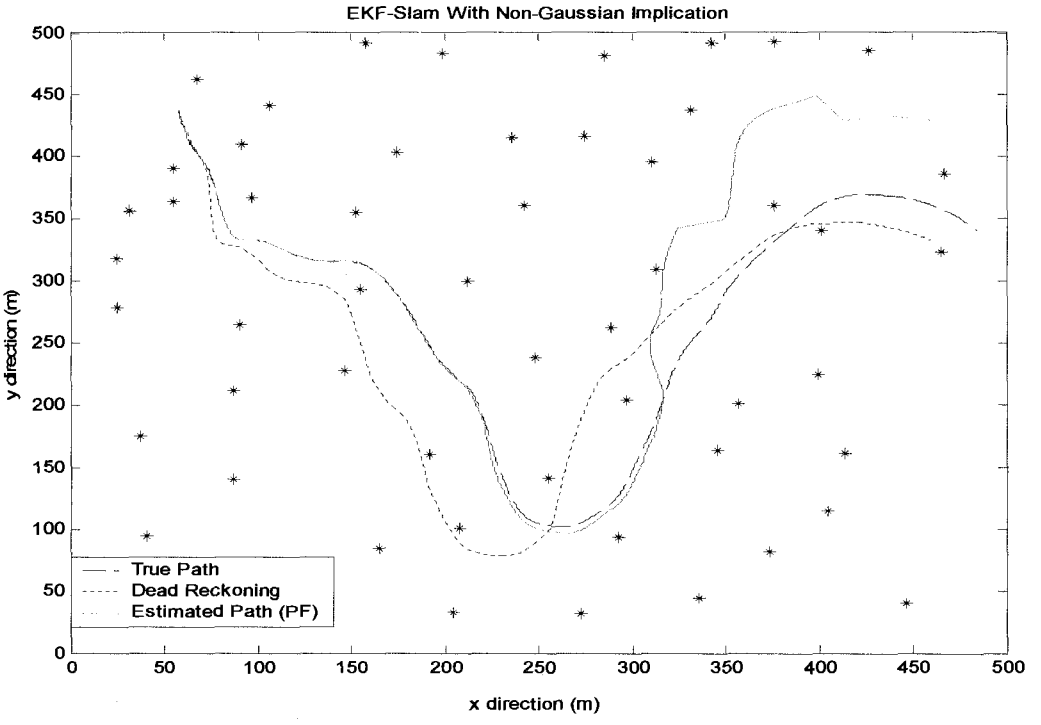


Figure 6.3: EKF-SLAM with non-Gaussian conditions

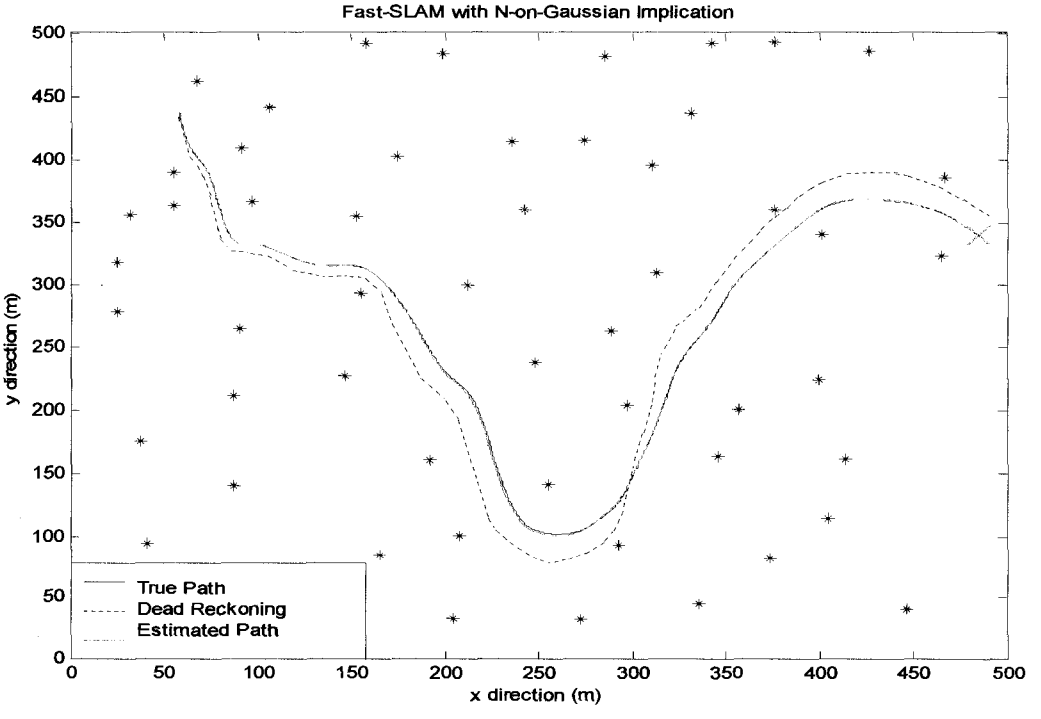


Figure 6.4: FAST-SLAM with non-Gaussian conditions

In figures 6.5 to 6.10 the position and orientation estimation errors using both EKF with Gaussian and non-Gaussian implication and RBPF with non-Gaussian implication for a non-linear system are compared. While the estimated position error in case of EKF-SLAM with non-Gaussian assumptions increases more than 90 meters, the FAST-SLAM limits the position estimation error up to 1 meter in the worst case and the average stays around 0.40 meters. This makes the result very much appropriate for the SLAM solution in case of non-Gaussian implication with non-linearity of the motion equations. Furthermore, while the average of orientation error in figure 6.9 is around 0.025 radian, this average does not have a pick more than 0.015 radian at the beginning of the trajectory and it goes down to the average of 0.005 radians for the rest of the trajectory while it gives the same promising results with the EKF-SLAM under Gaussian conditions.

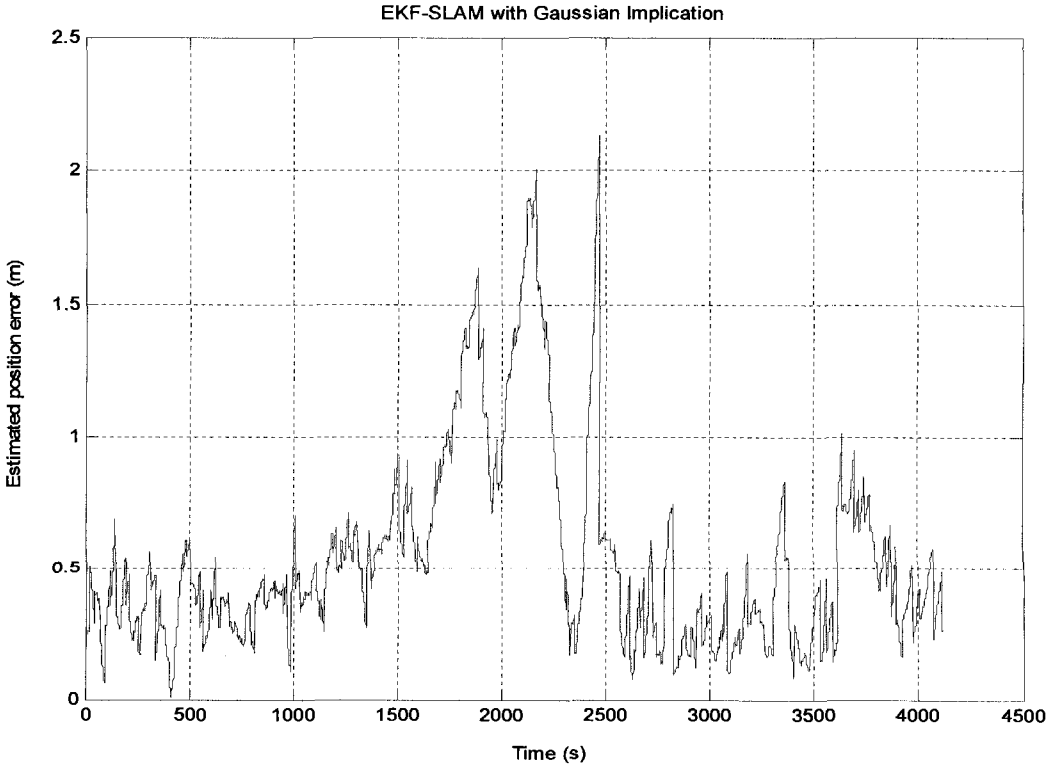


Figure 6.5: Position estimation error for EKF-SLAM with Gaussian conditions

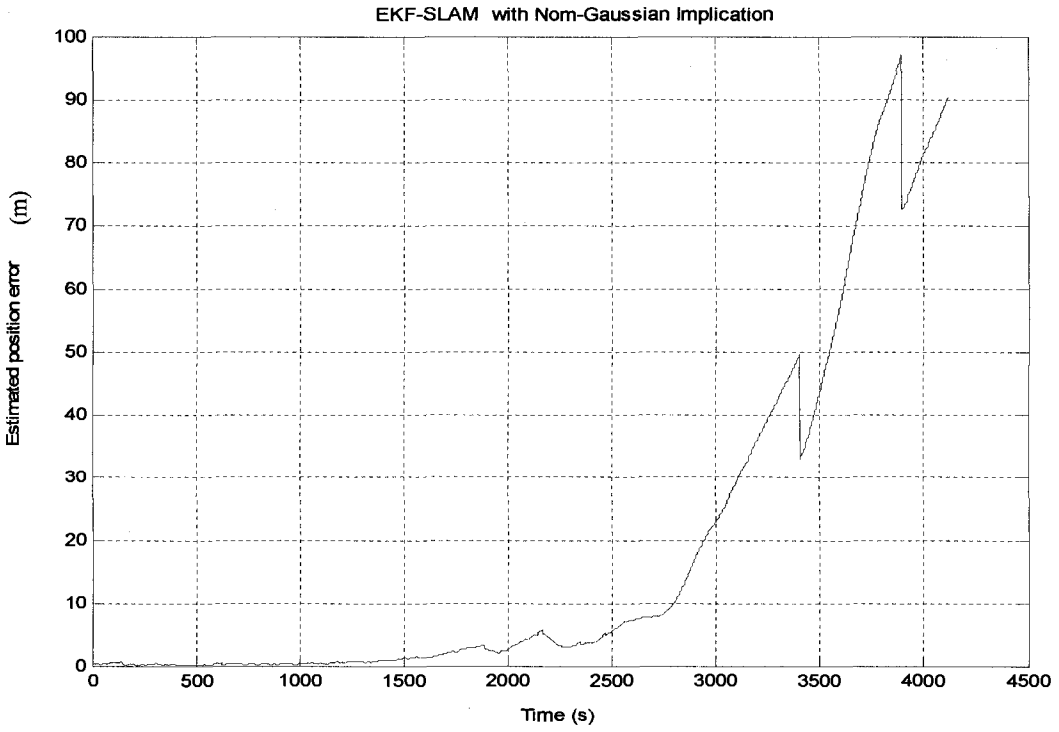


Figure 6.6: Position estimation error for EKF-SLAM with non-Gaussian conditions

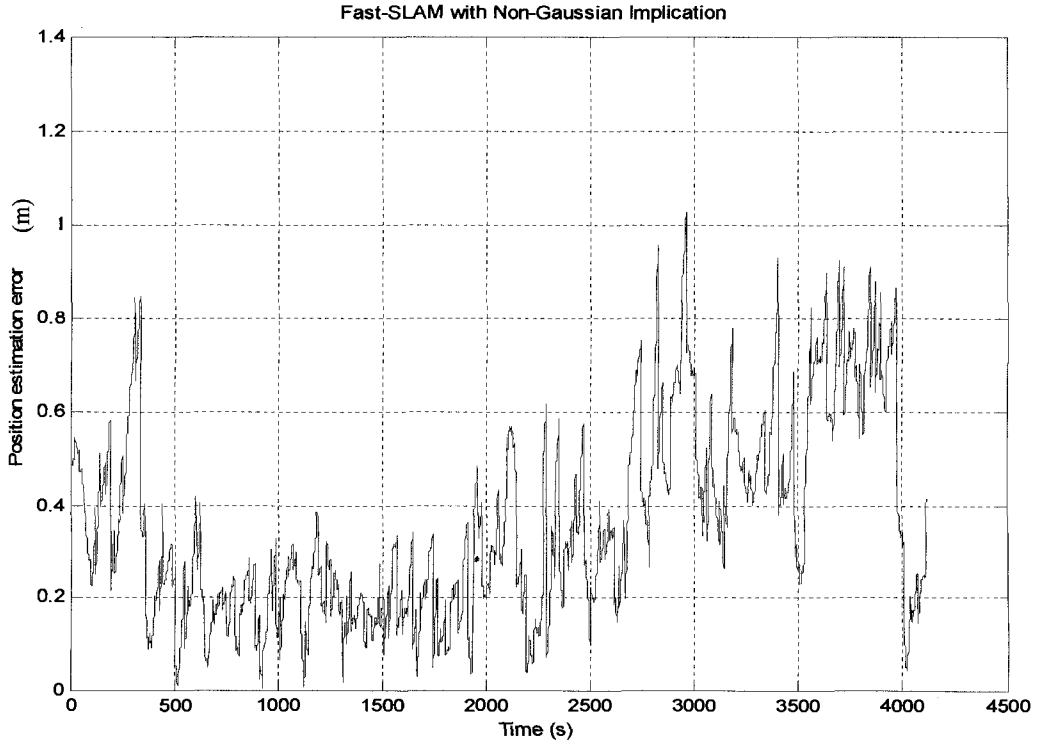


Figure 6.7: Position estimation error for FAST-SLAM with non-Gaussian conditions

Figure 6.11 again indicates a simulated result of FAST-SLAM performance versus EKF-SLAM in terms of increasing control noise of the process. In this simulation the root mean square position (RMS) error for both filters are illustrated. By increasing the level of control noise up to 0.3, the RMS position error of FAST-SLAM does not make a big difference while this error increases substantially for EKF-SLAM and consequently causes the filter to diverge. The RMS error for both filters was computed over 10 different runs in the simulation with two different levels of odometric noise. At each level, the other specifications and parameters were held constants.

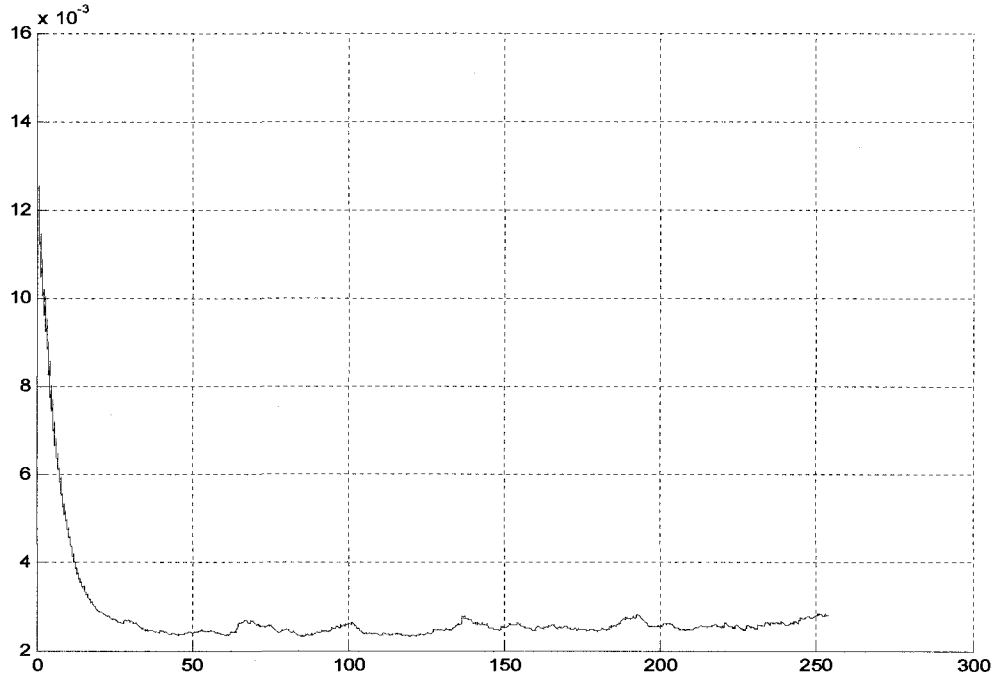


Figure 6.8: Orientation estimation error for EKF-SLAM with Gaussian conditions

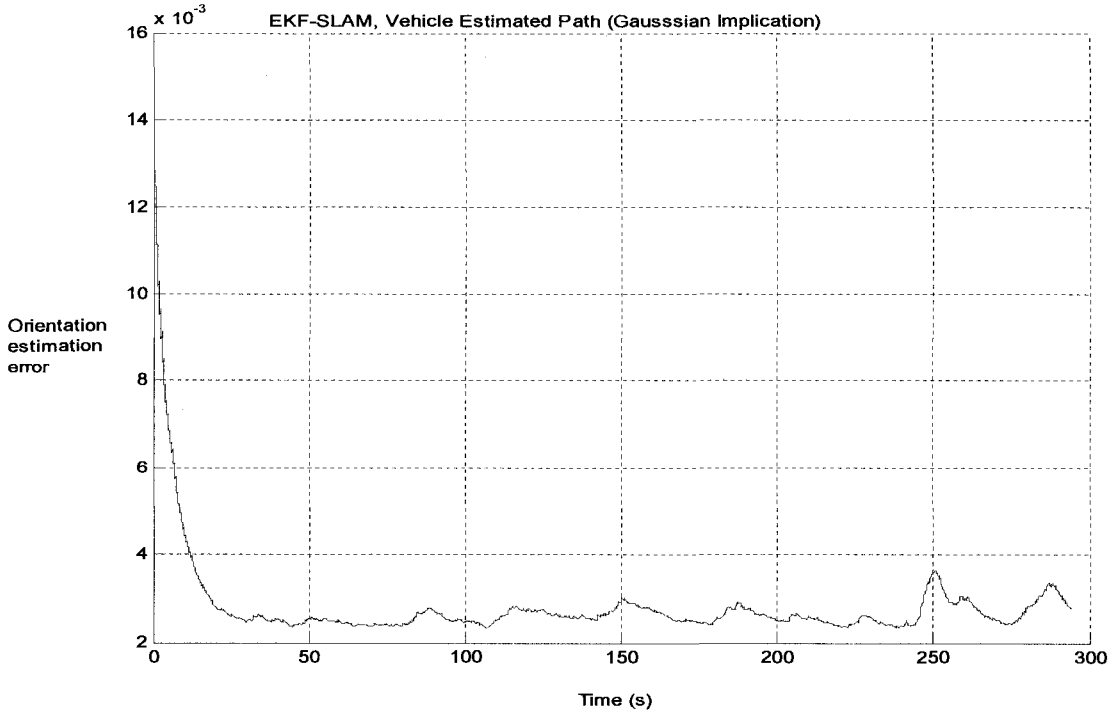


Figure 6.9: Orientation estimation error for EKF-SLAM with non-Gaussian conditions

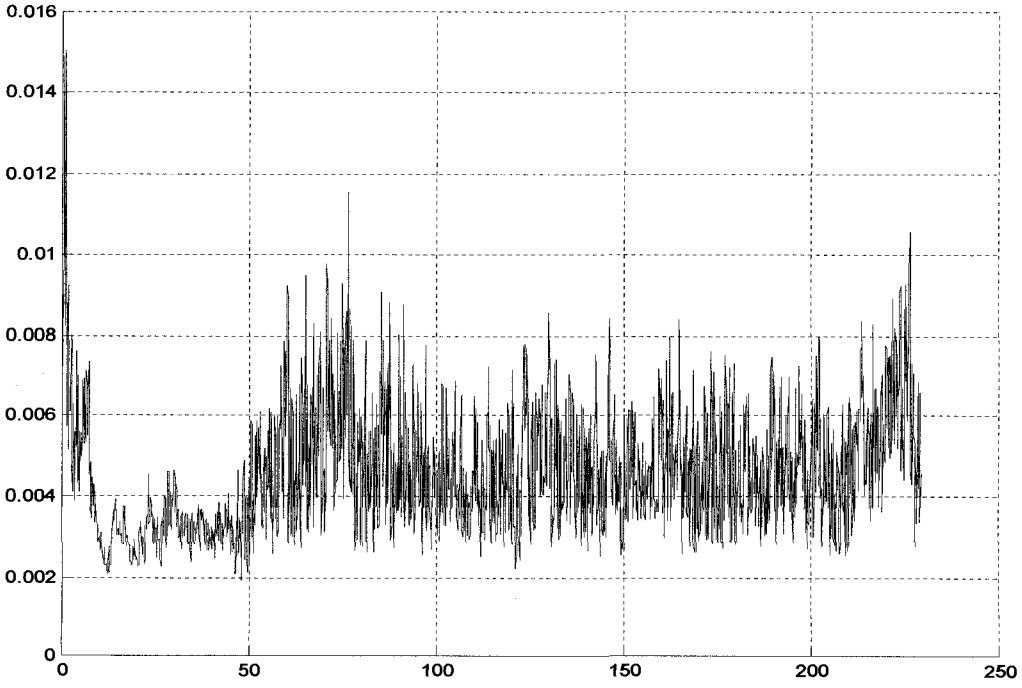


Figure 6.10: Orientation estimation error for FAST-SLAM with non-Gaussian conditions

Figure 6.12 is another proof of the accuracy of FAST-SLAM in terms of RMS position error compare to that of EKF-SLAM. As number of incorporated particles increases, the RMS position error difference between EKF-SLAM under Gaussian condition and RBPF under non-Gaussian condition becomes smaller. It is obvious that with a number of 100 particles the difference between RMS position errors of both filters become reasonable. In other words, 100 particles suffice to make the accuracy of FAST-SLAM under non-Gaussian condition as perfect as EKF-SLAM under Gaussian condition. The interesting fact is that FAST-SLAM with 100 particles requires an order of magnitude fewer parameters than the EKF-SLAM in order to achieve this level of accuracy. This result suggests that only with a few hundred of particles in FAST-SLAM the level of accuracy is high and there is no need of relatively high number of particle involvement.

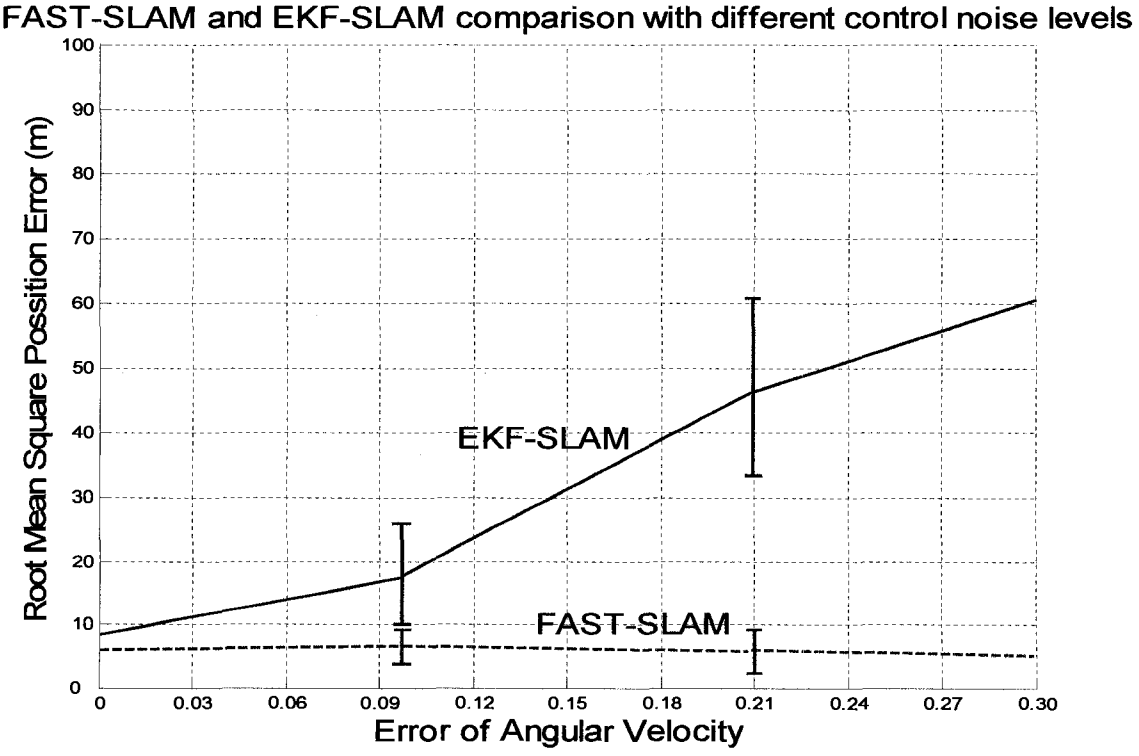


Figure 6.11: RMS position error for different levels of control noise

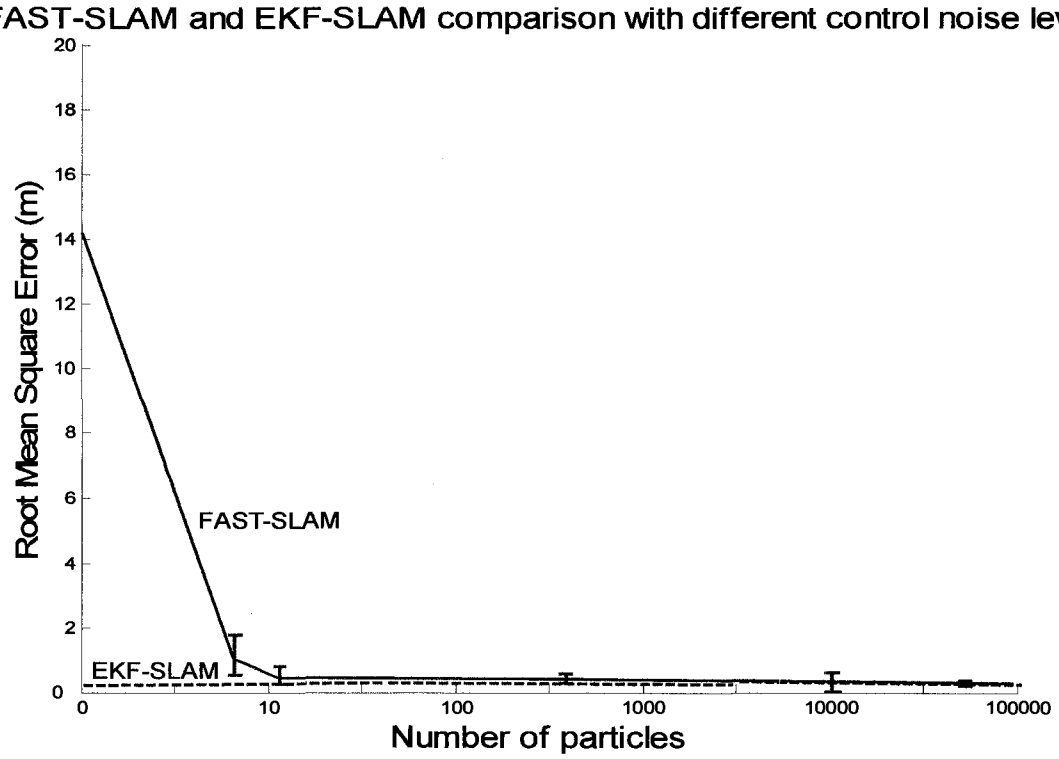


Figure 6.12: RMS position error for different amount of particles

6.2 Single and Multiple Hypothesis Data Association

To compare the performance of FAST-SLAM (RBPF) and EKF-SLAM, with a dense map we have done severl simulations. Due to single hypothesis data association employed by EKF-SLAM even when it is under Gaussian conditions, if landmarks are very close to eachother, the performance of the filter defects. Figure 6.13 illustrates a situation bywhich EKF algoriyhm was used. In this figure a real road has been simulated and it is assumed that the range/bearing sensor is able to detect the curbs of the road as very nearby objects. In this simulation the sensor was able to detect approximately 6 point landmarks per meter. The distance range of the sensor was also changed and set to 50 meters. All other specifications of the vehicle and sensor were held the same as section 6.1.

As we can see in figure 6.13, the small circles are the waypoints that the robot is supposed to pass through. Figure 6.14 shows how much the robot is far away from the true path. This is due to the fact that EKF-SLAM can not deal with a dense map with very close landmarks. In fact the filter is fragile due to the single data association property. Figure 6.15 depicts the observation result. From this figure, it is obvious that the filter performance is not acceptable. Figures 6.16 to 6.18 show the same scenario with consideration of RBPF. From these figures, it is evident that FAST-SLAM deals with the situation very reasonably. The mapping by RBPF in figure 6.18 is very vivid and the robot has succeeded to observe the environment applying the RBPF algorithm and finding its way through the waypoints. The multiple hypothesis uncertainty mechanism property of FAST-SLAM, makes the robot to cope the uncertainty due to close landmarks. This situation has been simulated by extending the road for both filters through figures 6.19 to 6.24.

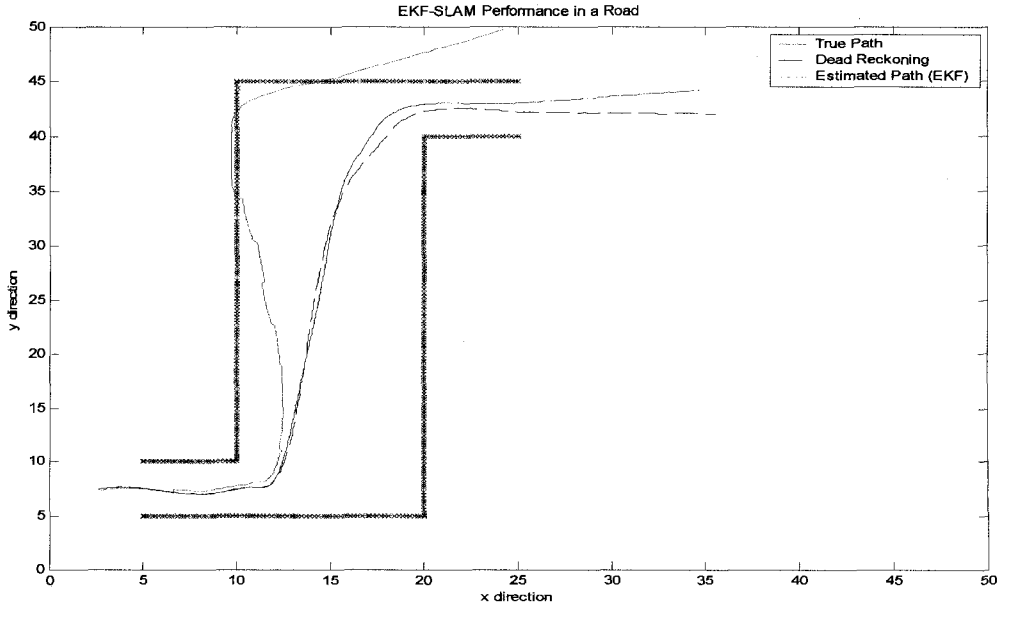


Figure 6.13: EKF-SLAM performance on a road (650 detected landmarks)

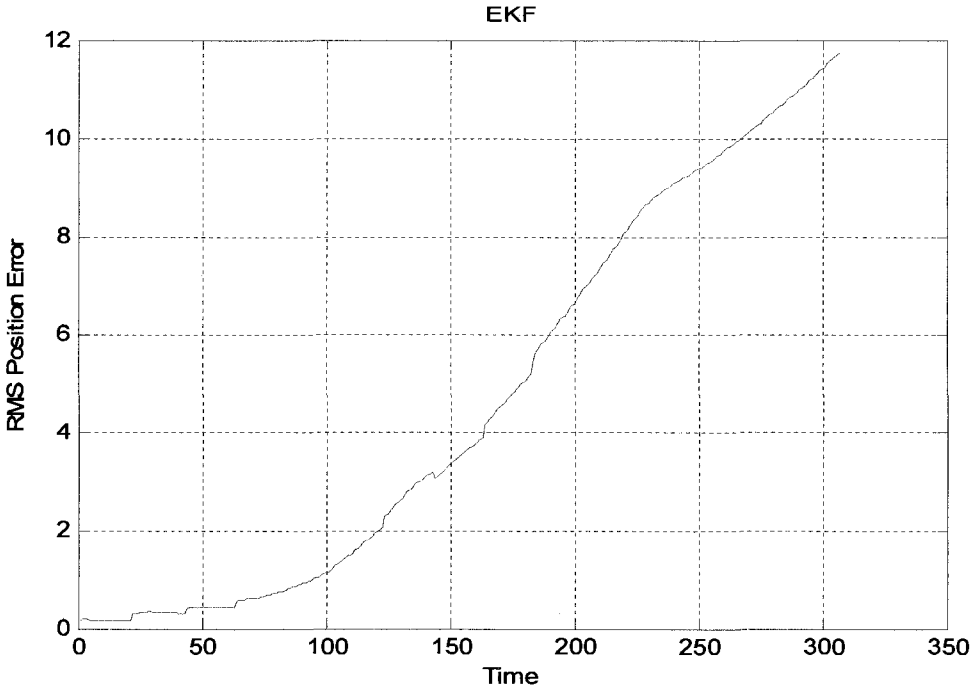


Figure 6.14: RMS error for EKF-SLAM on a road

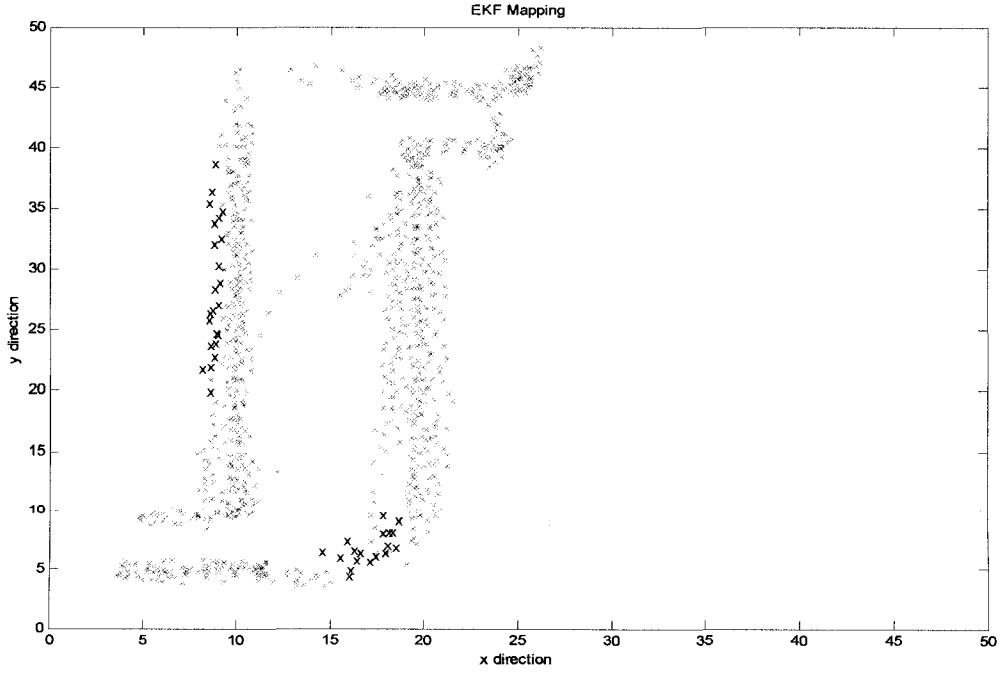


Figure 6.15: Mpping result by EKF-SLAM (650 detected landmarks)

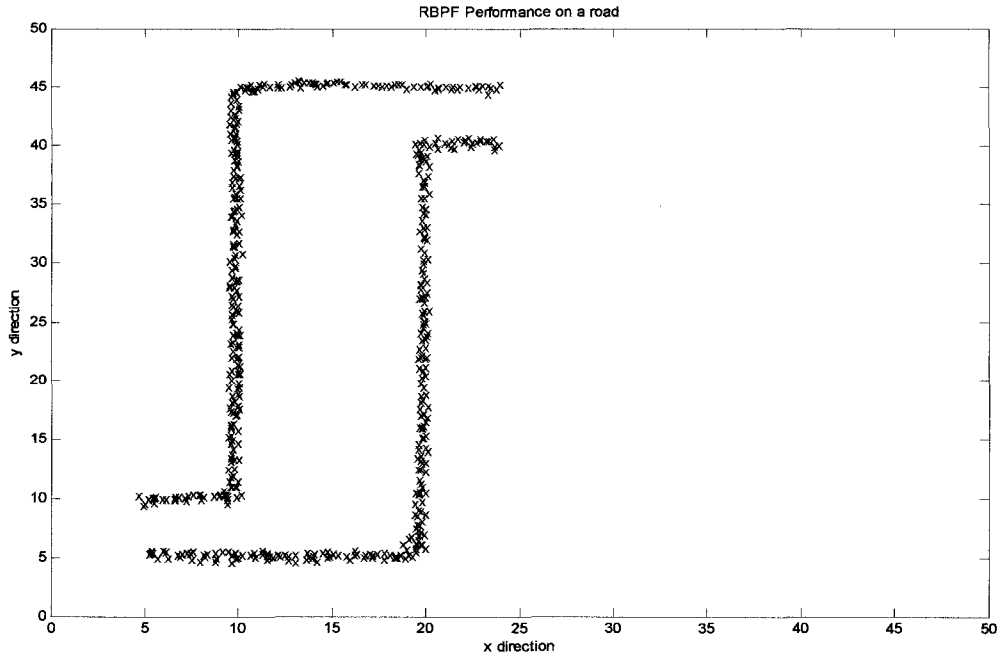


Figure 6.16: Mpping result by FAST-SLAM (650 detected landmarks)

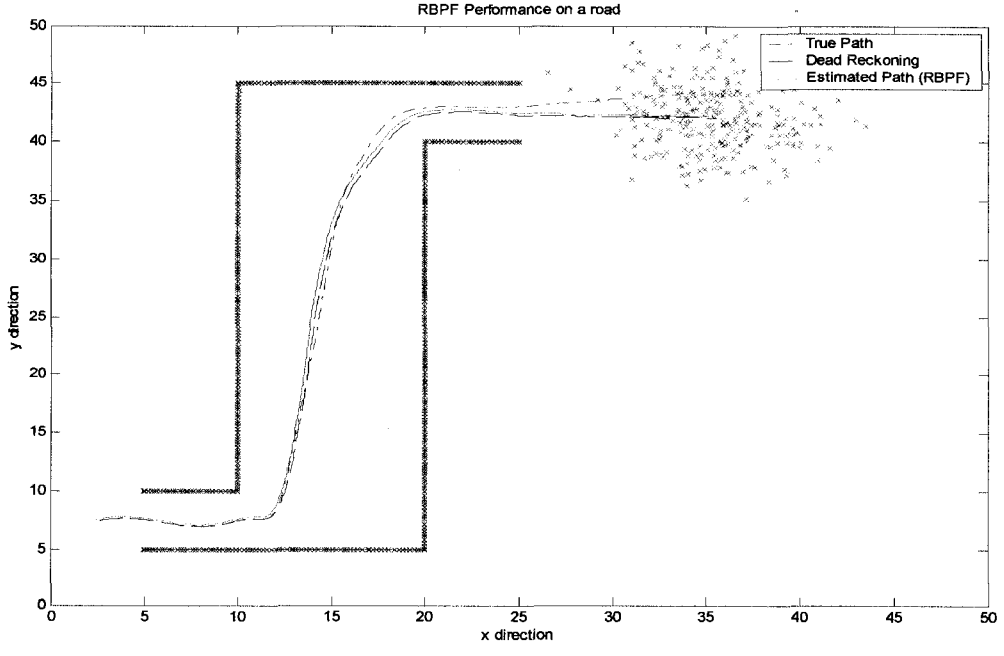


Figure 6.17: FAST-SLAM performance on a road (650 detected landmarks)

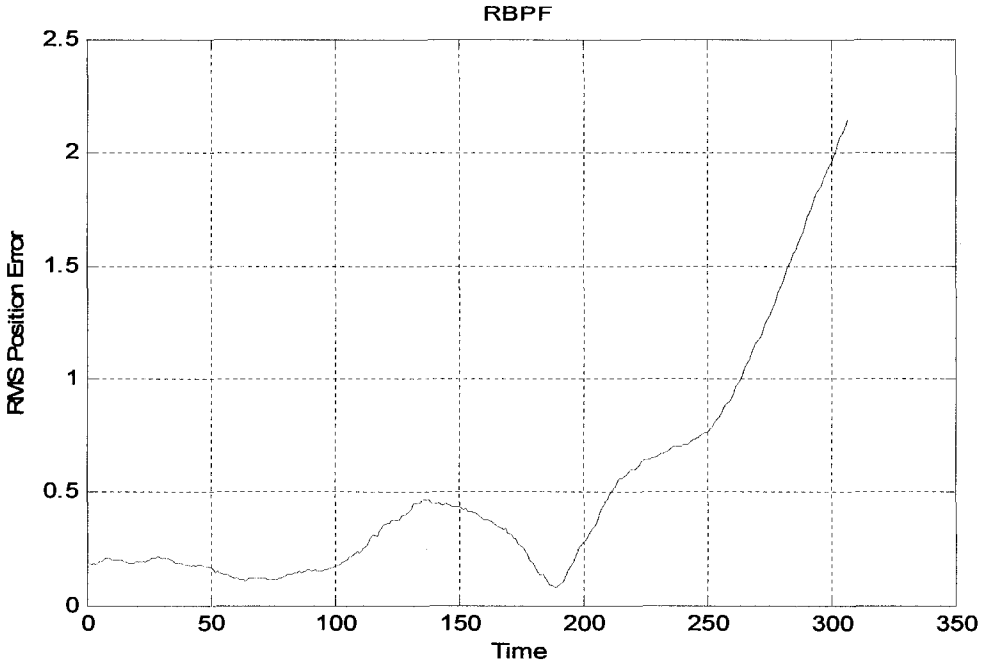


Figure 6.18: RMS error for FAST-SLAM on a road (650 detected landmarks)

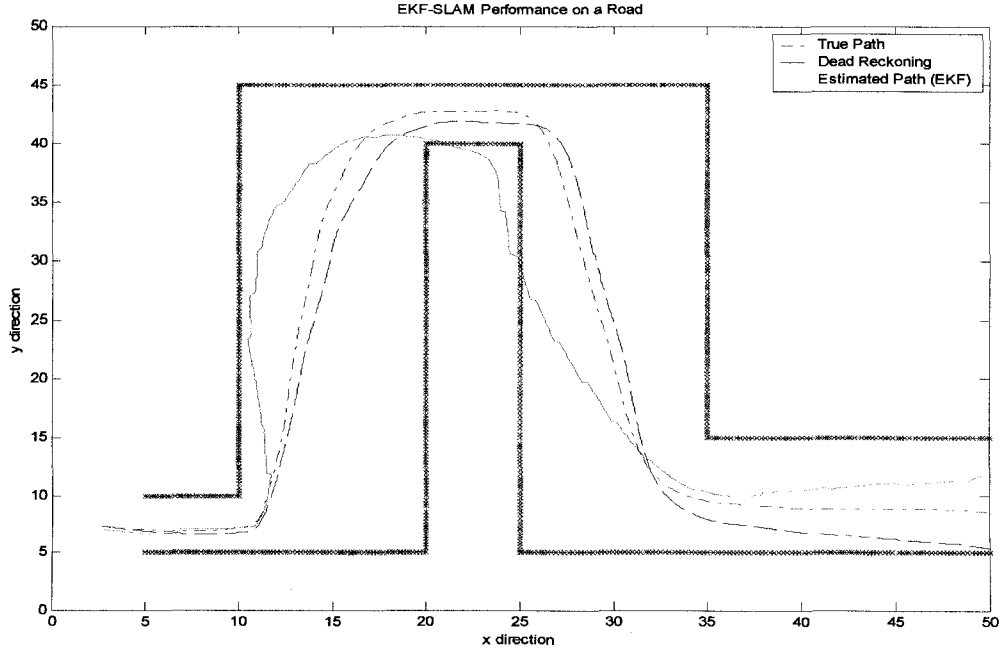


Figure 6.19: EKF-SLAM performance on a road (1350 detected landmarks)

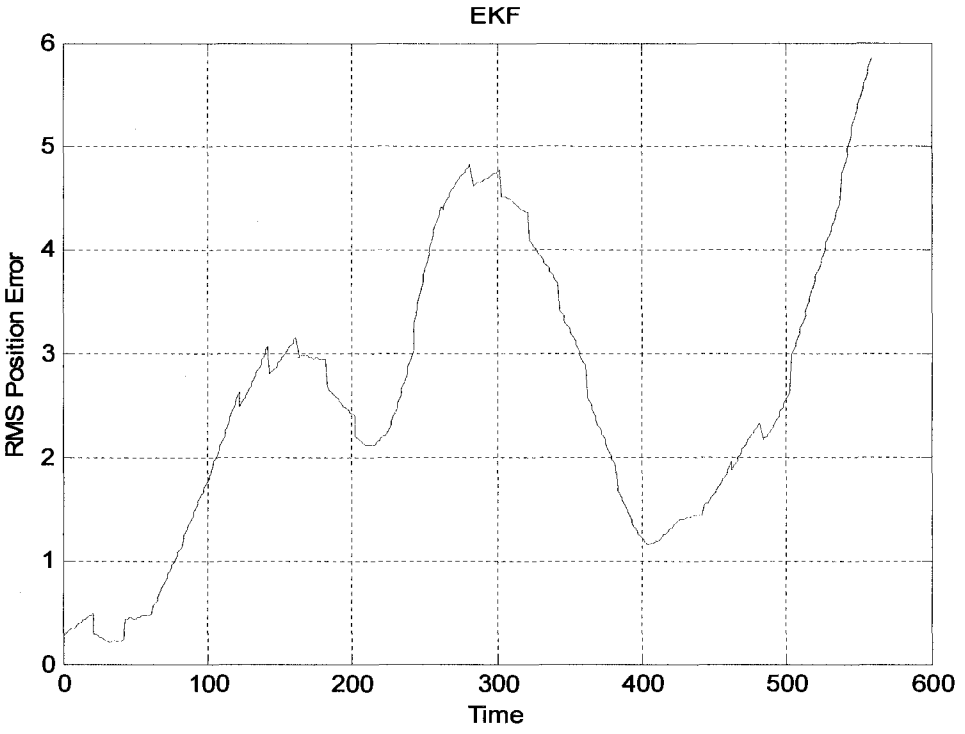


Figure 6.20: RMS error for EKF-SLAM on a road (1350 detected landmarks)

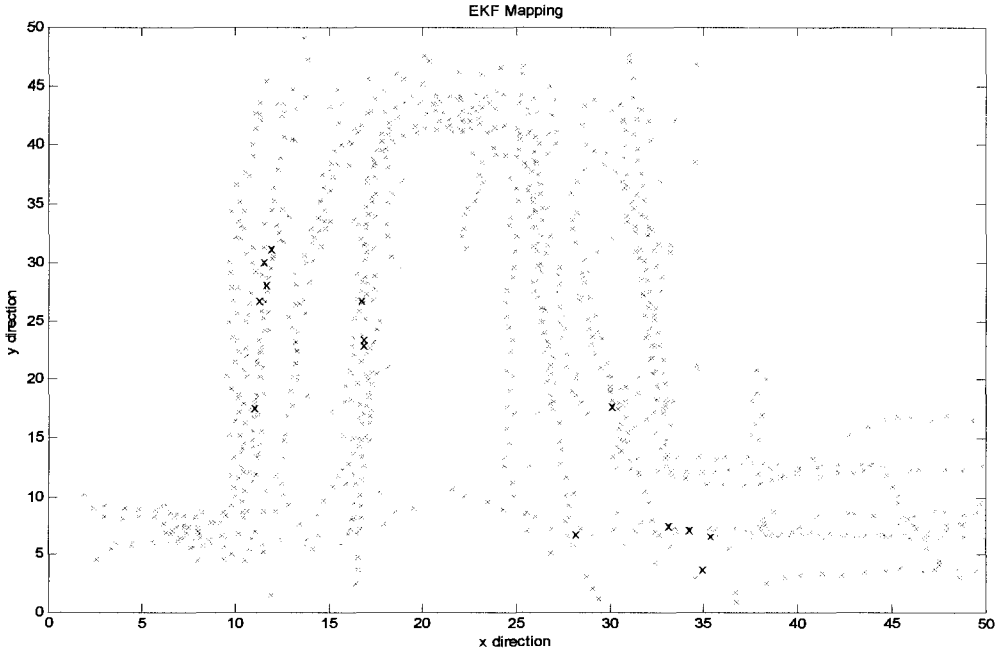


Figure 6.21: Mpping result by EKF-SLAM (1350 detected landmarks)

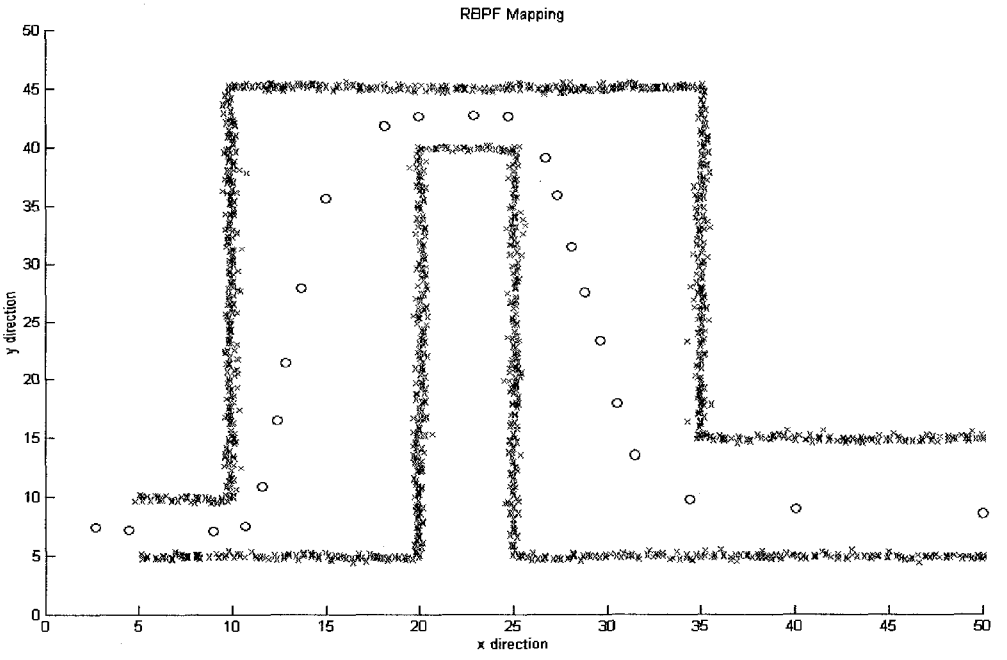


Figure 6.22: Mpping result by FAST-SLAM (1350 detected landmarks)

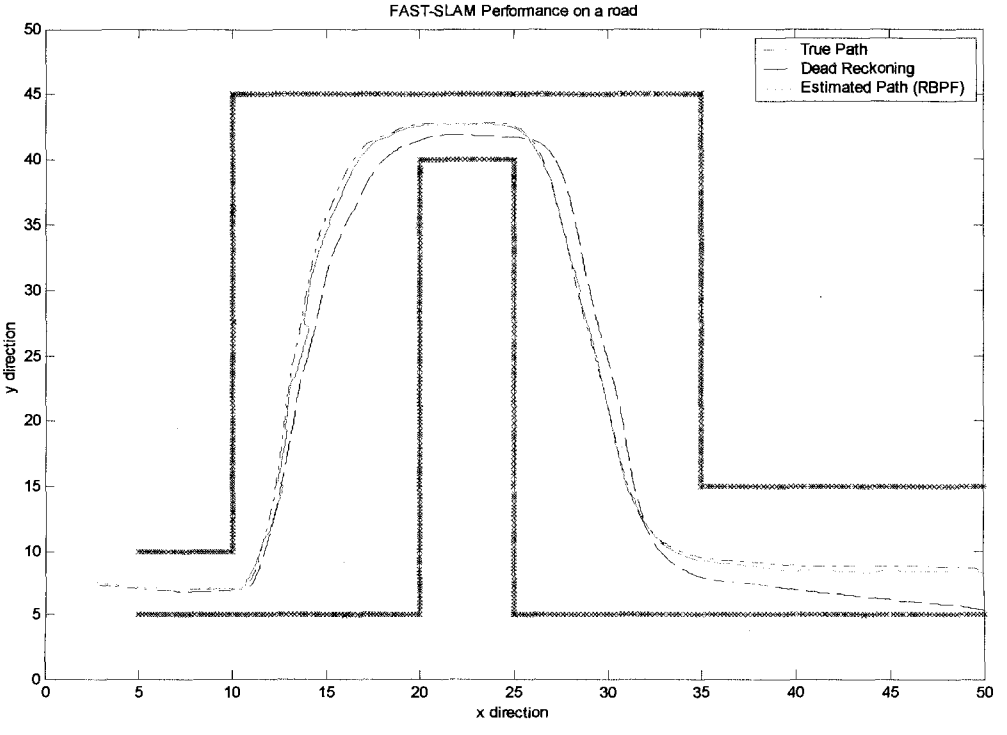


Figure 6.23: FAST-SLAM performance on a road (1350 detected landmarks)

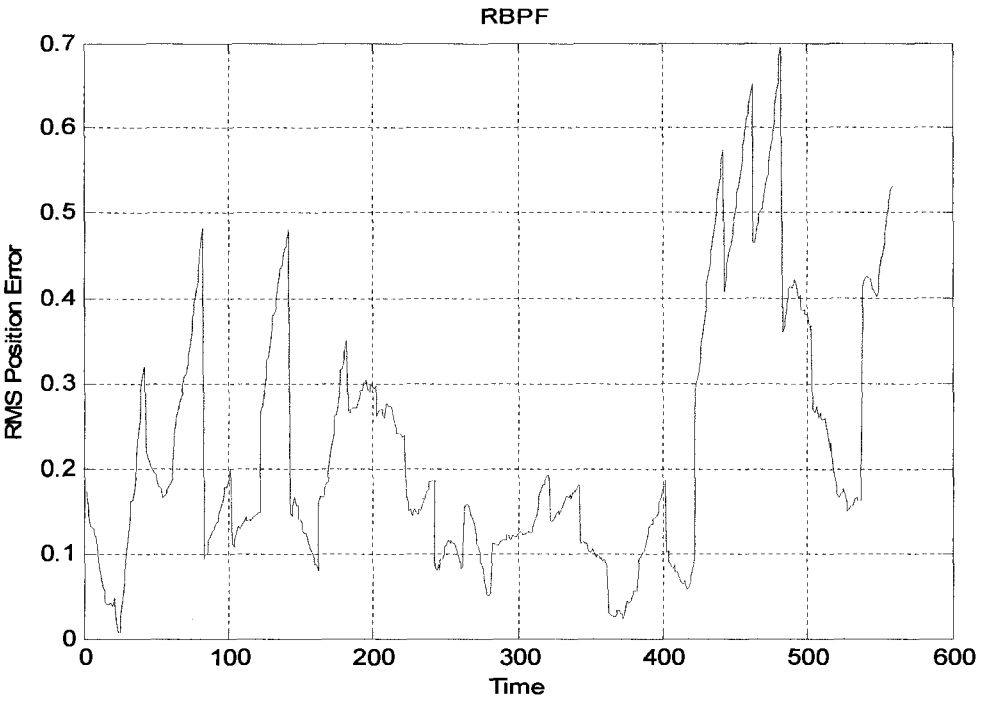


Figure 6.24: RMS error for FAST-SLAM on a road (1350 detected landmarks)

Another important reason rather than nearby landmarks that makes the observation in EKF very unreliable, is that, the number of landmarks affects the performance of the EKF. EKF is not able to deal with more than a few hundered landmarks due to growing number of elements in the main covariance matrix of the system. As was discussed in chapter 4, the main covariance matrix of the system grows quadratically as more landmarks are detected by the range/bearing sensor and the correlation between each pair of landmarks in the map becomes stronger and finally what remains is a solid map which makes no sense. Still the main reason that makes the EKF to diverge, is the nearby landmarks in the map wich makes the map very dense. Since EKF algorithm is based on single data association hypothesis, it can not handle the uncertainty of observing nearby landmarks. The result is then, what we see in figures 6.15 and 6.21 as very unclear maps.

Figure 6.25 depicts a true map of a rectangular object. Circles are the waypoints that the control is supposed to follow as part of robot's true path. Figure 6.26 shows the detectable landmarks by the sensor. These landmarks are very close to each other same as the simulated situations on a road. This time, the edges of the object are considered as nearby landmarks. The robot is supposed to travel in the environment and through the waypoints with identifying the object. The robot not only recognizes the object but also uses this object to localize itself. First the situation was simulated with EKF. Figures 6.27 and 6.29 demonstrate the map built using EKF and the orientation error. Results again show that EKF fails to map the landmarks as one whole object and consequently fails to localize itself successfully. Figures 6.28 and 6.30 show simulations of mapping and orientation error of the same situation with RBPF (FAST-SLAM). The simulations of two, three and multiple objects have been demonstrated through figures

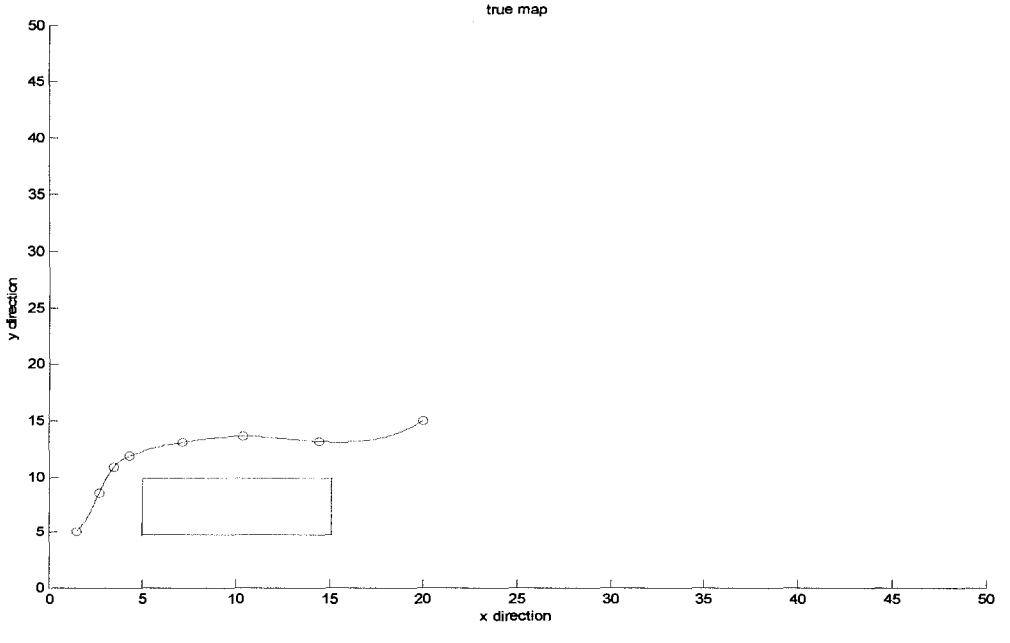


Figure 6.25: True map of a rectangular object

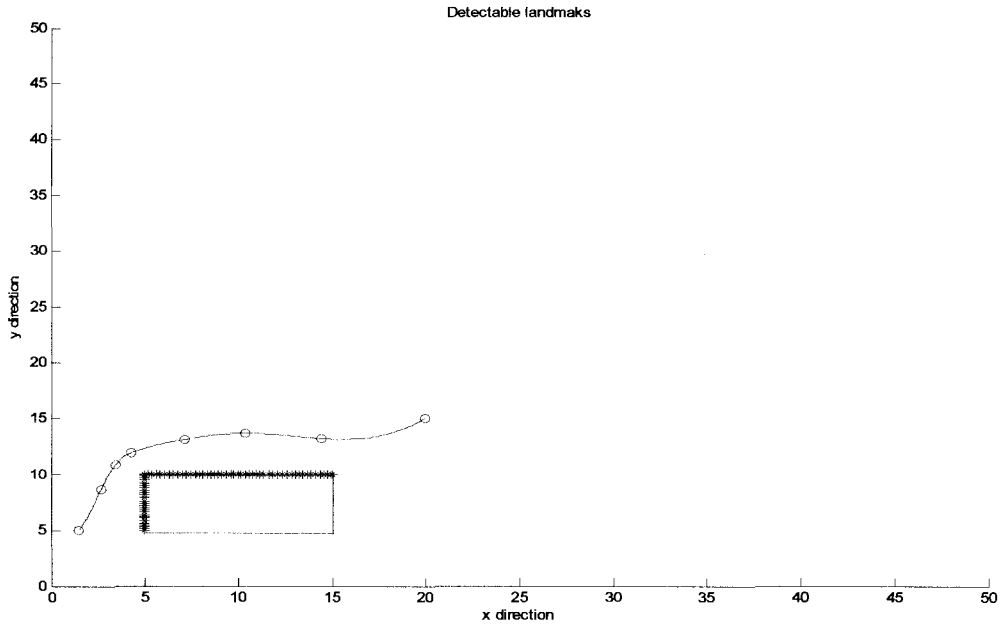


Figure 6.26: Detectable landmarks of a rectangular object (100 detectable landmarks)

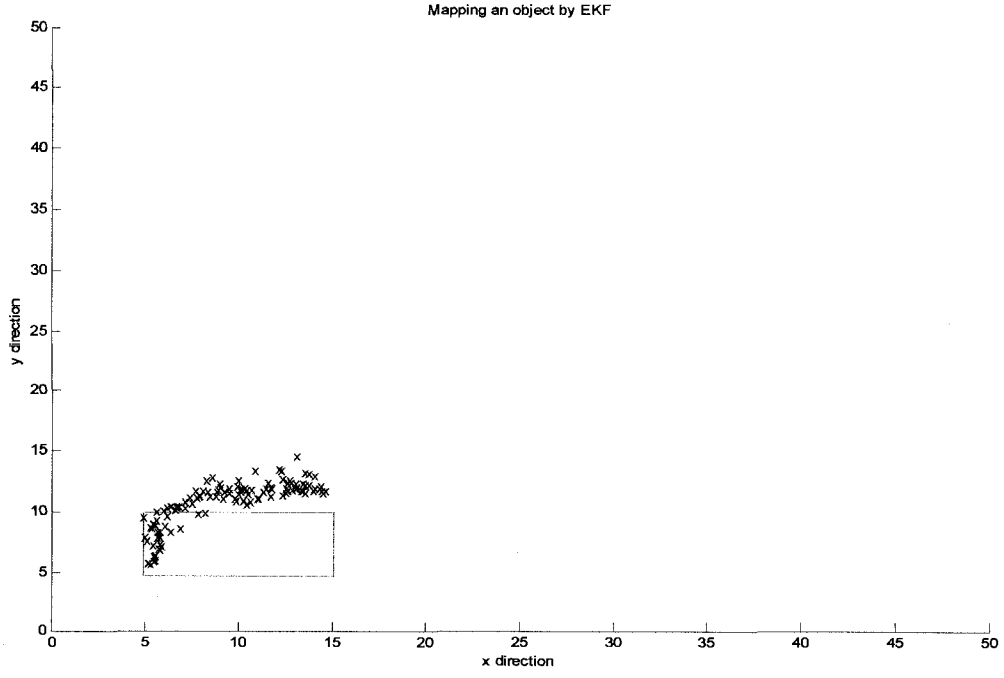


Figure 6.27: Mapping an object by EKF (100 detectable landmarks)

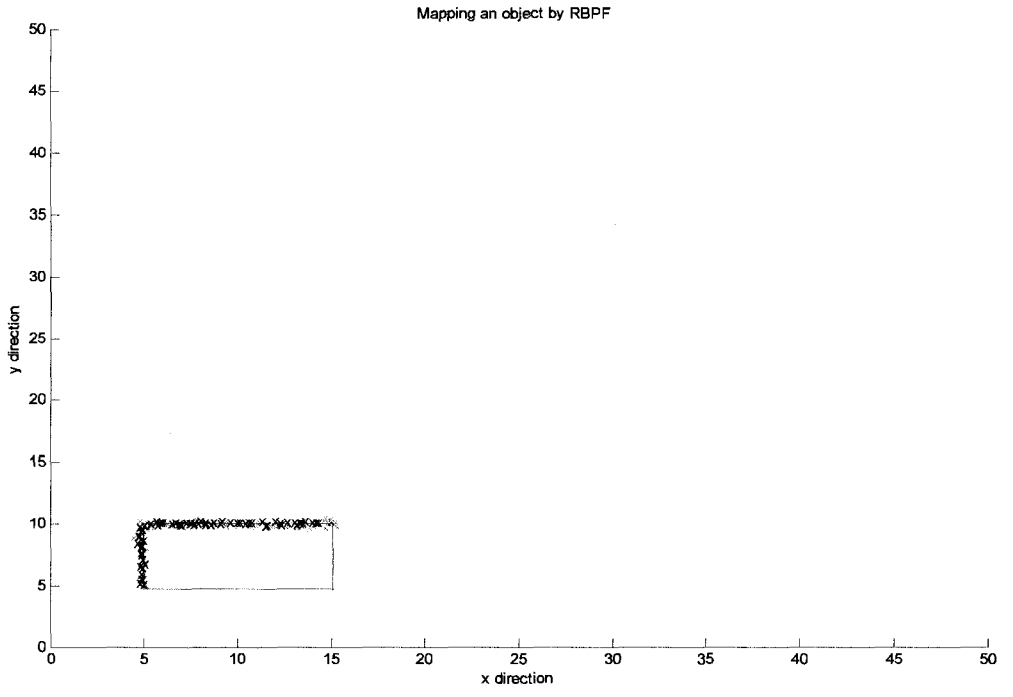


Figure 6.28: Mapping an object by FAST-SLAM (100 detectable landmarks)

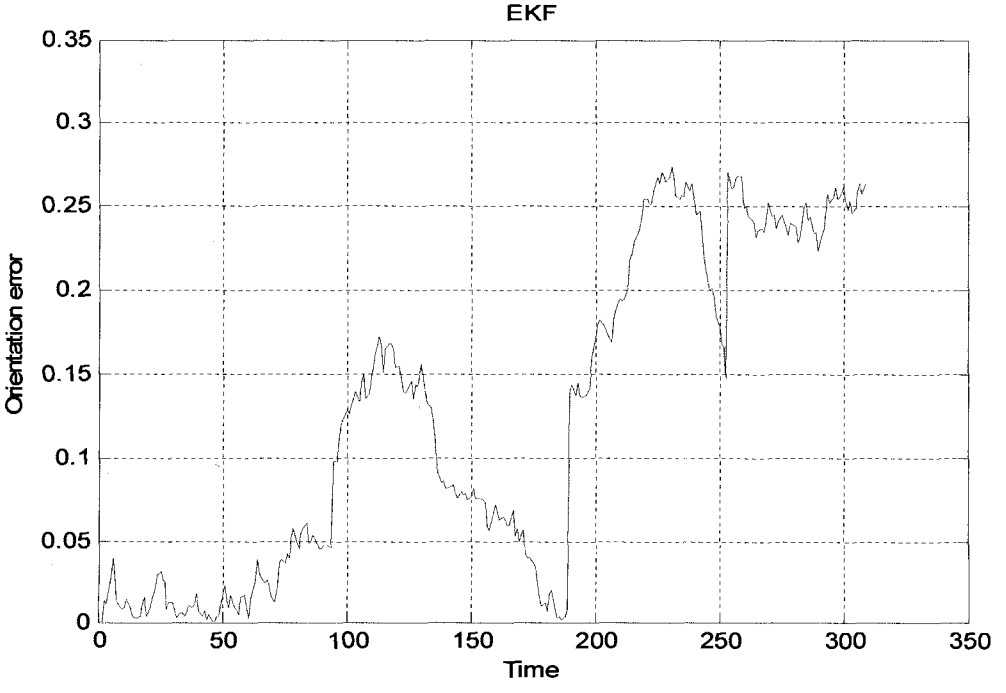


Figure 6.29: EKF orientation error (100 detectable landmarks)

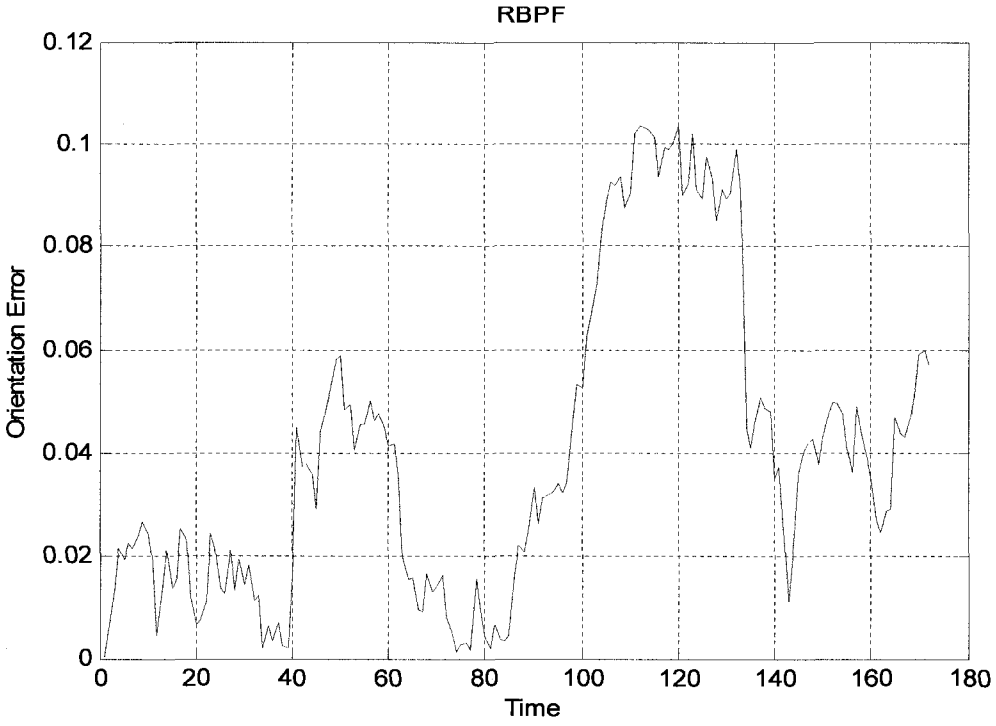


Figure 6.30: FAST-SLAM orientation error (100 detectable landmarks)

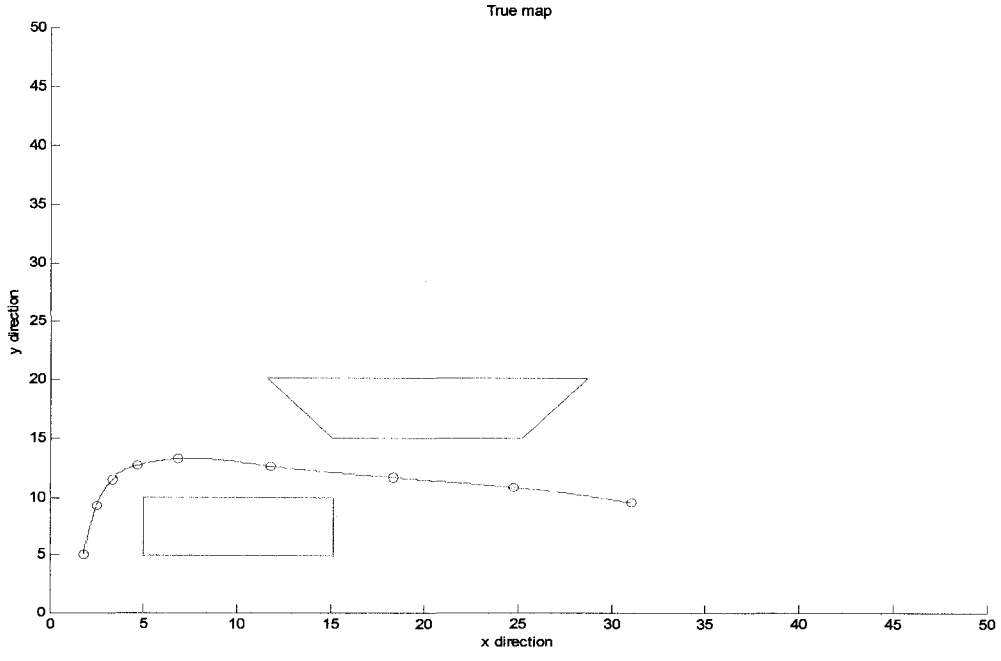


Figure 6.31: True map of two objects

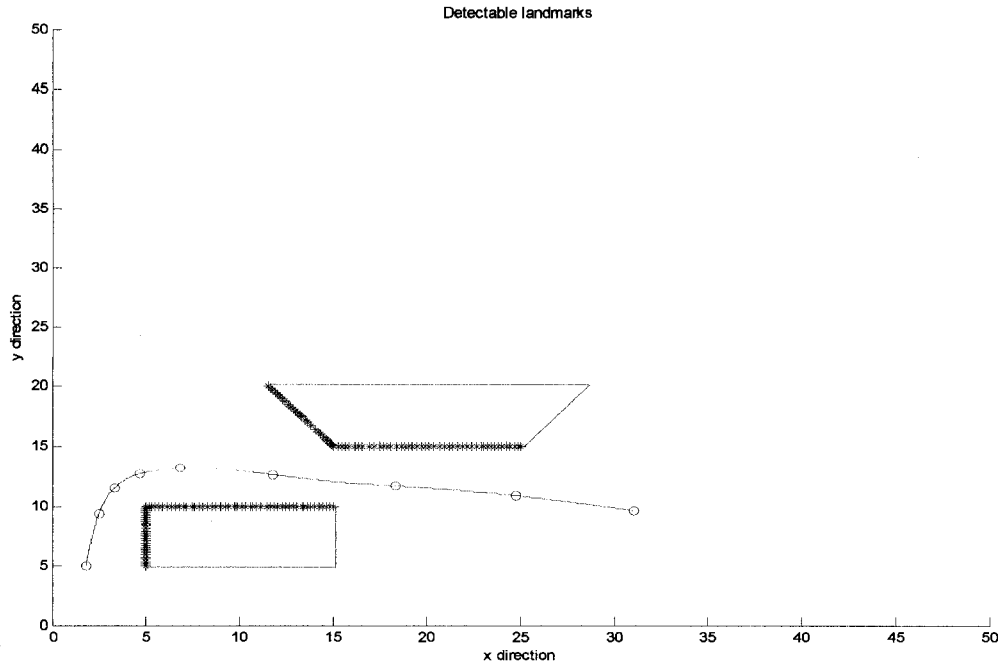


Figure 6.32: Detectable landmarks of two objects (220 detectable landmarks)

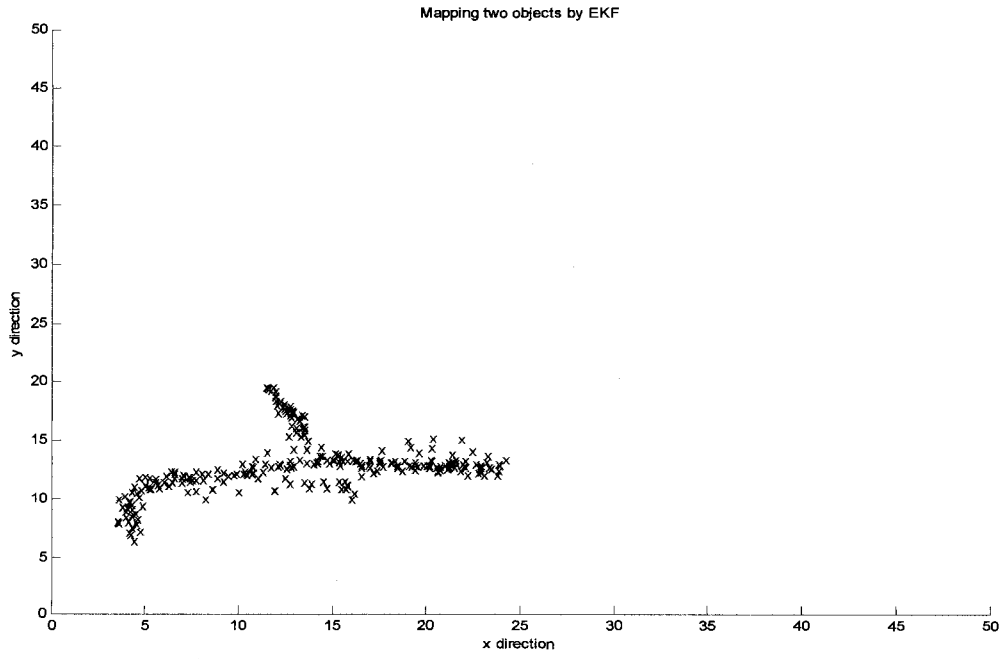


Figure 6.33: Mapping two objects by EKF (220 detectable landmarks)

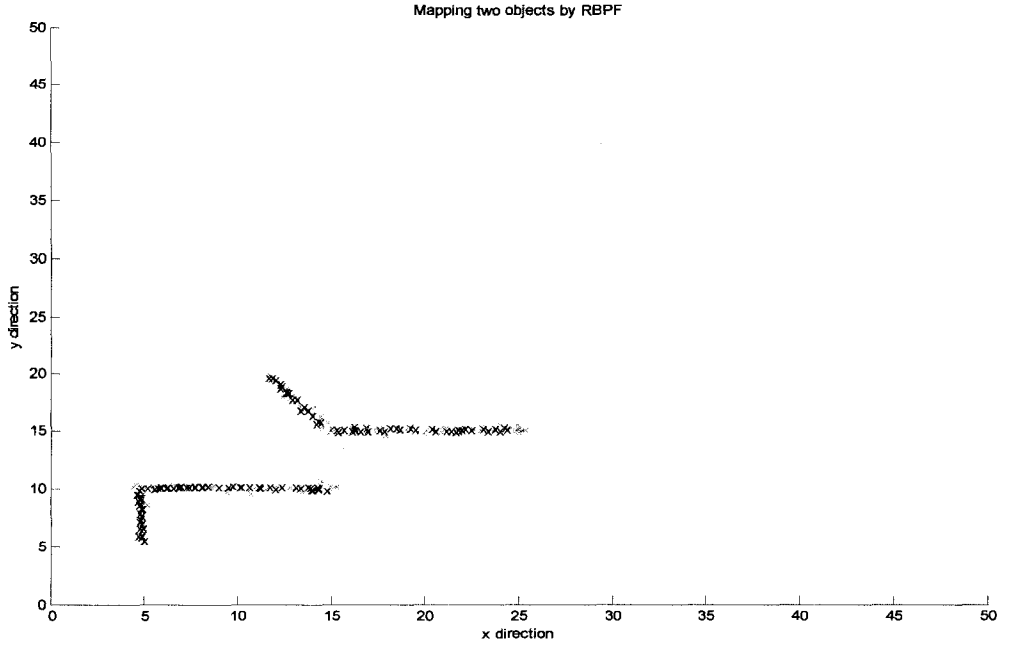


Figure 6.34: Mapping two objects by FAST-SLAM (220 detectable landmarks)

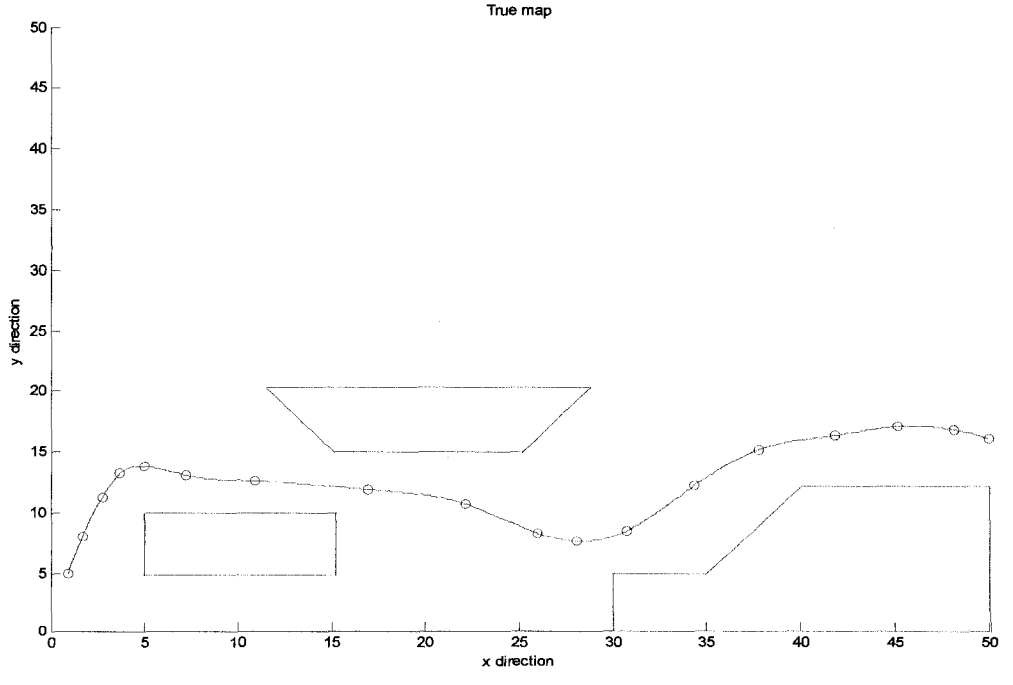


Figure 6.35: True map of three objects

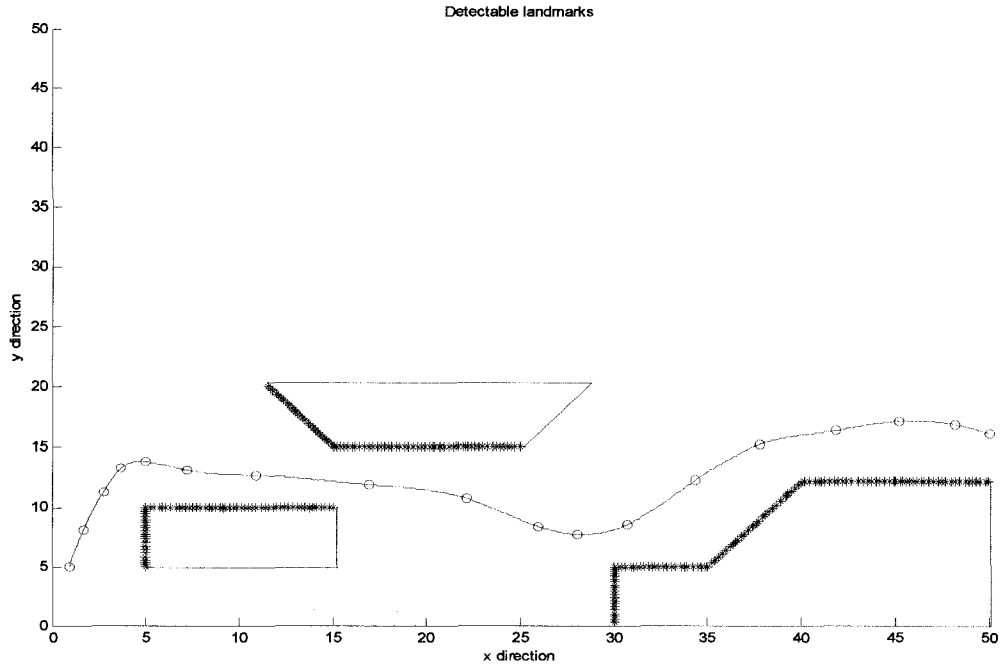


Figure 6.36: Detectable landmarks of three objects (390 detectable landmarks)

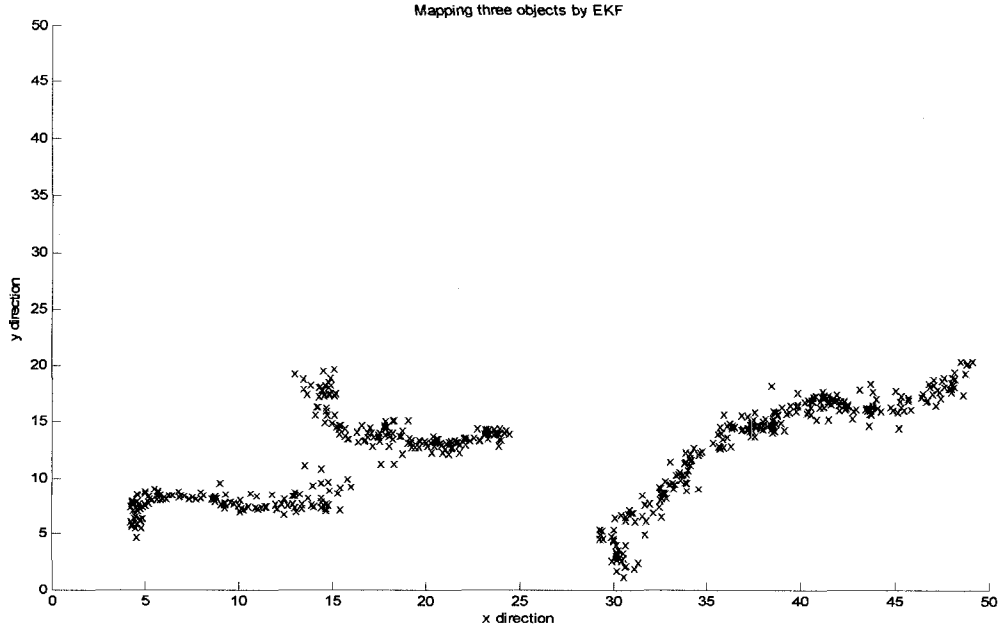


Figure 6.37: Mapping three objects by EKF (390 detectable landmarks)

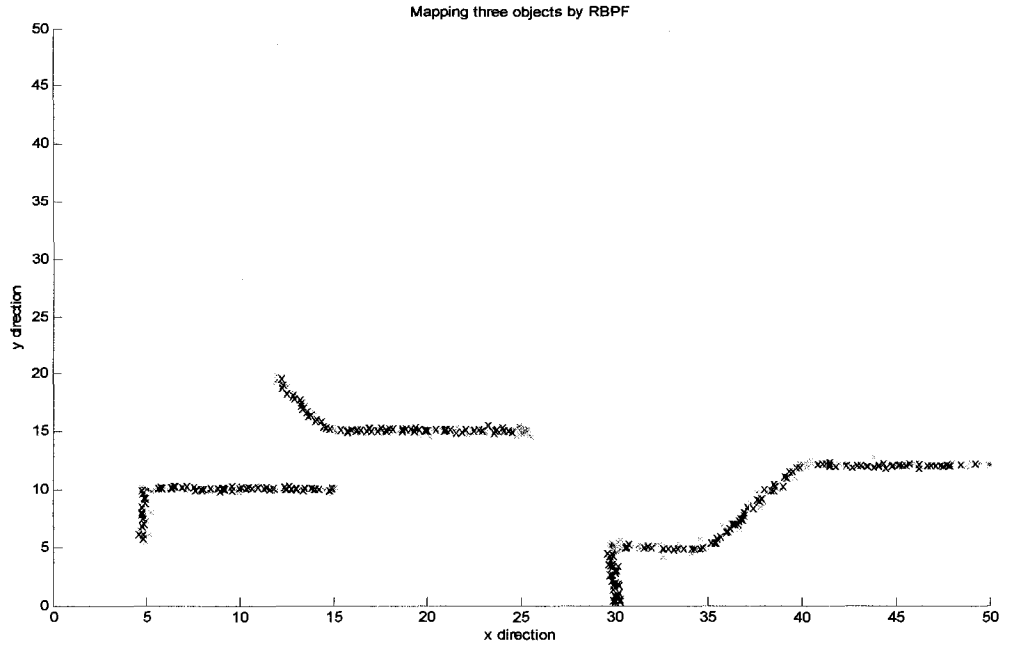


Figure 6.38: Mapping three objects by FAST-SLAM (390 detectable landmarks)

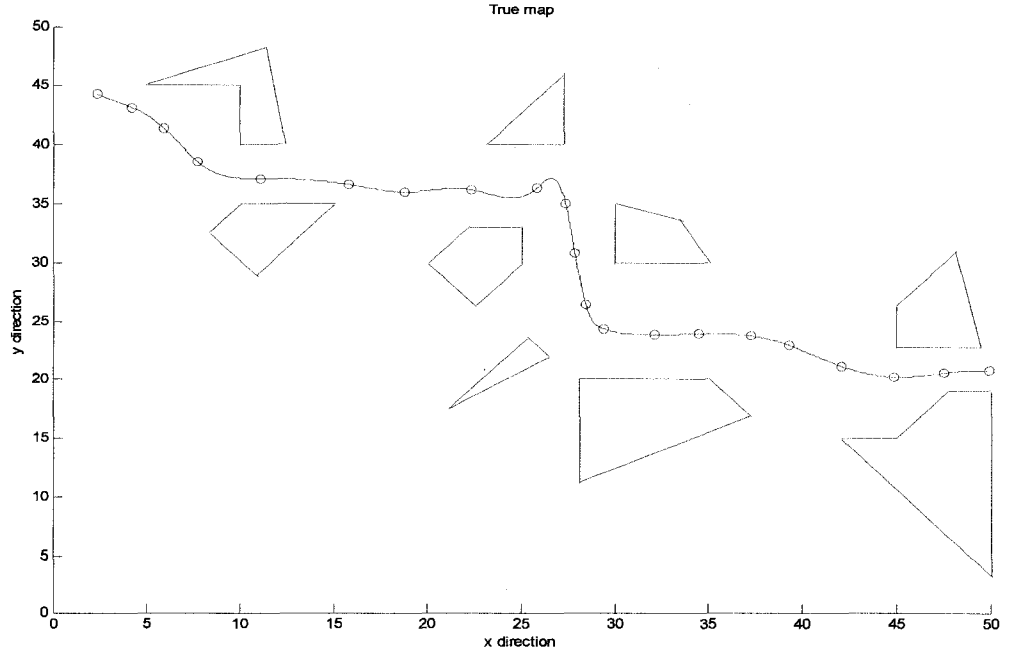


Figure 6.39: True map of multiple objects

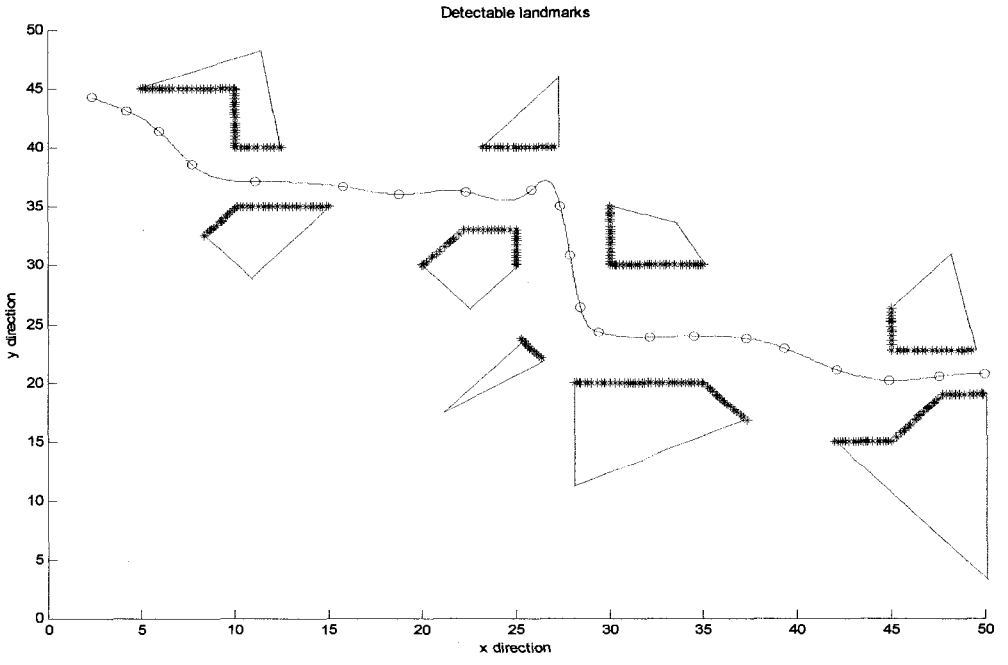


Figure 6.40: Detectable landmarks of three objects (750 detectable landmarks)

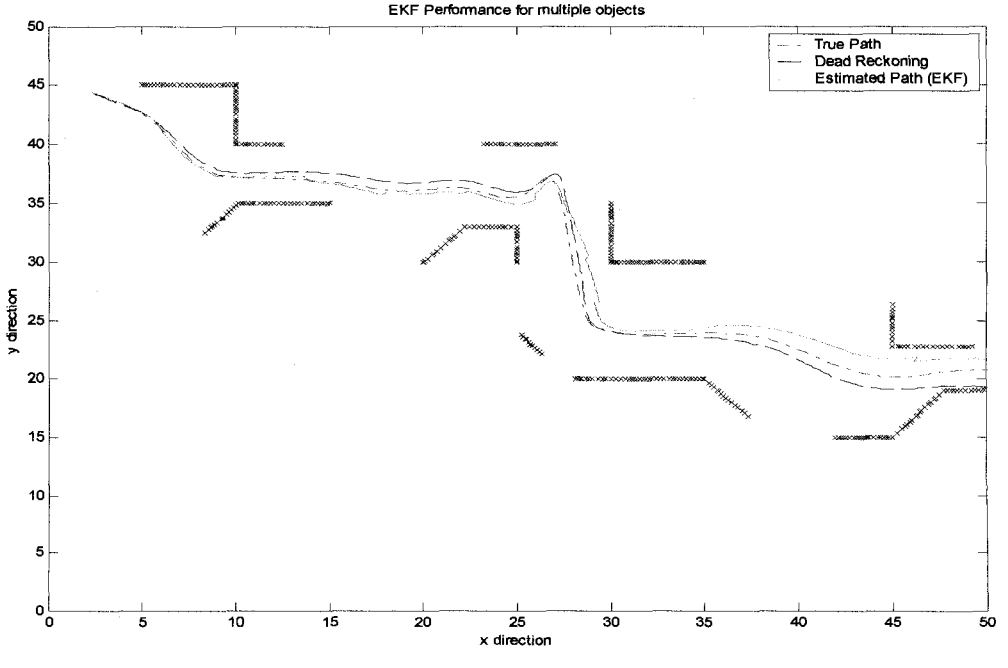


Figure 6.41: EKF-SLAM performance for multiple objects (750 detected landmarks)

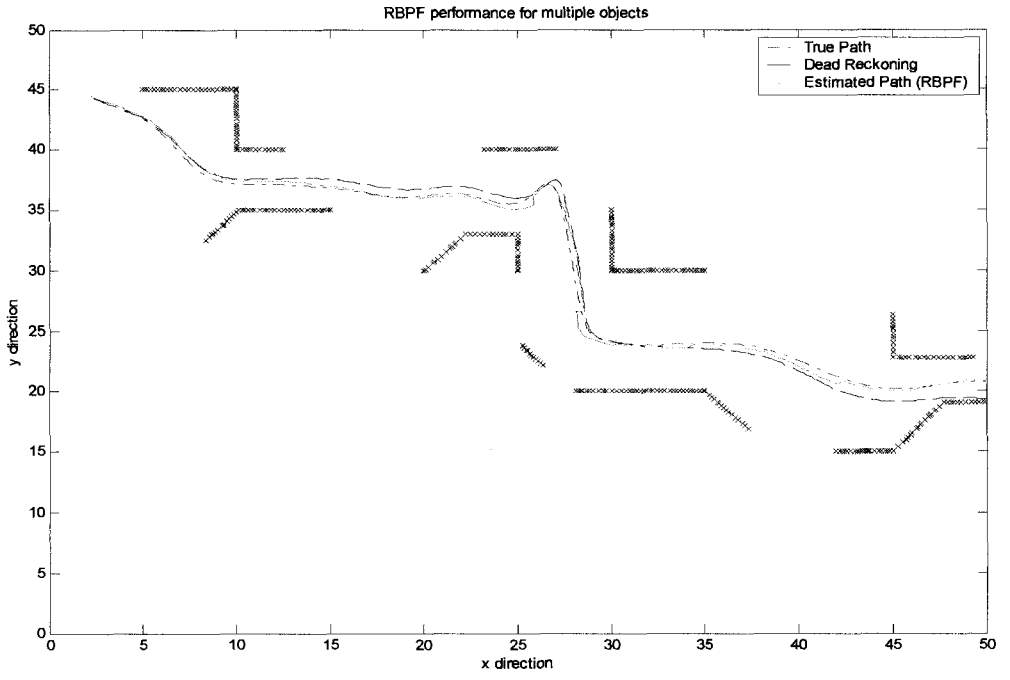


Figure 6.42: FAST-SLAM performance for multiple objects (750 detected landmarks)

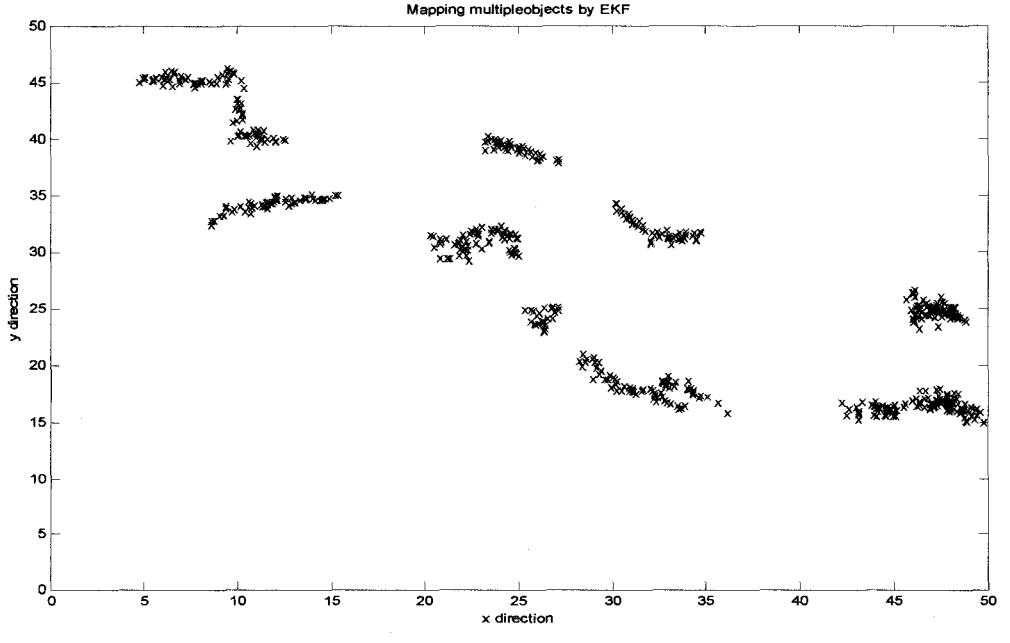


Figure 6.43: Mapping multiple objects by EKF-SLAM (750 detectable landmarks)

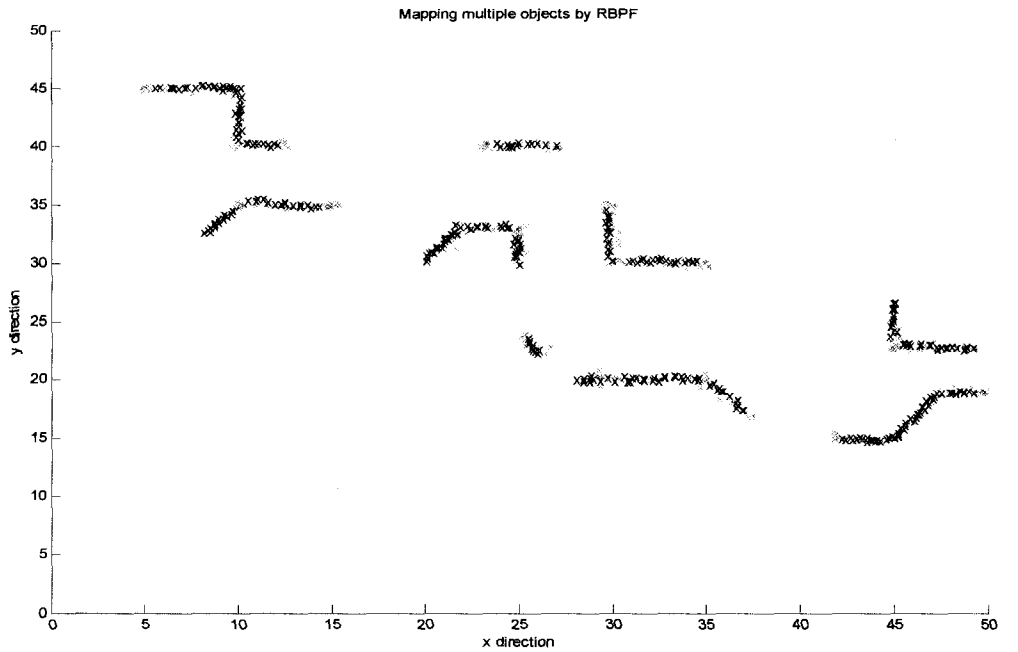


Figure 6.44: Mapping multiple objects by FAST-SLAM (750 detectable landmarks)

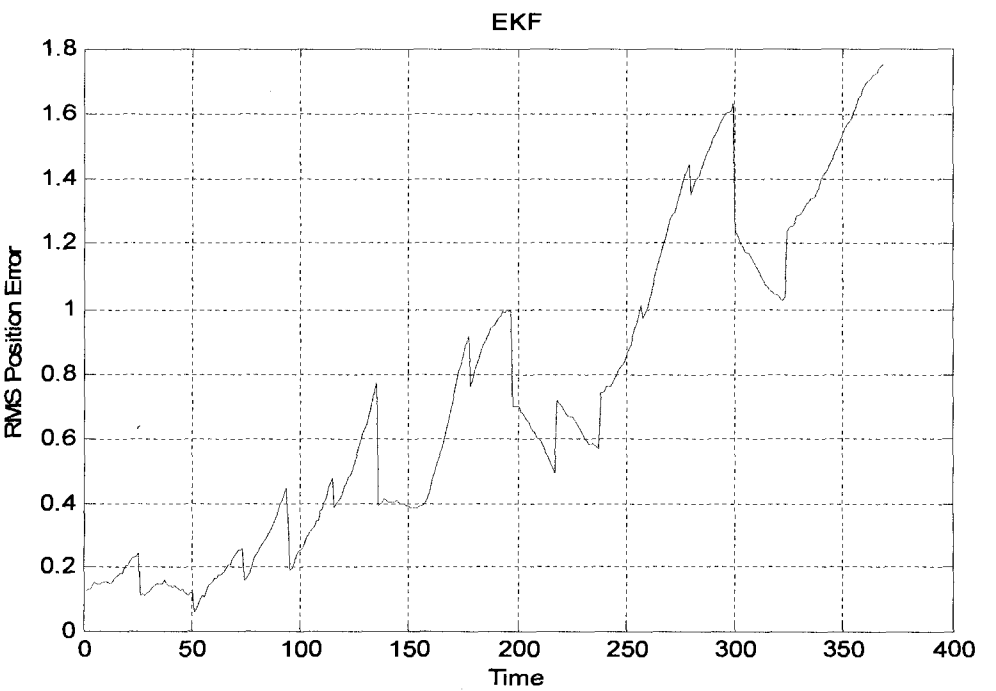


Figure 6.45: EKF-SLAM RMS error for multiple objects (750 detected landmarks)

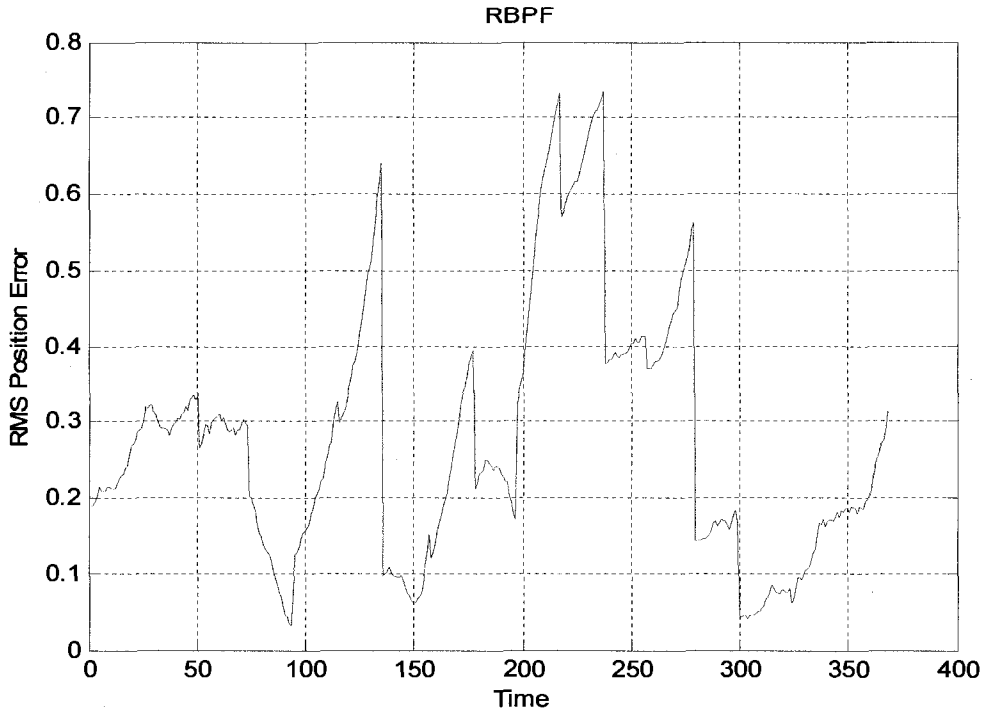


Figure 6.46: FAST-SLAM RMS error for multiple objects (750 detected landmarks)

6.3 Observation noise

Through figures 6.47 to 6.61 same path was investigated for different observation noise in FAST-SLAM. Number of particles in the following simulations is assumed as 300. When the motion system is very noisy (assumming 0.2), but the measurement noise approaches zero, the filter starts diverging. This is an important disadvantage of a standard particle filter. If the sensor is very accurate which means its error is close to zero, and at the same time the robot's motion is very noisy, the observation noise and the robot noise will not be matched properly.

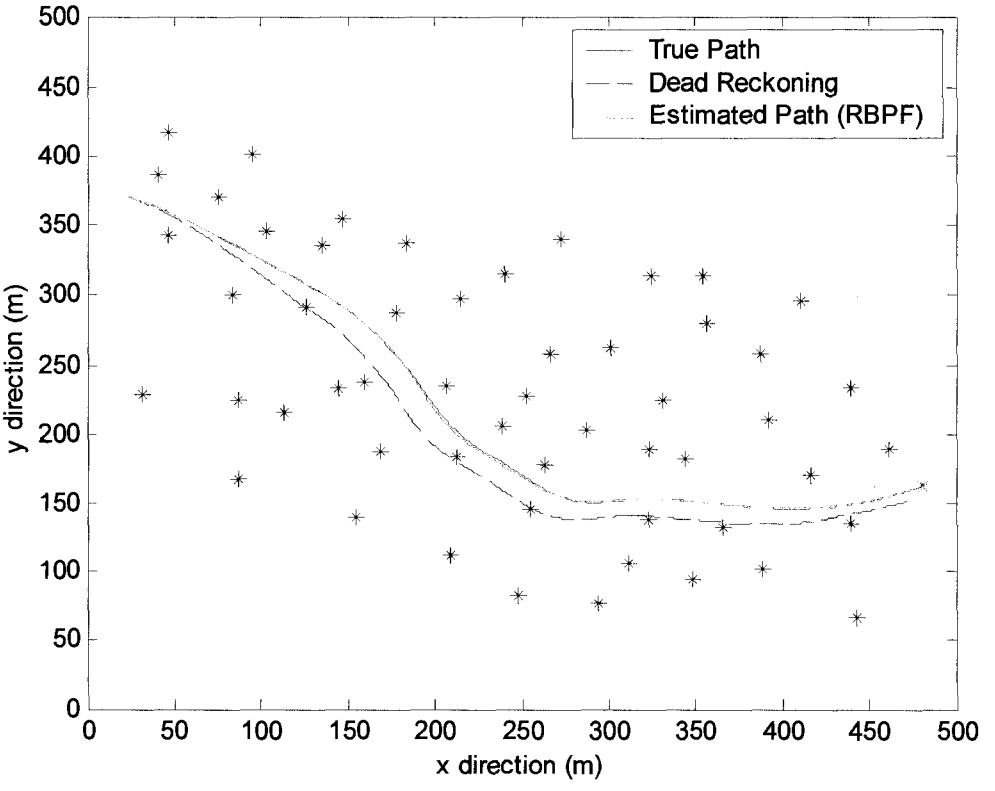


Figure 6.47: Simulated path of the robot with 300 particles. The velocity noise is 0.20. The observation noise is 0.20

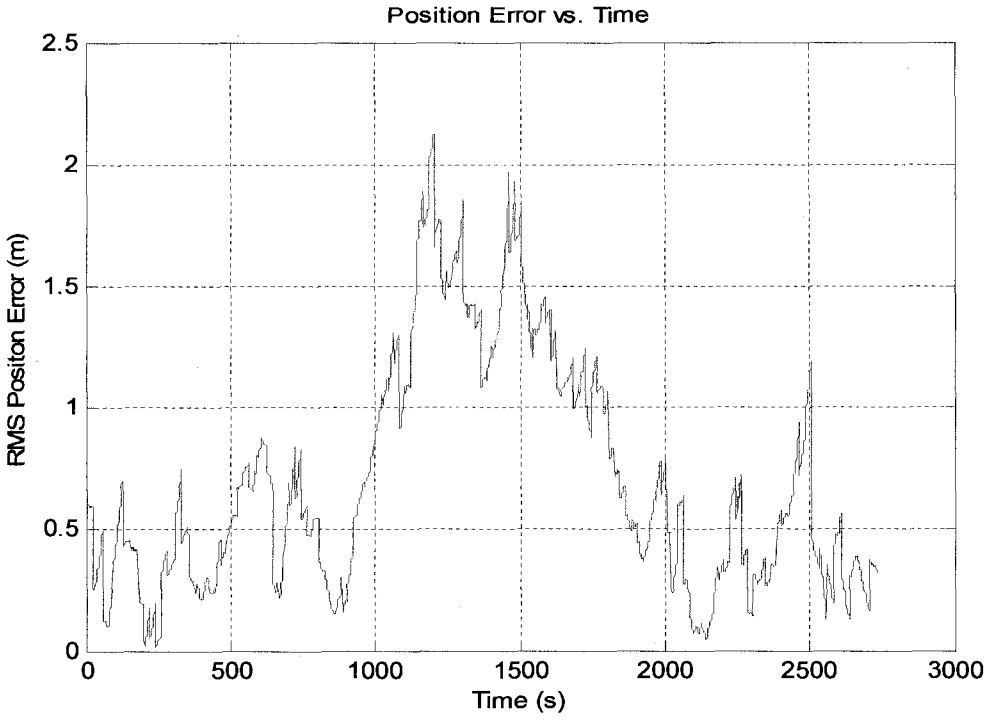


Figure 6.48: Root Mean Square of the position error with 300 particles. The velocity noise is 0.20. The observation noise is 0.20

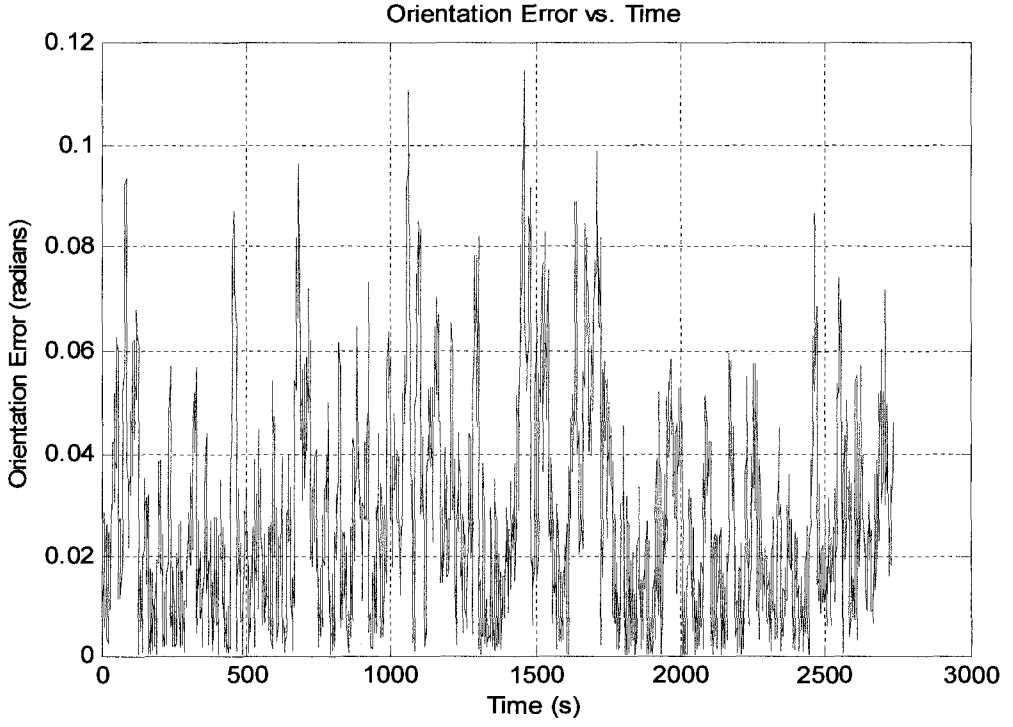


Figure 6.49: Orientation error with 300 particles. The velocity noise is 0.20. The observation noise is 0.20.

The reason for this is that many particles will be thrown in the re-sampling step and there will be not enough particles left to be incorporated for the rest of the path estimation. Consequently, the filter diverges and the result will be catastrophic. This property that affects the performance of FAST-SLAM is reffered as sample impoverishment. Here it should be mentioned that this amount of noise level near to zero (between 0 and 0.01) is usually 1% of average range value. In the mentioned simulations the robot linear velocity noise has been 0.2, while the observation noise changes from 0 to 0.20. As the observation noise decreases towards zero, the FAST-SLAM confronts unexpected results in terms of the robot path estimation and it is ended up with highly catastrophic results.

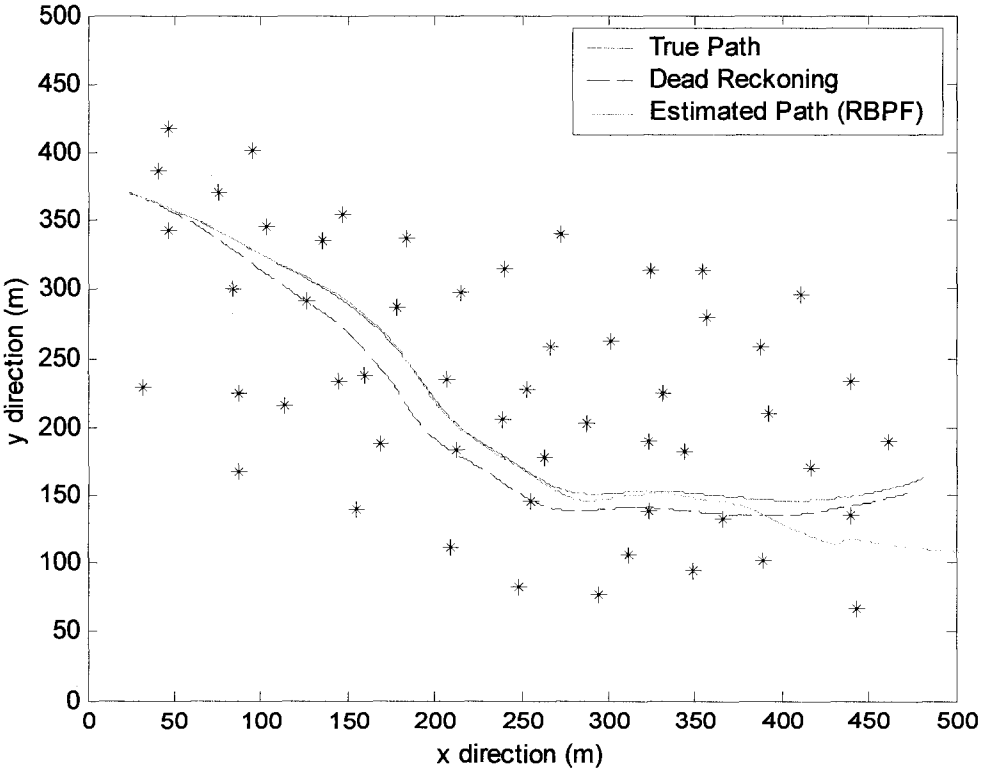


Figure 6.50: Simulated path of the robot with 300 particles. The velocity noise is 0.20. The observation noise is 0.10

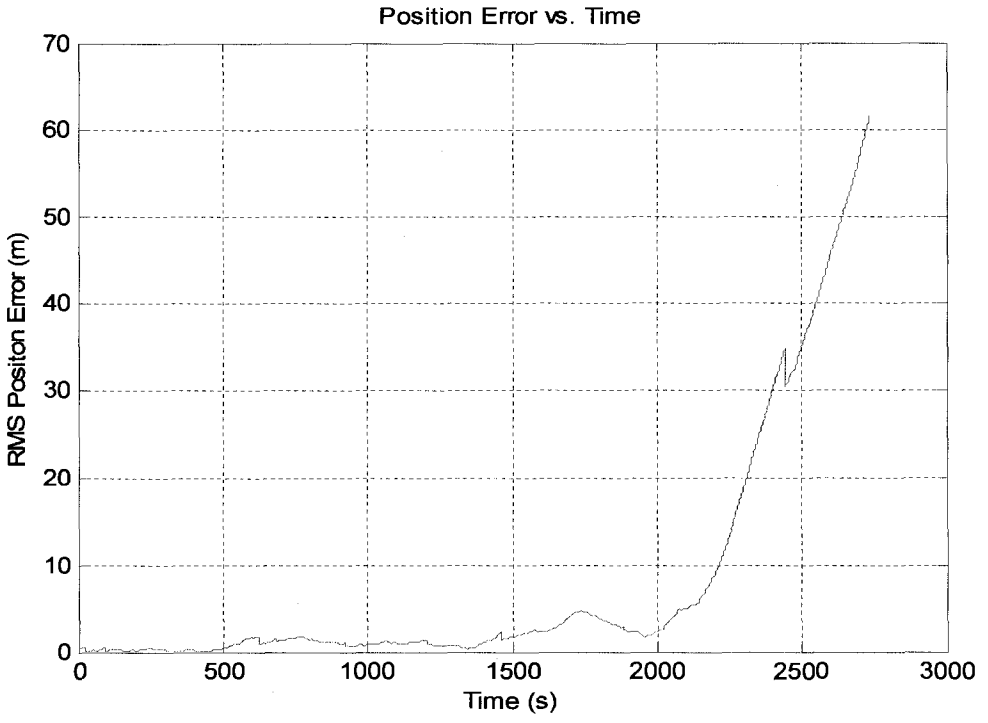


Figure 6.51: Root Mean Square of the position error with 300 particles. The velocity noise is 020. The observation noise is 0.10

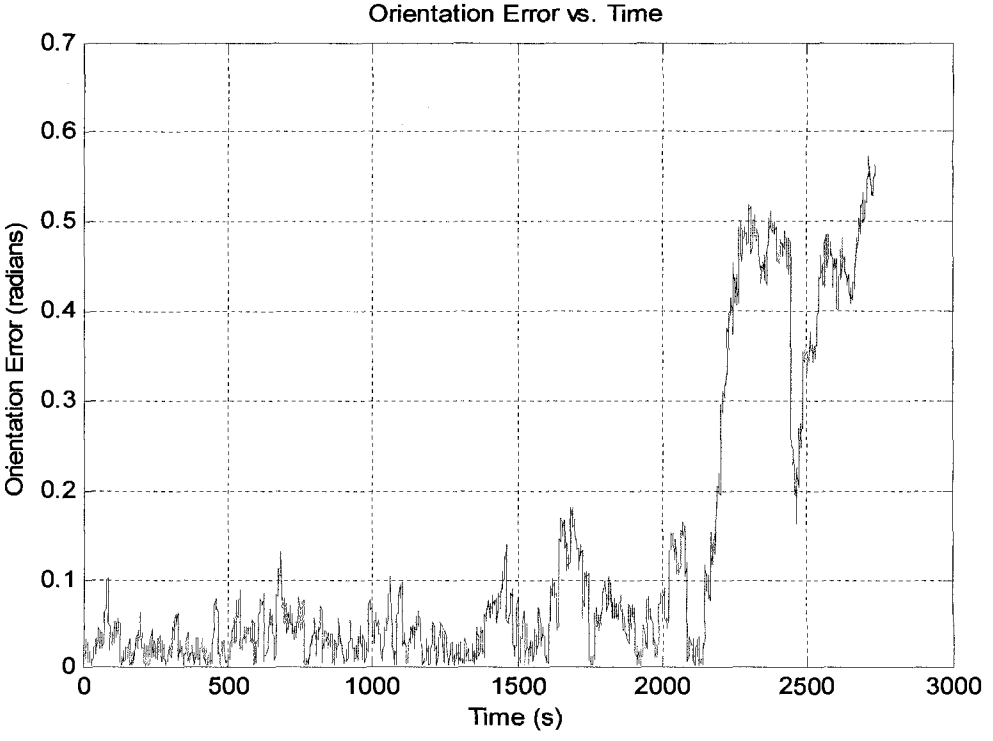


Figure 6.52: Orientation error with 300 particles. The velocity noise is 0.20. The observation noise is 0.10

The RMS position error and orientation error has been plotted for the comparison on each different level of observation noise. As the level of observation noise becomes lower, the RMS position error increases as well as the orientation error. It is important to mention that this situation happens only when the proposal and posterior distribution are mismatched and this can be the result of a big difference between noises of robot's motion and the sensor. The motion model spreads the particles out over a large space and only a small fraction of particles receive non-negligible weights. Consequently, there will be not many particles left to estimate the path at the next time step. Furthermore, as more landmarks are observed, many more particles are thrown out and the problem is more compounded.

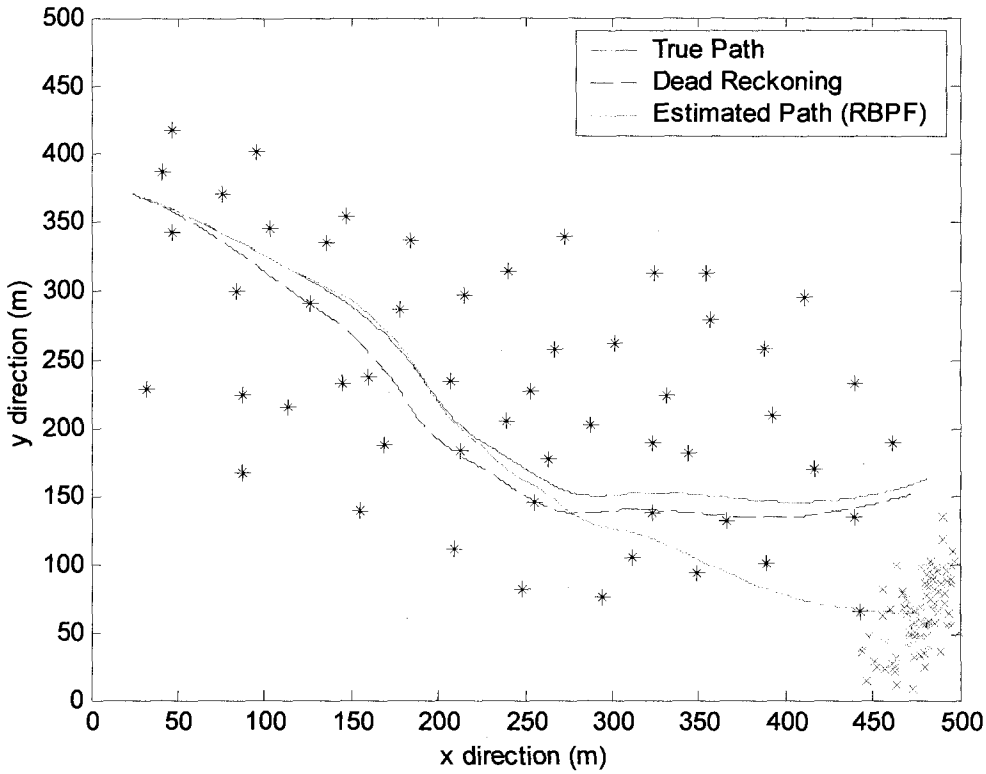


Figure 6.53: Simulated path of the robot with 300 particles. The velocity noise is 0.20. The observation noise is 0.075

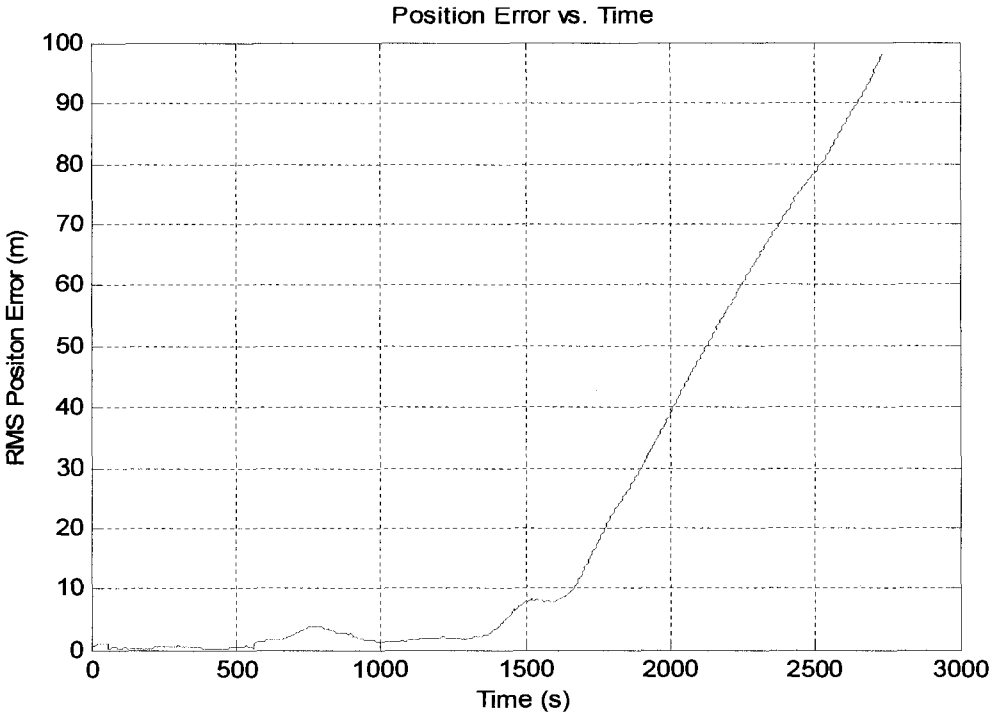


Figure 6.54: Root Mean Square of the position error with 300 particles. The velocity noise is 0.20. The observation noise is 0.075

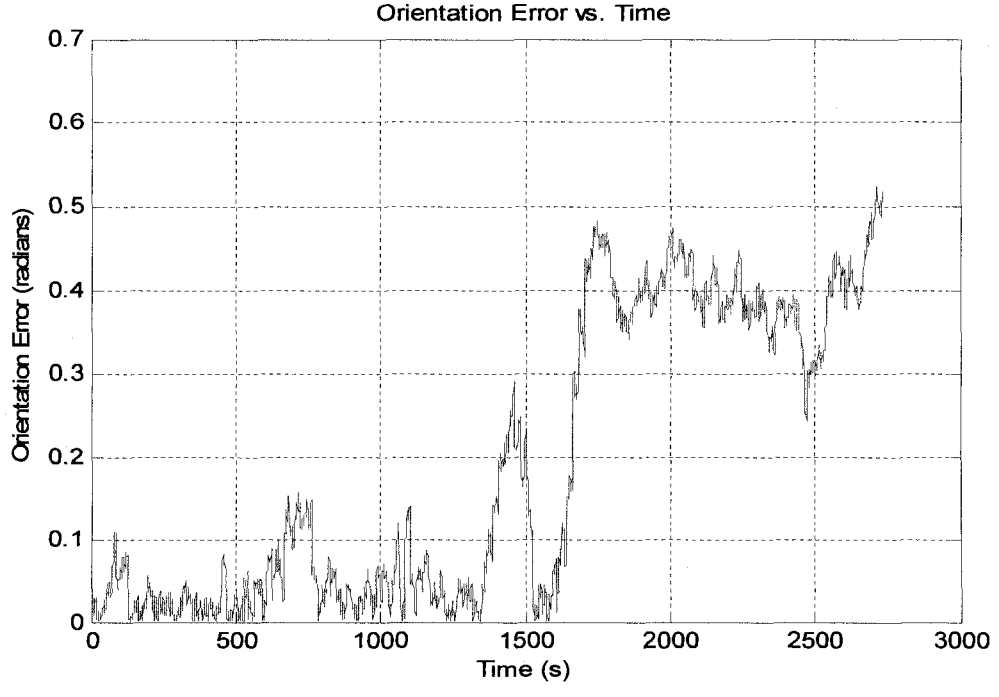


Figure 6.55: Orientation error with 300 particles. The velocity noise is 0.20. The observation noise is 0.075

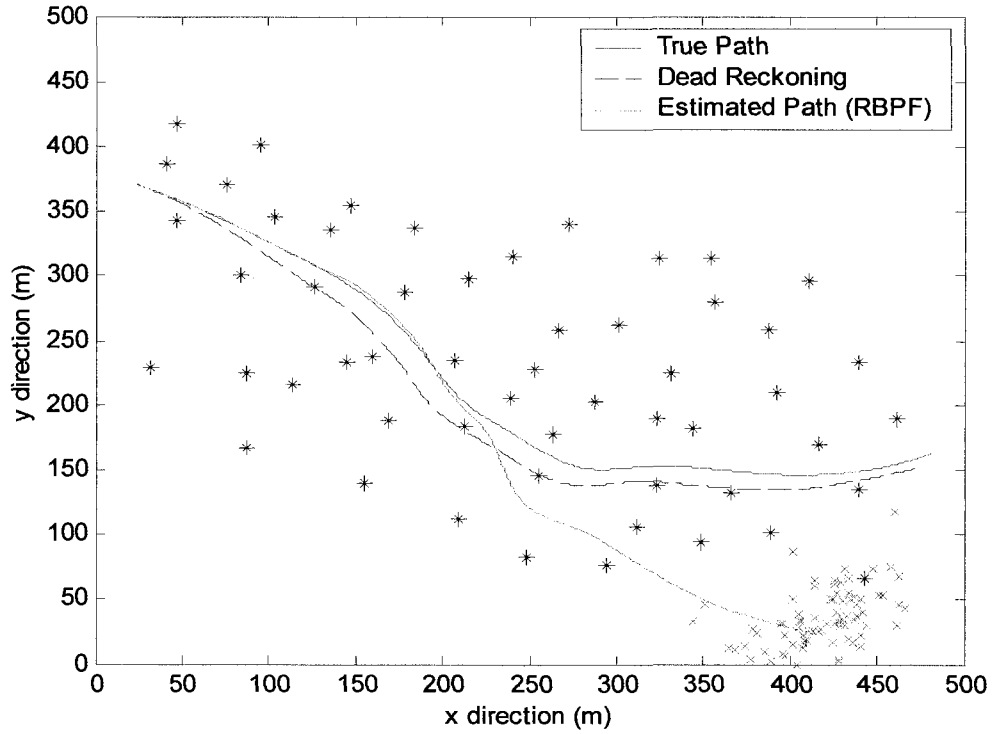


Figure 6.56: Simulated path of the robot with 300 particles. The velocity noise is 0.20. The observation noise is 0.025

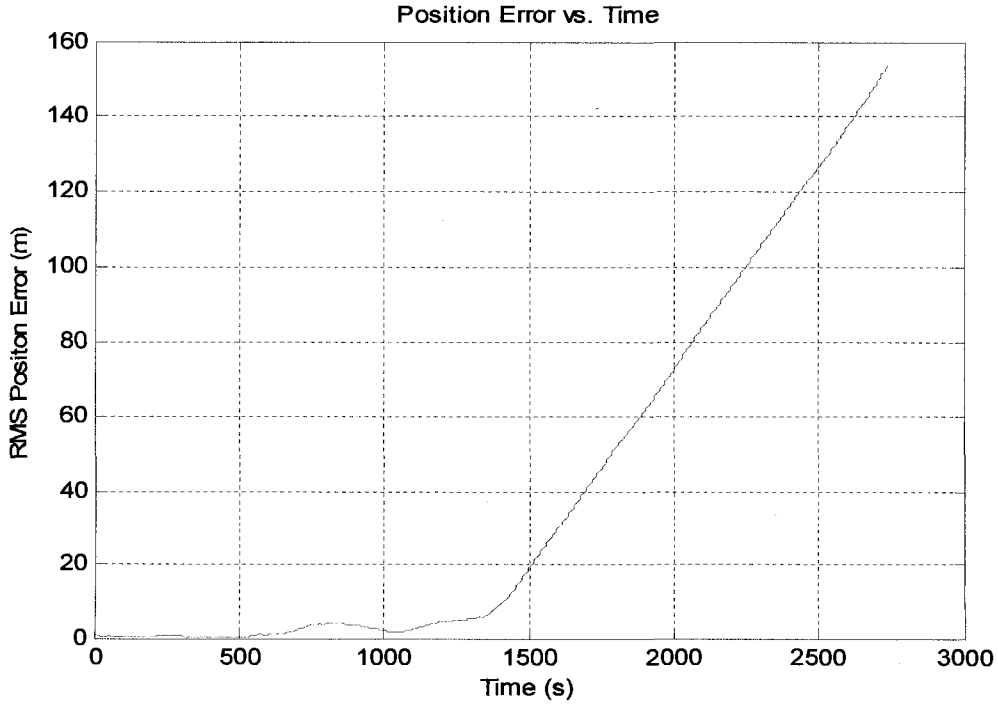


Figure 6.57: Root Mean Square of the position error with 300 particles. The velocity noise is 0.20. The observation noise is 0.025

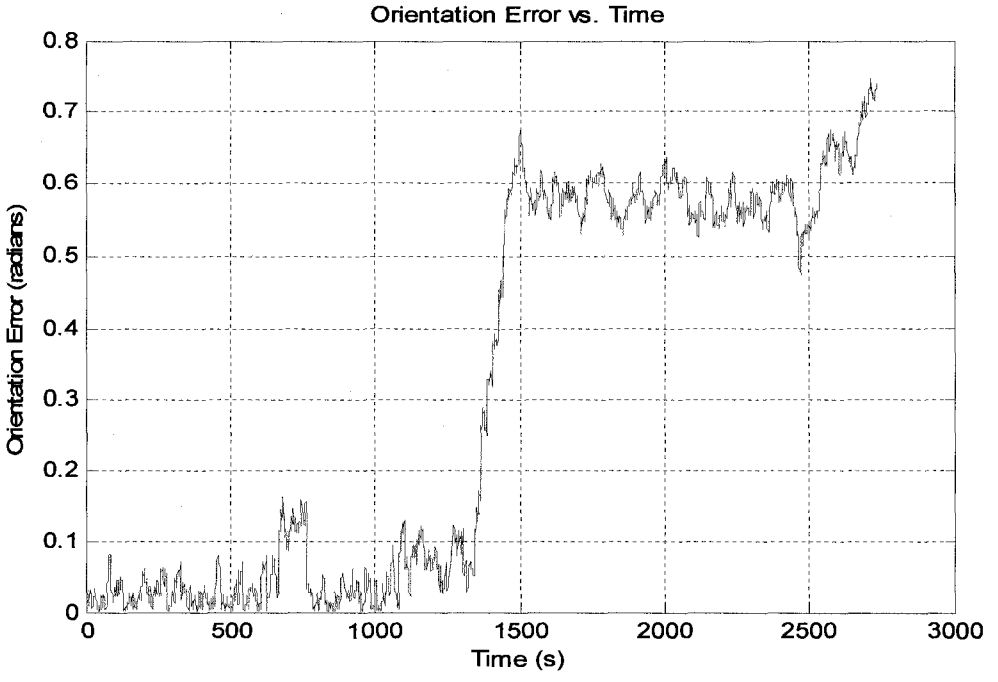


Figure 6.58: Orientation error with 300 particles. The velocity noise is 0.20. The observation noise is 0.025

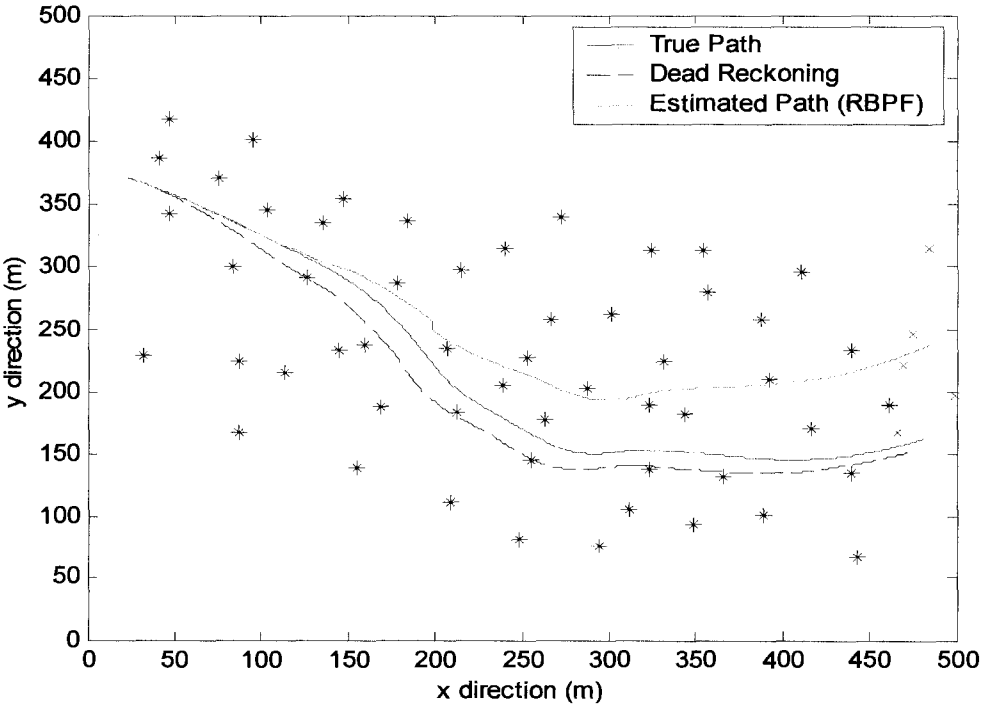


Figure 6.59: Simulated path of the robot with 300 particles. The velocity noise is 0.20. The observation noise is zero

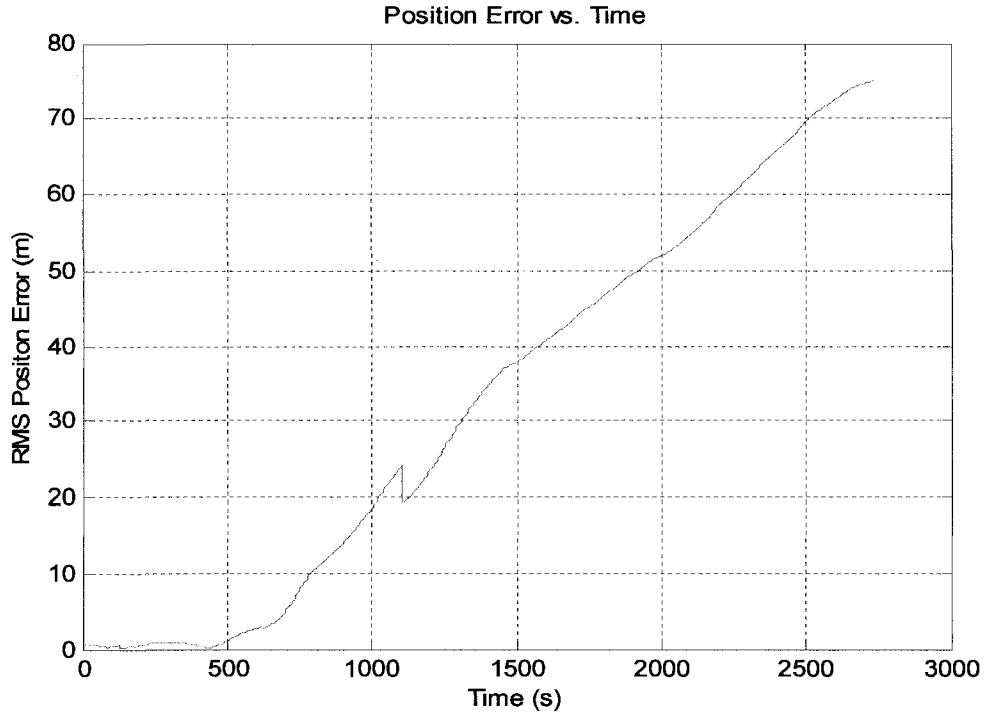


Figure 6.60: Root Mean Square of the position error with 300 particles. The velocity noise is 0.20. The observation noise is zero

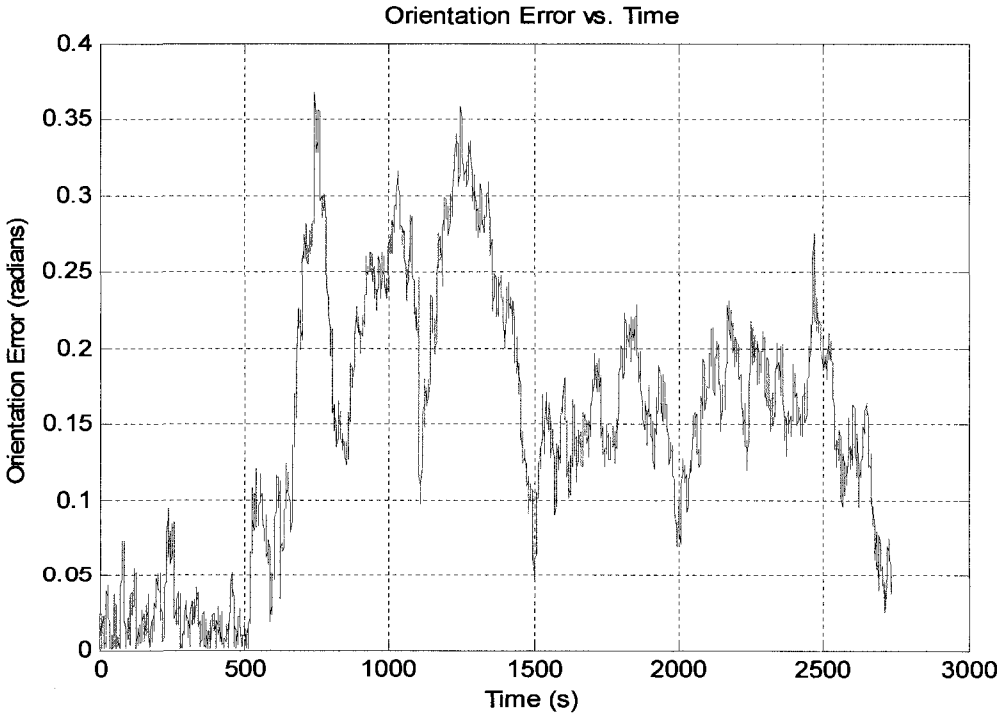


Figure 6.61: Orientation error with 300 particles. The velocity noise is 0.20. The observation noise is zero

Through figures 6.62 to 6.69 same path was simulated but this time observation noise was held constant at 0.025. Instead, the velocity noise differs from zero to 1.50. Results show that as long as the difference between velocity noise and observation noise is held small, the performance of the filter is good. When the motion becomes noiser, the filter diverges and robot can not stay near the desired path. This is again due to sample impoverishment. Results show that many particles will be thrown in the re-sampling step and there will not be enough particles left to be incorporated for the rest of the path estimation. The RMS position error has been plotted for the comparison on each different level of velocity noise.

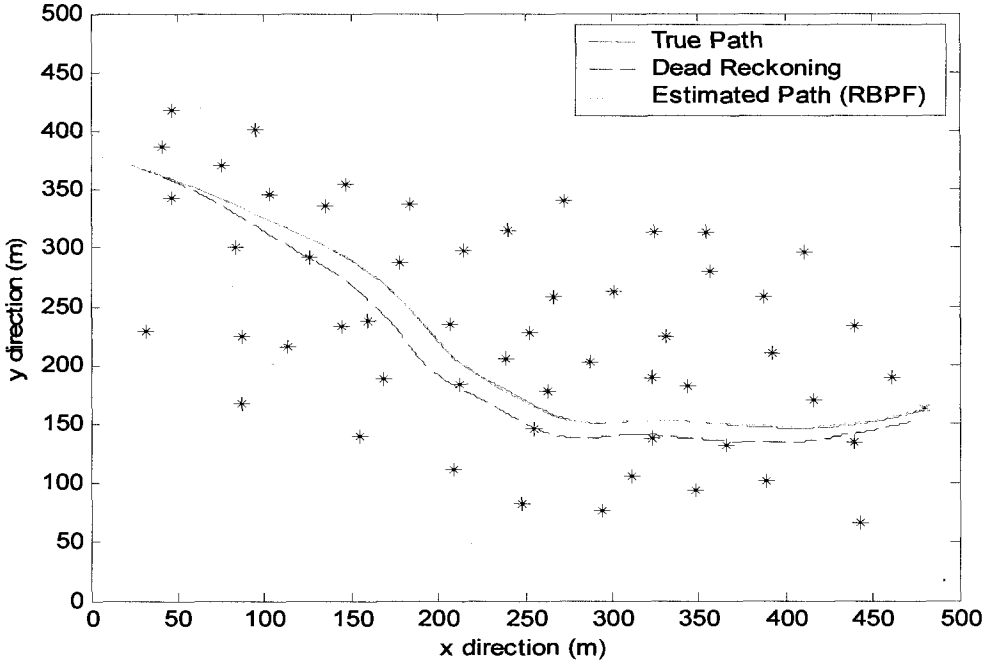


Figure 6.62: Simulated path of the robot with 300 particles. The velocity noise is zero. The observation noise is 0.10

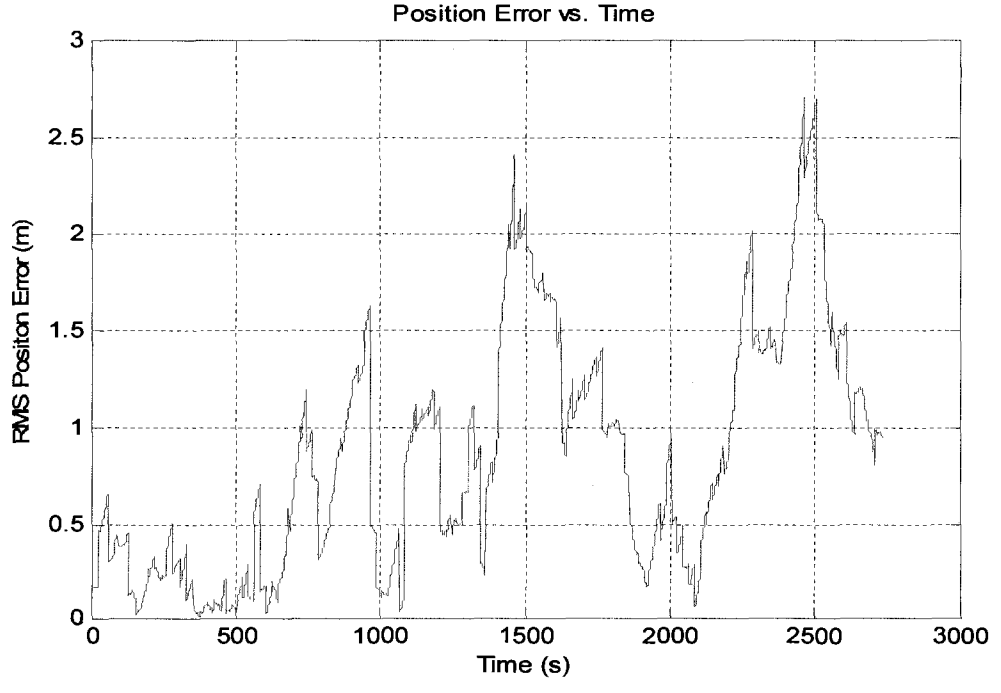


Figure 6.63: Root Mean Square of the position error with 300 particles. The velocity noise is zero. The observation noise is 0.10.

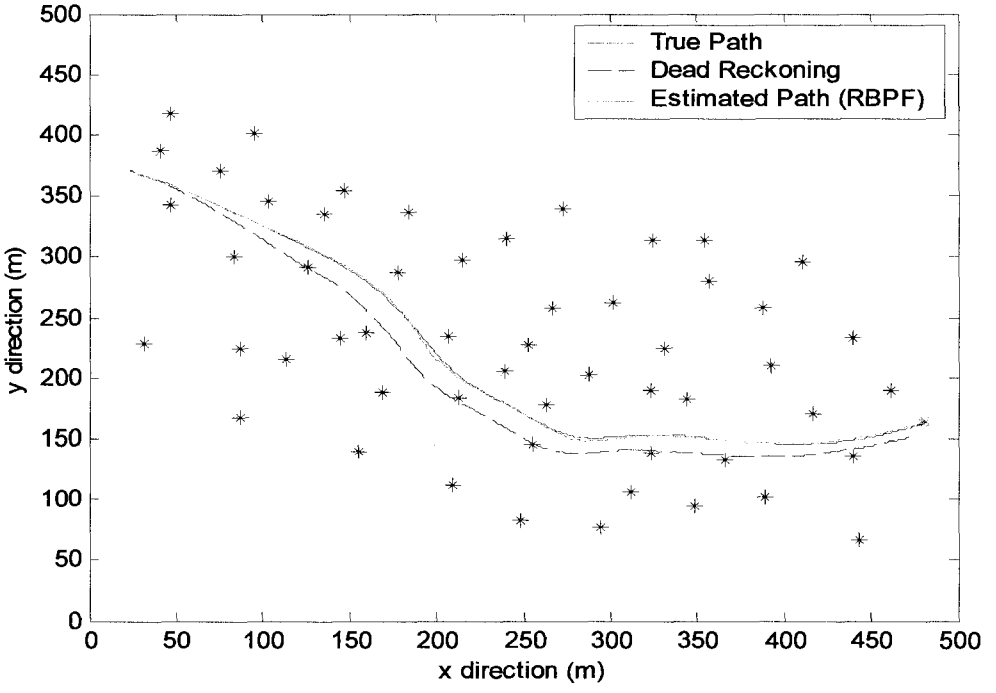


Figure 6.64: Simulated path of the robot with 300 particles. The velocity noise is 0.10. The observation noise is 0.10

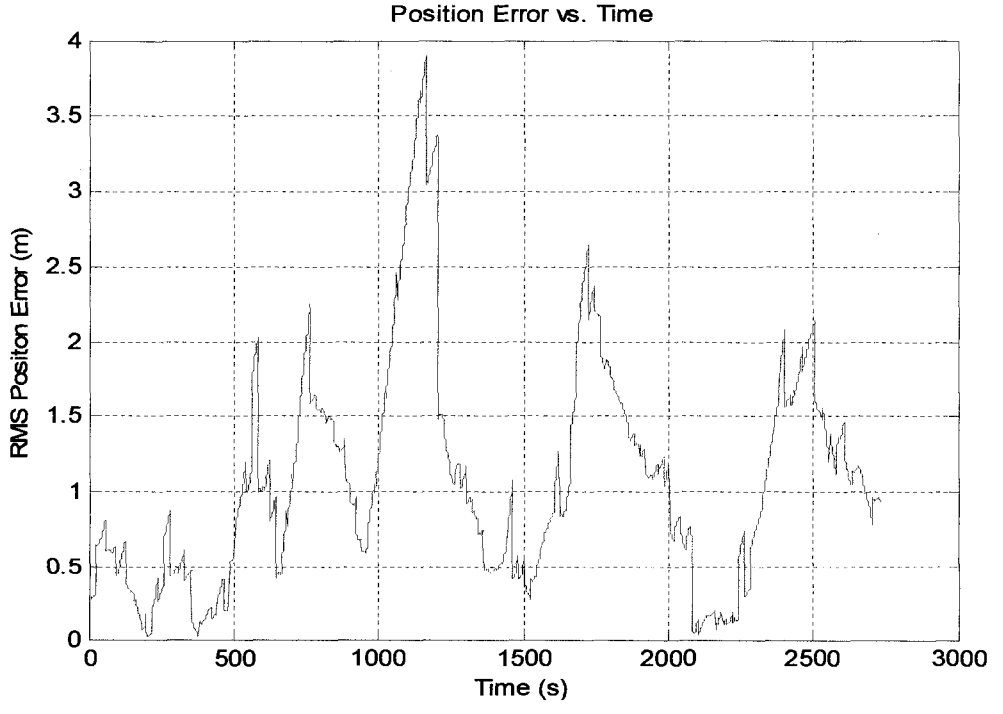


Figure 6.65: Root Mean Square of the position error with 300 particles. The velocity noise is 0.10. The observation noise is 0.10.

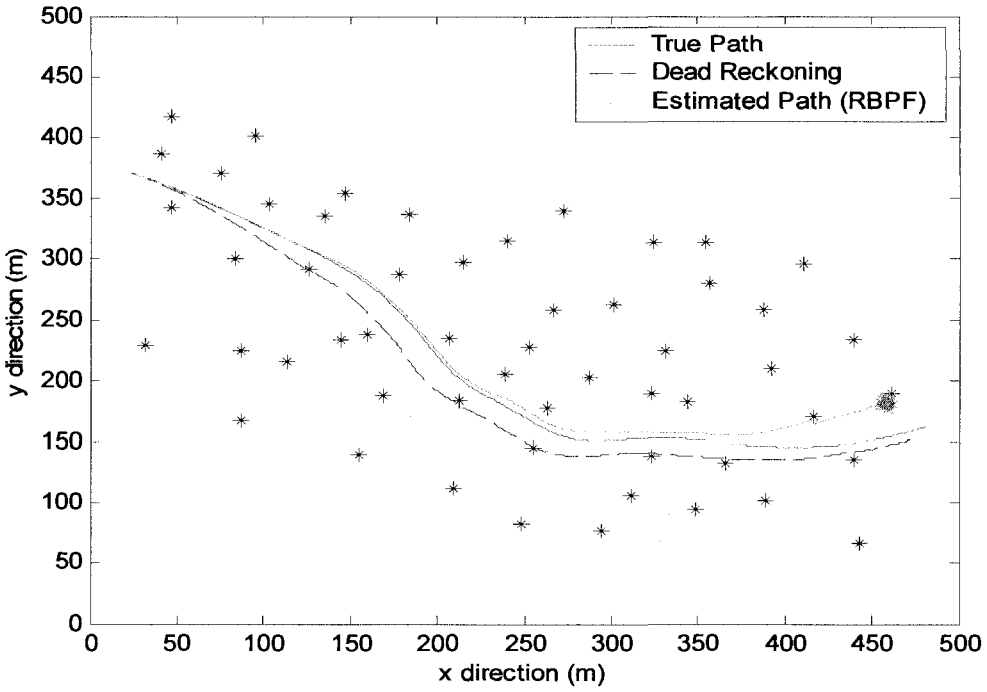


Figure 6.66: Simulated path of the robot with 300 particles. The velocity noise is 0.20. The observation noise is 0.10

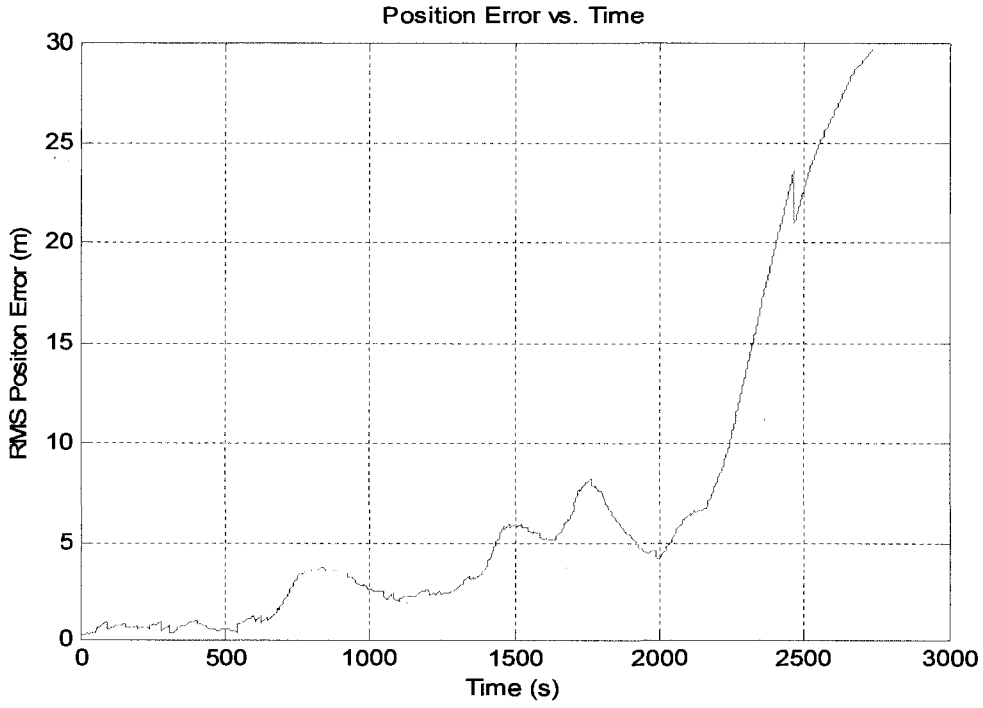


Figure 6.67: Root Mean Square of the position error with 300 particles. The velocity noise is 0.20. The observation noise is 0.10.

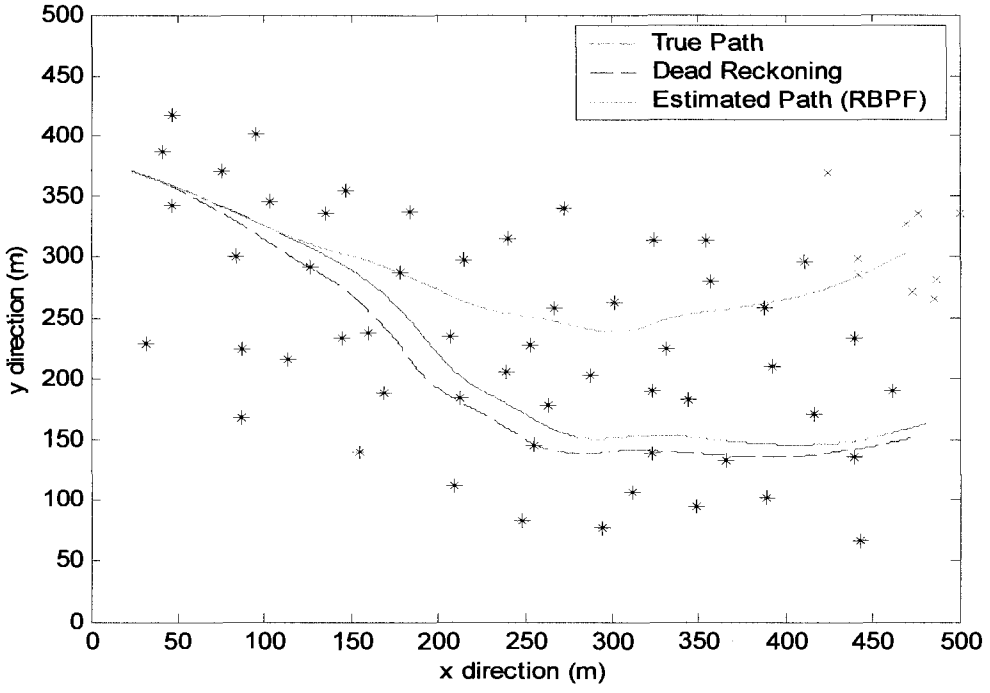


Figure 6.68: Simulated path of the robot with 300 particles. The velocity noise is 0.30. The observation noise is 0.10.

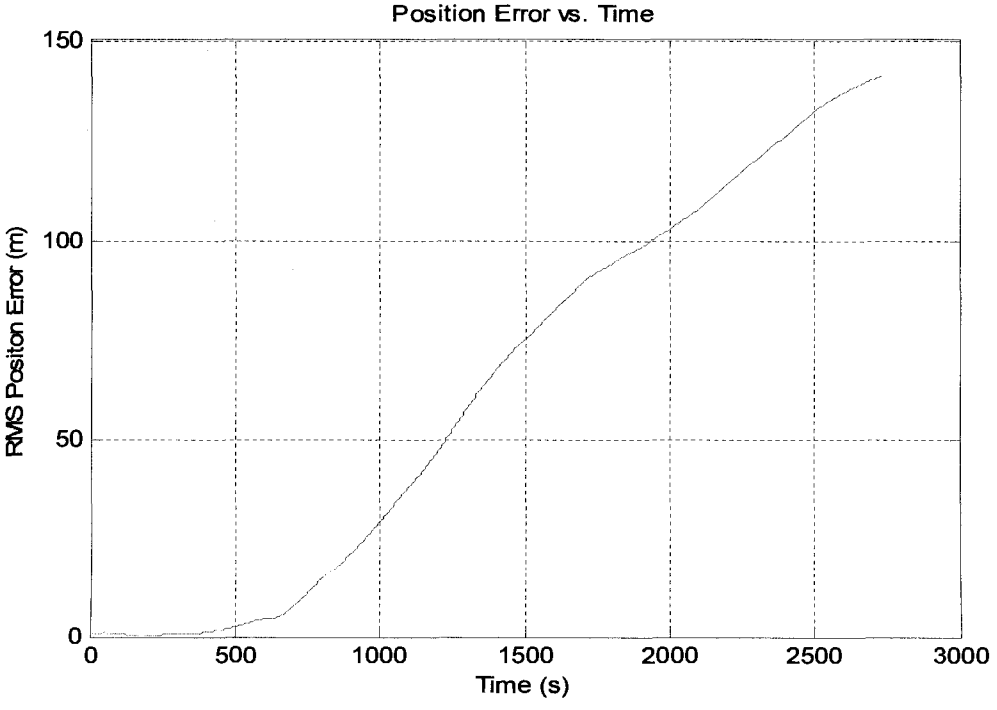


Figure 6.69: Root Mean Square of the position error with 300 particles. The velocity noise is 0.30. The observation noise 0.10.

Figure 6.70 shows how a substantially noisy motion model with the observation noise equal to 0.001 in which the particles are drawn. The subsequent observation makes the robot's true position to lie within the overlaid probability, while the samples inside the ellipse will be duplicated multiple times by the resampling process. As the observations become more accurate, fewer unique samples will remain for the rest of process. Finally, sufficiently accurate observations will cause the particle filter to diverge. This is a well known failure mode of a standard PF. Another version of FAST-SALM called FAST-SLAM 2.0 has been developed by Montemerlo [11] that addresses this defect and by a more detailed re-sampling representation prevents the failure in this particular situation.

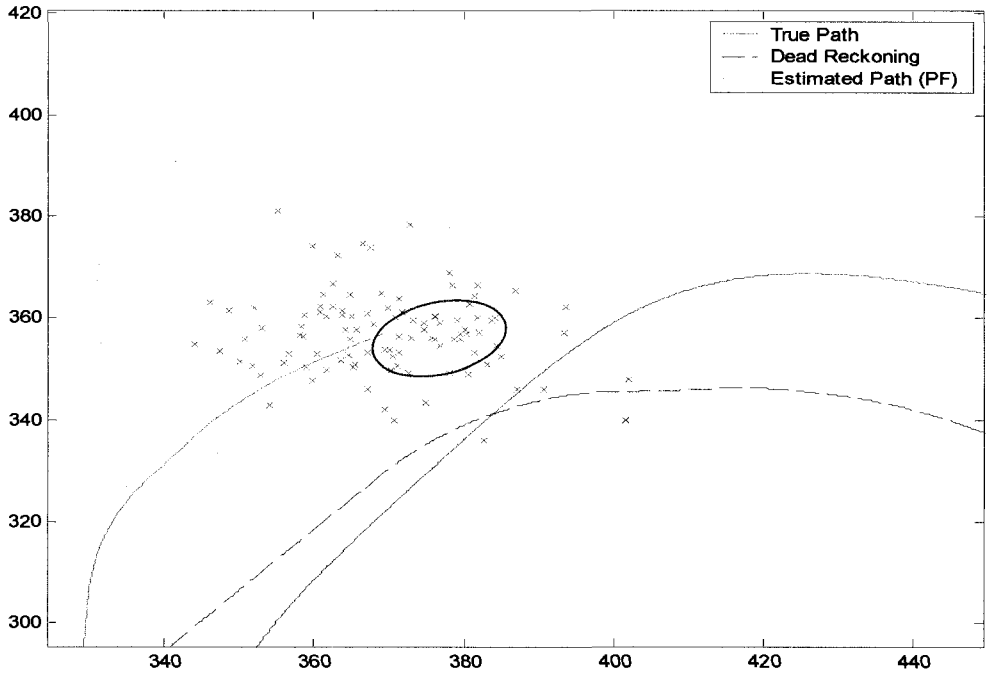


Figure 6.70: Mismatch between proposal and target distributions. Number of particles is 100. Angular velocity noise is zero and linear velocity noise is 0.02.

Figures 6.71, demonstrates the root mean square position error for different levels of measurement noise. As it is shown in the diagram, for noise levels less than 0.1 the root mean square position error increases without bound. The number of particles for this simulation has been considered 1000. This diagram proves that when the sensor becomes very much accurate, particle filter diverges and error increases without bound. Increasing number of particles may decrease the RMS position error but the result will still be inaccurate enough to make the standard particle filter an inappropriate algorithm for noise levels less than 0.1. However a compromise between the motion noise and the observation noise can make the filter to work appropriately with no filter diverging.

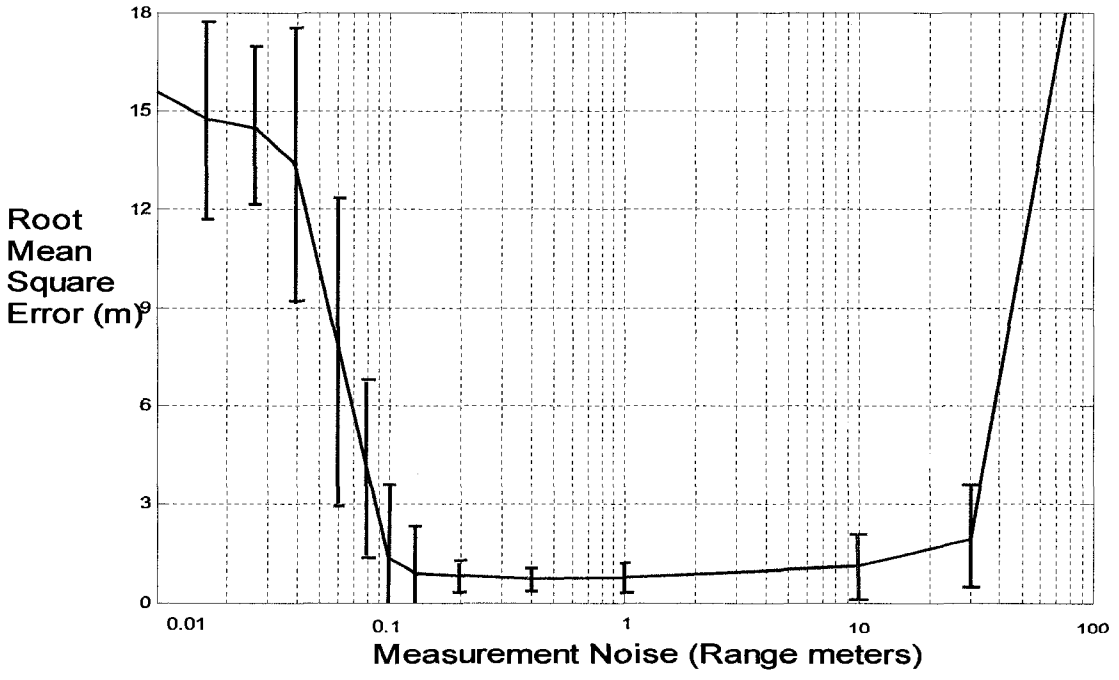


Figure 6.71: Demonstrates different observation noise levels versus RMS position error for 1000 particles.

6.4 Number of Particles

Figures 6.72 to 6.83 demonstrate navigation of autonomous robot with different numbers of particles for a similar path. All the simulated data has been done using computer specification of Pentium 4 with 1.8 GHz speed, and 2.0 GB of RAM. Each simulation with a specific number of particles was run 10 times. The observation noise is 0.25, angular velocity noise is 0.004, linear velocity noise is 0.15, and noise levels along x and y directions are both 0.2. All simulations are done based on logarithmic FAST-SLAM. Reults show that while number of particles for path estimation changes from 300 to 110000, the average of RMS position error changes only 0.15m. This concludes that unless the path estimation must be as accurate as possible, increasing number of particles should be avoided. The reason for not increasing number of particles is that in the resampling process, there will be unlimited number of particles that have the same information about path with not really making that much accuracy difference in the path estimation. This leads to more calculation and consequently more memory and time consuming.

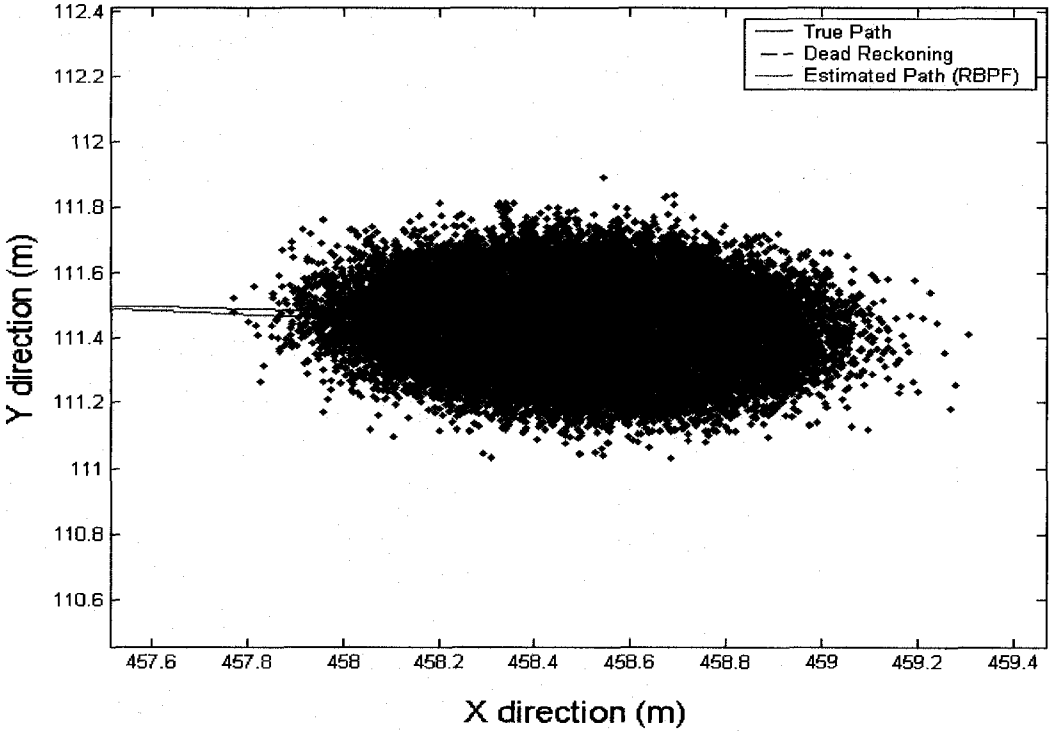


Figure 6.72: Simulation of the robot path with 110000 particles.

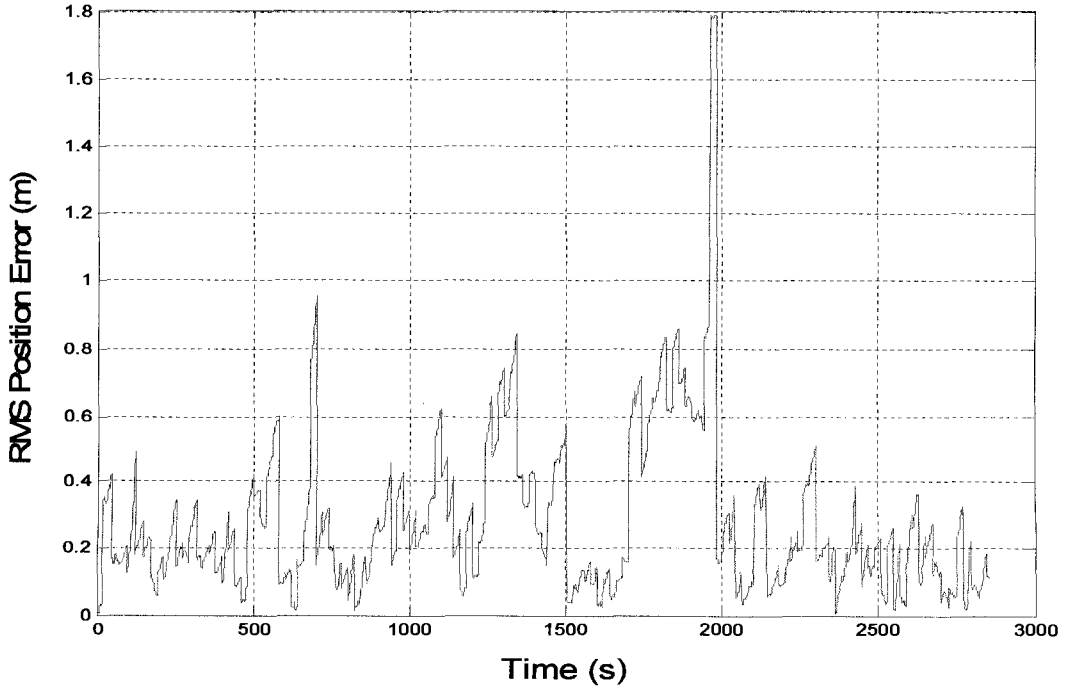


Figure 6.73: Simulation of the RMS position error with 110000 particles

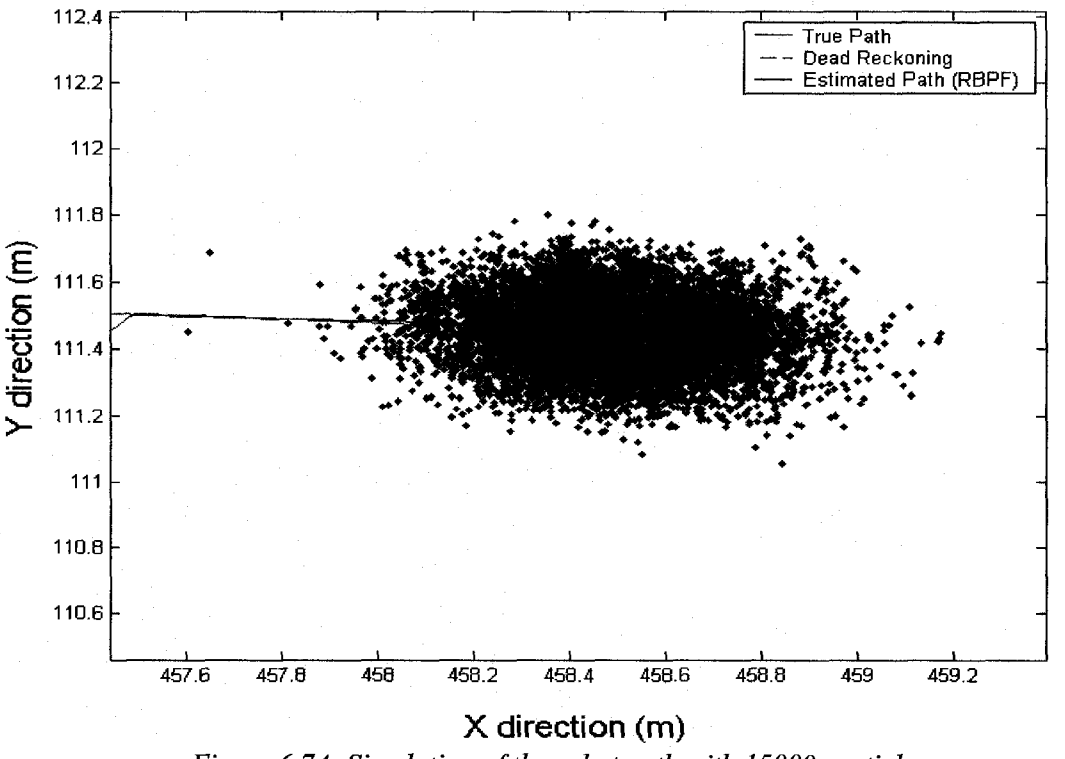


Figure 6.74: Simulation of the robot path with 15000 particles

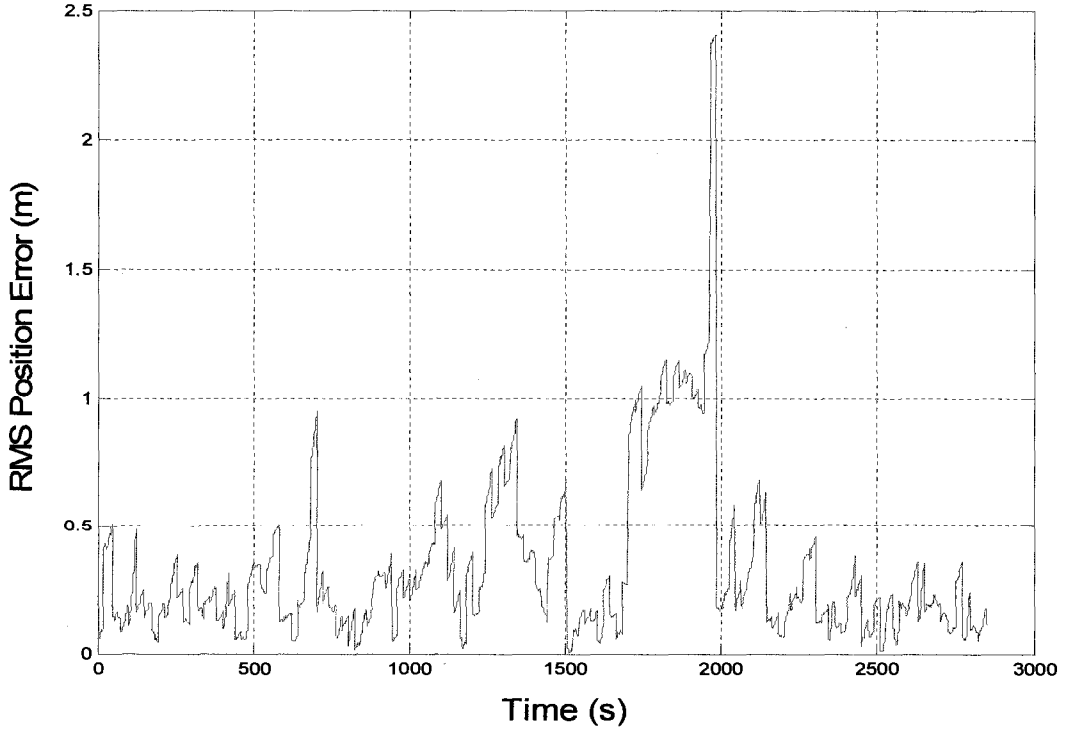


Figure 6.75: Simulation of the RMS position error with 15000 particles

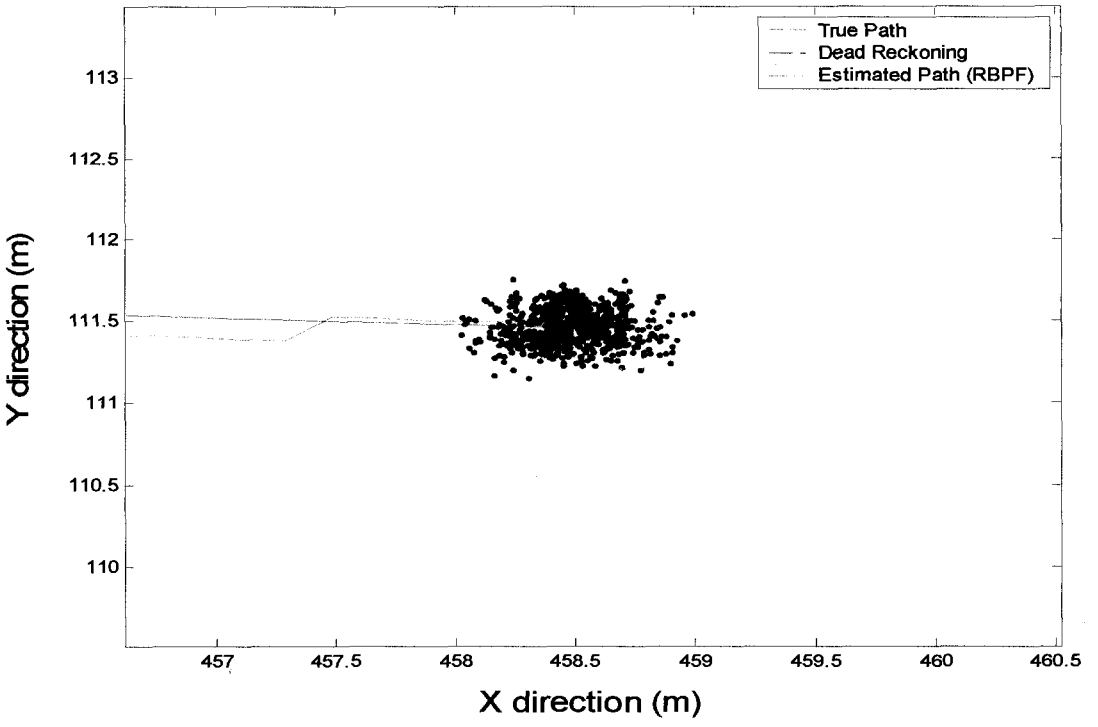


Figure 6.76: Simulation of the robot path with 1400 particles

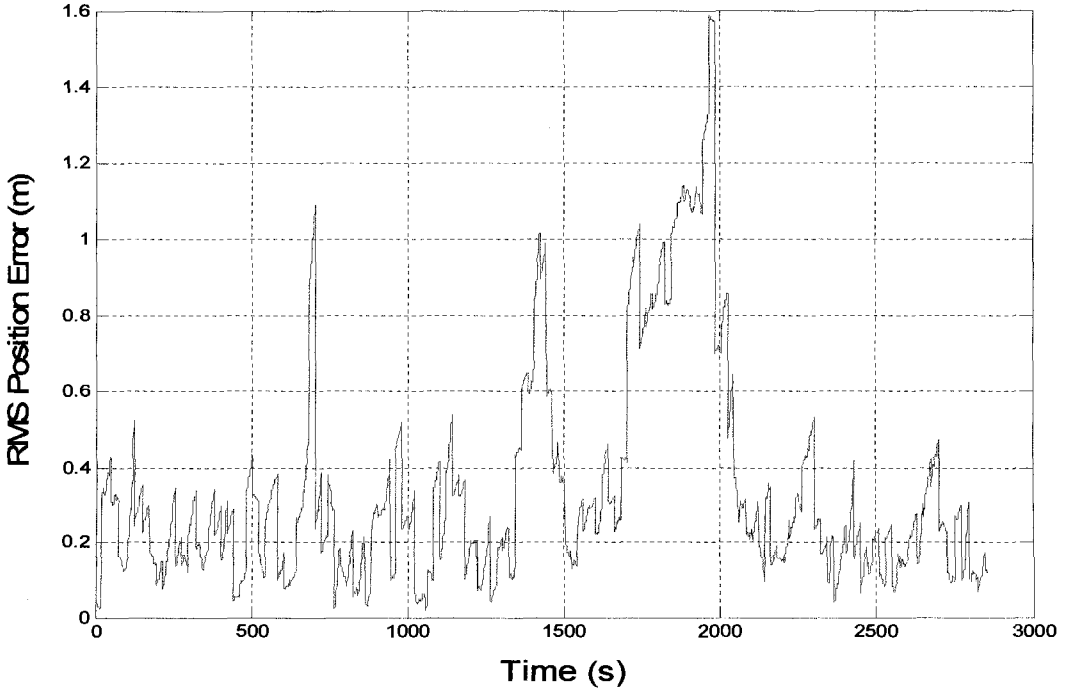


Figure 6.77: Simulation of the RMS position error with 1400 particles

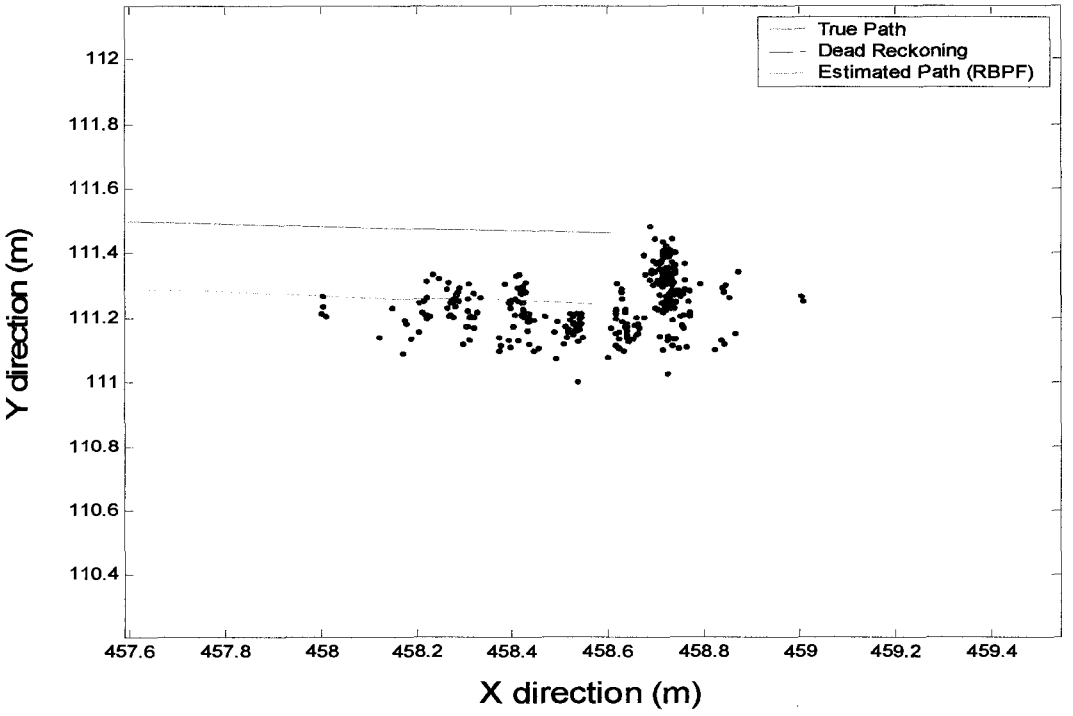


Figure 6.78: Simulation of the robot path with 107 particles

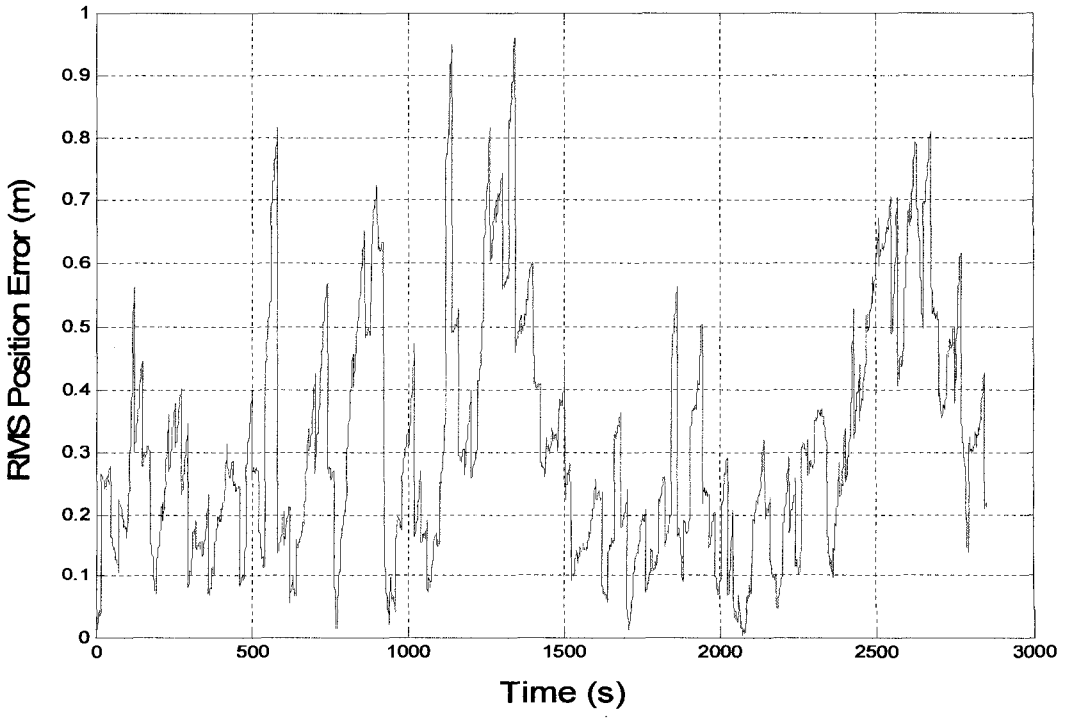


Figure 6.79: Simulation of the RMS position error with 107 particles

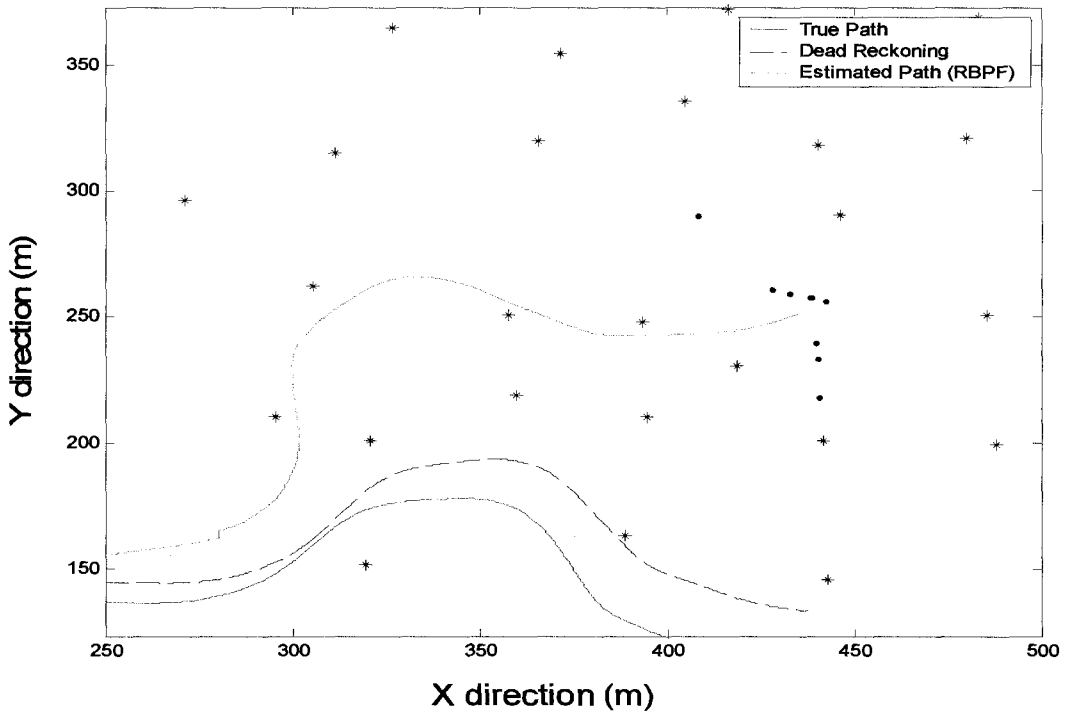


Figure 6.80: Simulation of the robot path with 10 particles

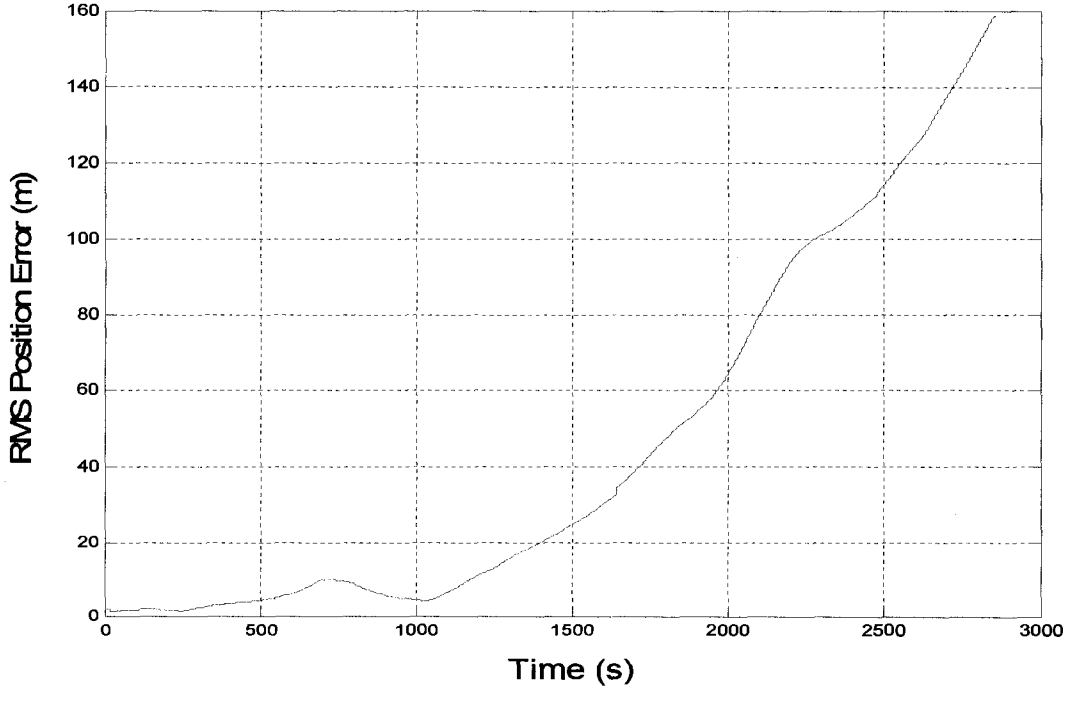


Figure 6.81: Simulation of the RMS position error with 10 particles

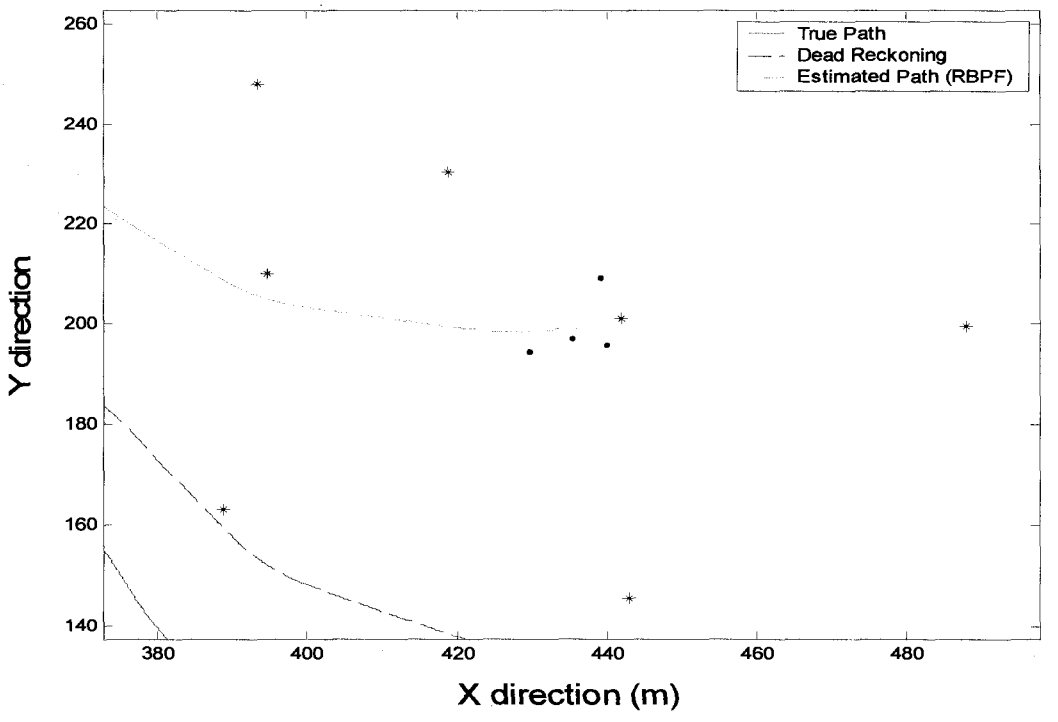


Figure 6.82: Simulation of the robot path with 4 particles

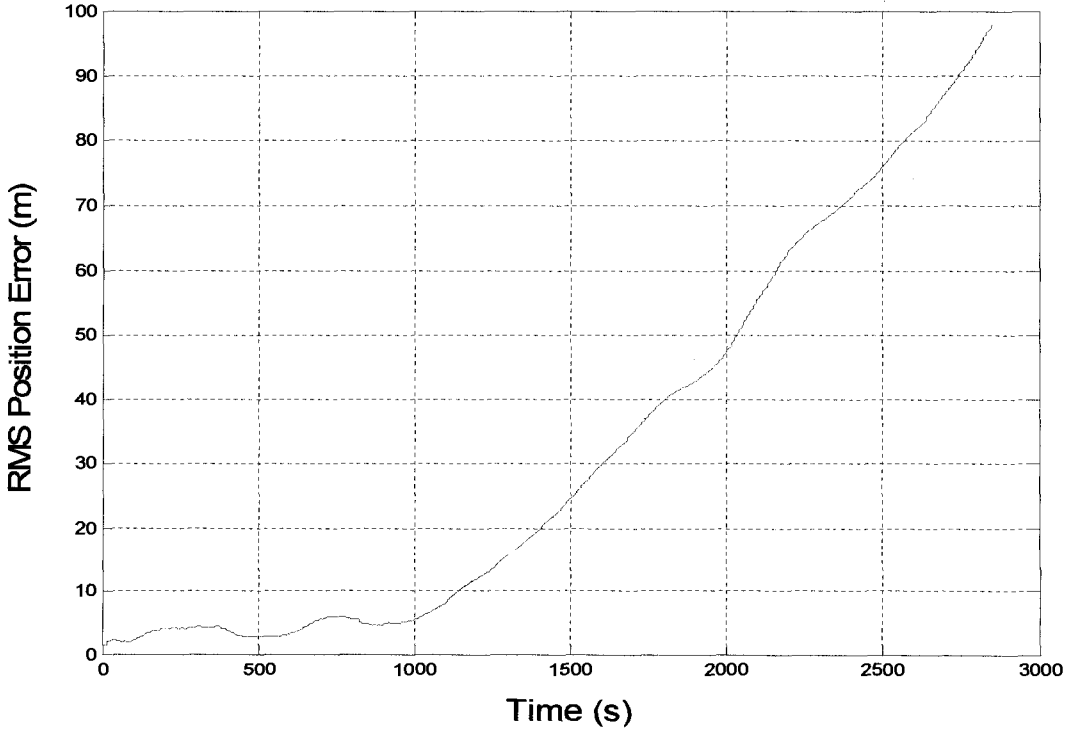


Figure 6.83: Simulation of the RMS position error with 4 particles

Figure 6.84 shows the results of 10 different runs for each amount of particles. The average RMS position error decreases as number of particles increase. For 300 particles the RMS position error average reaches upto 0.35 while for 200,000 particles this error only decreases 0.15 meters. In this case a difference of 0.020 radians will be appeared due to involving many particles instead of only a few hundereds. This figure also indicates time for simulation, memory used for the simulation and the average of root mean square position error for different number of particles. Obviously, increasing number of particles increases the simulation time substantially. The memory used for different number of particles does not make a big difference in the average position error. Likewise the time for the simulation makes a big difference on the simulation. Result shows that even with the amount of memory, the filter behaves linerly. Number of particles usually depends on the accuracy of application. Usually a 0.5m of RMS position error is suitable for most of real world applications and just a few hundred particles suffice for the FAST-SLAM algorithm to fulfill the task.

Number of Particles	Average RMS Position Error (m)	Average Orientation Error (radians)	Time (s)	Memory used (Mb)
200,000	0.20	0.024	2012	6
110,000	0.21	0.024	1444	4
90,000	0.22	0.025	918	3
55,000	0.23	0.025	319	2.3
25,000	0.24	0.026	255	1.8
15,000	0.25	0.027	118	1.7
10,000	0.26	0.028	52	1.6
6,000	0.28	0.029	27	1.5
3,000	0.29	0.03	27	1.4
1,400	0.3	0.032	25	1.3
1,000	0.31	0.033	18	1.3
700	0.33	0.039	16	1.3
300	0.35	0.044	16	1.2
104	0.38	0.053	14	1.1
50	0.86	0.061	13	1
10	45	0.083	13	1
4	80	0.089	9	1

Figure 6.84: Average RMS position error, simulation time and memory used for different amount of particle. For each amount of particles 10 runs were done and the average for each case was depicted.

Chapter 7

Conclusion

7.1 Conclusion

In this thesis, simultaneous localization and mapping (SLAM) was introduced and SLAM problem from a probabilistic point of view and based on Bayesian approach was thoroughly discussed. Since the map of environment is highly correlated with motion of the robot, imperfection in a noisy motion results in uncertainty in mapping. On the other hand, uncertainty appears in the system when the sensor is noisy. To solve the SLAM problem two alternative algorithms were presented; EKF-SLAM and FAST-SLAM.

Implementation of EKF as a classical solution was discussed under Gaussian assumptions of the system. EKF estimates the state of a linearizable dynamic system corrupted by Gaussian distributed noise which is assumed white, independent, and with zero-mean. This assumption leads us to the equations that form EKF-SLAM algorithm. The covariance matrix of the EKF was thourouly discussed and two major issues of EKF-SLAM in terms of Computational complexity and single hypothesis data association were investigated. Simulated results showed that EKF algorithm could be optimal, only if the system was linear or linearizable, meaning that for non-linear systems EKF-SLAM was unsuitable. Furthermore, EKF is limited to environments having a few hundered landmarks since at each time step, every two pairs of landmarks become correlated. Therefore, for environments with thousands of landmarks, EKF diverges. EKF does not have any mechanism in landmarks uncertainty due to its single hypothesis data association property. Since EKF-based SLAM has a quadratic complexity in

computation, a substantial amount of memory is demanded. By simulation results, the defects of EKF-SLAM algorithm in many different cases were also shown and compared to that of FAST-SLAM algorithm.

The alternative algorithm called FAST-SLAM based on Monte Carlo Localization by implementation of Rao-Blackwellised particle filtering (RBPF) was presented next. In this algorithm, the noise did not have to be considered with Gaussian assumptions. It however, could be considered any type of noise. FAST-SLAM samples over the robot's path instead of just relying on a single state at each time step. This alternative algorithm computes independent landmark estimates conditioned on each particle in a Dynamic Bayesian Network. Since landmarks in this network are mutually independent, there is not any correlation between any pairs of landmarks in the map. Therefore, the computation complexity can be presented in a linear scale. Here after, a logarithmic scale of FAST-SLAM was presented instead of the linear scale FAST-SLAM. The logarithmic presentation saves a significant amount of time and memory in the computation. The data association problem in FAST-SLAM was thourouly discussed and the aspects were investigated in the performance of FAST-SLAM application. Multiple hypothesis data association, results an uncertainty mechanism in FAST-SLAM algorithm. Due to this mechanism, FAST-SLAM is very suitable for an environment with very close landmarks.

Using Matlab codes, simulations of the robotic vehicle trajectory and system behavior were presented based on EKF and RBPF algorithms and two alternative SLAM algorithms were investigateed. The advantages of FAST-SLAM compared to that of EKF-SLAM in terms of memory usage, data association ambiguity and computational complexity were discussed. Performance of FAST-SLAM in a dence map was also shown and compared to EKF-SLAM

in the same situation. Finally different situations for the logarithmic FAST-SLAM algorithm in terms of accuracy of observation devices with a noisy motion were analyzed and some weak points of the algorithm were discussed.

7.2 Future Work

This work can be used as theoretical basis for further studies in a number of different ways. First, in the SLAM chapter, we discussed some SLAM methods implementing the Bayesian representation. This gives a starting point for further readings into the different SLAM methods. Secondly the filter chapters specially, RB particle filtering from a basis for further studies. With the basic derivations of FAST-SLAM it is interesting to look for more details into other extensions. Third, the experiments we performed in chapters 4 and 6, were by a simulator and merely meant as illustrations to concepts of filtering and path estimation. In order to draw conclusions practically, the theory discussed in this thesis can be implemented in practice.

Through out the process of this work, interesting ideas for future work came up. One of these is to look for ways to make SLAM problem applicable in dynamic environments, since the ral world is not static. In this work we discussed the use of both EKF and RBPF in SLAM problem using static landmarks of which it knows the location with some uncertainty. It is interesting to look at what will happen if the landmarks are not static, but move through the environment according to some even easy motion model. Multiple robots may together learn a rough motion model of moving objects and share their information regarding their own and the landmarks locations.

Bibliography

[1] S. B. Niku, "Introduction to Robotics: Analysis Systems, applications", Prentice Hall, USA, 2001.

[2] I. J. Cox, "Position estimation for an autonomous robot vehicle", Autonomous mobile robots: Control, Planning, and Architecture, IEEE Computer Society Press, Los Alamitos, Vol 2, pp. 285-292, Canada, 1991.

[3] L. Warren, "Robot Named Dante to Explore Inferno of Antarctic Volcano", The New York Times, December 8, pp. B7, 1992.

[4] P. M. Newman, J. J. Leonard, R. J. Rikoski, "Towards Constant-Time SLAM on an Autonomous Underwater Vehicle Using Synthetic Aperture Sonar", Proceedings of the 11th International Symposium on Robotics Research, Sienna, Italy, October, 2003.

[5] J. Z. Sasiadek, P. Hartana, "Sensor Fusion for dead-reckoning mobile robot navigation", Proceeding of the 6th IFAC Symposium on Robot Control, pp. 531-536, SYROCO, 2000.

[6] J. Borenstein, "Control and kinematic design of multi-degree-of-freedom robots with compliant linkage", IEEE Transactions on Robotics and Automation, 1995.
Available at http://citeseer.ist.psu.edu/borenstein92compliantlinkage.html

[7] D. Necsulesco, "Mechatronics", Prentice Hall, New Jersey 07458, USA, 2002.

[8] J. Borenstein, L. Feng, "Measurement and correction of systematic odometry errors in mobile robots", IEEE Transactions on Robotics and Automation, Vol 12, pp. 869-880, 1996.

[9] E. Nebot, J. Guivant, S. Baiker, "Autonomous navigation and map building using laser range sensors in outdoor applications", Journal of Robotic Systems, vol 17, no 10, pp. 565-583, October, 2000.

[10] J. Leonard, H. Durrant-Whyte, "A computational efficient method for large-scale concurrent mapping and localization", J. Hollerbach and D. Koditscheck, editors, Robotics Research, Proceedings of the 9th International Symposium (ISRR'99), pp. 169-176, Springer Verlag, 2000.

[11] M. Montemerlo, "FastSLAM: A factored solution to the simultaneous localization and mapping problem with unknown data association", PhD thesis, School of computer science, Carnegie Mellon University, Pittsburgh, USA, 2003.

[12] A. Singhal, "Issues in Autonomous mobile robot navigation", 1997. Available at http://citeseer.ist.psu.edu/cis?q=Autonomous+Mobile+Robot+Navigation+and+Learning.&cs=1

[13] A. Fujimori, P. N. Nkiforuk, M. M. Gupta, "Adaptive navigation of mobile robots with obstacle avoidance", Proceeding of IEEE Transaction on Robotics and Automation, vol 13, No. 4, August, 1997.

[14] H. Durrant-Whyte, T. Bailey, "Simultaneous localization and mapping (SLAM): part I the essential algorithms", IEEE Robatics and Automation Magazine, Vol 13, No. 2, pp. 99-108, June, 2006.

[15] R. Smith, M. Self, P. Cheesman, "Estimating uncertain spatial relationships in robotics", Autonomous robot vehicles, I. J. Coxand, G. T. Wilforn, Ed., pp. 167-193, Springer Verlag, 1990.

[16] R. Kalman, "A new approach to linear filtering and prediction problems", *Transaction* ASME Journal of Basic Engineering 82, pp. 35-44, 1960.

[17] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, "FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges", Proceeding of IJCAI, 2003.

[18] H. Durrant-Whyte, T. Bailey, "Simultaneous localization and mapping (SLAM): part II state of the art", Proceeding IEEE RAM. Vol 13, No. 3, pp. 108-117, September, 2006.

[19] J. A. Castellanos, J. M. M. Montiel, J. Neira, J. D. Tradós, "The SPmap: A probabilistic framework for simultaneous localization and map building", IEEE Transactions on Robotics and Automation, Vol. 15, No. 5, October, 1999. Available at http://www.ntu.edu.sg/home/eadams/Reading_Group_Webpage/paper/The%20SPmap--a%20probabilistic%20gramework%20for%20SLAM.pdf

[20] T. Bailey, J. Nieto, J. Guivant, M. Stevens, E. Nebot, "Cosistency of the EKF-SLAM Algorithm", Australian Centre for Field Robotics, University of Sydney, NSW, Australia. Available at http://www.acfr.usyd.edu.au/homepages/academic/tbailey/papers/ekfslam.pdf

[21] M. W. M. G. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, M. Csorba, "A solution to the simultaneous localization and map Building (SLAM) problem", IEEE Transactions on Rbotics and Automation. Vol 17, No. 3, pp. 229-241, June, 2001.

[22] T. Bailey, J. Nieto, E. Nebot, "Consistency of the FastSLAM algorithm", Australian Centre for Field Robotics, University of Sydney, NSW, Australia. Available at http://www.acfr.usyd.edu.au/homepages/academic/tbailey/papers/fastslam.pdf

[23] G. Grisetti, G. D. Tipaldi, C. Stachniss, W. Burgard, D. Nardi, "Fast and accurate SLAM with Rao-Blackwellised particle filters", Universities of Freiburg, La Sapienza and Swiss Federal Institute of Technology.

[24] M. Montemerlo, S. Thrun, "Simultaneous localization and mapping with unknown data association using FastSLAM". Available at http://www.cs.cmu.edu/afs/cs.cmu.edu/user/thrun/public_html/papers/montemerlo.fastslam-dataassoc03.pdf

[25] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto, E. Nebot, "FastSLAM: an efficient solution to the simultaneous localization and mapping problem with unknown data association", Journal of Machine Learning Research, 2004.

[26] J. Nieto, "Detailed environment representation for the SLAM problem", PhD thesis, Australian Centre for Field Robotics, University of Sydney, Sydney, Australia, 2005.

[27] S. Thrun, D. Fox, W. Burgard, "A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping", IEEE International Conference on Robotics and Automation, San Francisco, USA, April, 2000. Available at http://www.ai.mit.edu/courses/16.412J/thrun.map3d.pdf

[28] H. Durrant-Whyte, D. Rye, E. Nebot, "Localization on automatic guided vehicles", G. Giralt and G. Hirzinger, editors, Robotics Research, Proceedings of the 7th International Symposium (ISRR'95), pp. 613-625, Springer Verlag, 1996.

[29] R. Negenborn, "Robot localization and Kalman filters: on finding your position in a noisy world". MSc. thesis, Utrecht University, Netherland, 2003.

[30] S. Thrun, D. Fox, W. Burgard, "Markov localization for mobile robots in dynamic environments", Journal of Artificial Intelligence, Research 11, pp. 391-427, 1999.

[31] H. Durrant-Whyte, "Localization, mapping, and simultaneous localization and mapping (SLAM) problem", SLAM summer school notes, Centre for Autonomous Systems Kungl Tekniska Högskolan, SE-100 44 Stockholm, Sweden, 2002. Available at http://www.cas.kth.se/SLAM/Presentations/hdw-slides.pdf

[32] J. E. Guivant, E. M. Nebot, S. Baiker, "Localization and map building using laser range sensors in outdoor applications", Journal of Robotic Systems, 17(10): pp. 565-583, 2000.

[33] S. Thrun, d. Fox, W. Burgard, "A probabilistic approach to concurrent mapping and localization for mobile robots", Machine Learning 31, pp. 29-53 and Autonomous Robots 5, pp. 253-271, (joint issue), 1998.

[34] M. Csorba, "Simultaneous localization and map building", PhD thesis, University of Oxford, Oxford, United Kingdom, 1997.

[35] S. Thrun, "Probabilistic algorithms in robotics", AI Magazine 21, 4, pp. 93-109, 2000.

[36] A. Curran, K. Kyriakopoulos, "Sensor based self-localization for wheeled mobile robots", Proceedings of IEEE International Conference on Robotics and Automation, pp. 8-13, GA, Atlanta, May, 1993.

[37] R. van der Merwe, E. Wan, S. Julier, "Sigma-point Kalman Filters for nonlinear estimation and sensor-fusion: Applications to integrated navigation", Proceedings of the AIAA Guidance, Navigation and Control Conference, Providence, Rhode Island, August, 2004.

[38] J. Z. Sasiadek, Q. Wang, "Sensor fusion based on fuzzy Kalman filtering for autonomous robot vehicle", Proceedings of volume from the IFAC Workshop on

Mobile Robot Technology (J. Z. Sasiadek (Ed)), pp. 251-256, Pergamon, Oxford, 2001.

[39] A. Doucet, N. Freitas, K. Murphy, S. Russell, "Rao-Blackwellised particle filtering for dynamic Bayesian Networks", Proceedings of Conference on Uncertainty in Artificial Intelligence, 2000.

[40] R. G. Brown, P. Y. C. Hwang, "Introduction to random signals and applied Kalman filtering", second edition, John Wiley and Sons, Inc. New York, USA, 1992.

[41] M. S. Grewal, A. P. Andrews, "Kalman filtering: Theory and practice", Prentice-Hall, Englewood Cliffs, New Jersey 07632, 1993.

[42] P. S. Meybeck, "Stochastic models, estimation and control", academic press, Inc. New York, USA, 1979.

[43] Y. Bar-Shalom, T. Fortmann, "Tracking and data association", Academic press, Inc.1988.

[44] Z. Chen, J. Samarabandu, "Using multiple view geometery within extended Kalman filter framework for simultaneous localization and map-building", Proceedings of IEEE International Conference on Mechatronics and Automation, Niagara Falls, Canada, pp. 695-700, July-August, 2005.

[45] S. Thrun, D. Ferguson, D. Haehnel, M. Montemerlo, W. Burgard, R. Triebel, "A system for volumetric robotic mapping of abandoned mines", Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'03), 2003.

[46] S. J. Julier, J. K. Uhlmann, "A counter example to the theory of simultaneous localization and map building", Proceedings of IEEE Int. Conf. Robotics and Automation, pp. 4238-4243, 2001.

[47] J. Z. Sasiadek, P. Hartana, "Odometery and sensor data fusion for mobile robot navigation", Proceedings of the 6th IFAC Symposium on Robot Control, pp. 531-536, SYROCO, 2000.

[48] J. Guivant, E. Nebot, "Compressed filter for real time implementation of simultaneous localization and map building", Australian Centre for Field Robotics, Departemant of Mechanical and Mechatronic Engineering, University of Sydney, Sydney, Australia, NSW 2006. available at http://www.acfr.usyd.edu.au/publications/downloads/2000/Guivant145/fsr_hd_nebot.pdf

[49] A. Agent, "The advantages of absolute encoders for motion control", Sensors, pp. 19-24, April, 1991.

[50] J. E. Guivant, "Efficient simultaneous localization and mapping in large environments", PhD thesis, Australian Centre for Field Robotics, University of Sydney, Sydney, Australia, 2002.

[51] T. Bailey, "Mobile robot localization and mapping in extensive outdoor environments", PhD thesis, Australian Centre for Field Robotics, University of Sydney, Sydney, Australia, 2002.

[52] S. Thrun, D. Fox, W. Burgard, "Monte Carlo localization with mixture proposal distribution", Proceedings of AAAI National Conference on Artificial Intelligence, Austin, Texas, USA, 2000.

[53] D. B. Rubin, "Bayesian Statistics 3", Oxford University Press, 1988.

[54] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem", Proceedings of AAAI National Conference of Artificial Intelligence, 2002.

[55] S. Russell, P. Norvig, "Artificial intelligence: A modern approach", Prentice hall, 1995.

[56] W. Cochran, "Sampling techniques", Third Edition, John Wiley and Sons, Inc.1977.

Appendix A

Proof of the FAST-SLAM Factorization

We consider the path posterior of the robot explained in equation (5.4). If we show that for all non-negative values of k, $P(\mathbf{m} \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) = \prod_{i=1}^{M} P(\mathbf{m}_i \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k)$ we can simply reach to the factorization derivation. If landmark $\mathbf{m}_n$ is observed at time step k, according to Bayes rule we can have follwing probability equality.

$$P(\mathbf{m}_n \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k)\, P(\mathbf{z}_k \mid \mathbf{X}_k^R, \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) =$$

$$P(\mathbf{z}_k \mid \mathbf{m}_n, \mathbf{X}_k^R, \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k)\, P(\mathbf{m}_n \mid \mathbf{X}_k^R, \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) \qquad \text{(A.1)}$$

The observation $\mathbf{z}_k$ depends on the state of robot at time step k, and the landmark being observed by the robot. So the current state and control vectors, and also the current data association regarding the landmark observation are not involved in the probability terms of equation (A.1). Therefore, equation (A.1) can be re-written as

$$P(\mathbf{m}_n \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k)\, P(\mathbf{z}_k \mid \mathbf{X}_k^R, \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) =$$

$$P(\mathbf{z}_k \mid \mathbf{m}_i, \mathbf{X}_k^R, \mathbf{x}_0^R, \mathbf{d}_{k,n})\, P(\mathbf{m}_i \mid \mathbf{X}_{k-1}^R, \mathbf{Z}_{k-1}, \mathbf{U}_{k-1}, \mathbf{x}_0^R, \mathbf{d}_{k-1}) \qquad \text{(A.2)}$$

We rearange the equation (A.2) as follows

$$P(\mathbf{m}_n \mid \mathbf{X}_{k-1}^R, \mathbf{Z}_{k-1}, \mathbf{U}_{k-1}, \mathbf{x}_0^R, \mathbf{d}_{k-1}) =$$

$$\frac{P(\mathbf{z}_k \mid \mathbf{X}_k^R, \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k)}{P(\mathbf{z}_k \mid \mathbf{m}_i, \mathbf{x}_0^R, \mathbf{d}_{k,n})} P(\mathbf{m}_n \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) \qquad \text{(A.3)}$$

During navigation, if a landmark is not observed by the robot, according to Markov process, the prior and posterior observations of the landmark are the same, therefore;

$$P(\mathbf{m}_n \text{ not observed} \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) = P(\mathbf{m}_n \text{ not observed} \mid \mathbf{X}_{k-1}^R, \mathbf{Z}_{k-1}, \mathbf{U}_{k-1}, \mathbf{x}_0^R, \mathbf{d}_{k-1}) \qquad \text{(A.4)}$$

If we assume

$$P(\mathbf{m} \mid \mathbf{X}_{k-1}^R, \mathbf{Z}_{k-1}, \mathbf{U}_{k-1}, \mathbf{x}_0^R, \mathbf{d}_{k-1}) = \prod_{i=1}^{M} P(\mathbf{m}_i \mid \mathbf{X}_{k-1}^R, \mathbf{Z}_{k-1}, \mathbf{U}_{k-1}, \mathbf{x}_0^R, \mathbf{d}_{k-1}) \qquad \text{(A.5)}$$

For the zero time step, (when the robot starts moving), the factorization is trivally correct. For time steps $k>0$, and using Bayes rule, we can write

$$P(\mathbf{m} \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) =$$

$$\frac{P(\mathbf{z}_k \mid \mathbf{m}, \mathbf{X}_k^R, \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k)}{P(\mathbf{z}_k \mid \mathbf{X}_k^R, \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k)} P(\mathbf{m} \mid \mathbf{X}_k^R, \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) \qquad \text{(A.6)}$$

according to Markov process, the observation $\mathbf{z}_k$ depends oly on the map $\mathbf{m}$, stae $\mathbf{x}_k^R$, and the data association $\mathbf{d}_{k,i}$. Furthermore, position of the landmark does not depend on path stae $\mathbf{x}_k^R$, control $\mathbf{u}_k$, or even the data association $\mathbf{d}_{k,i}$. equation (A.6) can then be re-written as

$$P(\mathbf{m} \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) =$$

$$\frac{P(\mathbf{z}_k \mid \mathbf{m}_n, \mathbf{X}_k^R, \mathbf{x}_0^R, \mathbf{d}_{k,i})}{P(\mathbf{z}_k \mid \mathbf{X}_k^R, \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k)} P(\mathbf{m} \mid \mathbf{X}_{k-1}^R, \mathbf{Z}_{k-1}, \mathbf{U}_{k-1}, \mathbf{x}_0^R, \mathbf{d}_{k-1}) \tag{A.7}$$

According to (A.5) we can re-write equation (A.7) as

$$P(\mathbf{m} \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) =$$

$$\frac{P(\mathbf{z}_k \mid \mathbf{m}_n, \mathbf{X}_k^R, \mathbf{x}_0^R, \mathbf{d}_{k,i})}{P(\mathbf{z}_k \mid \mathbf{X}_k^R, \mathbf{Z}_{k-1}, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k)} \prod_{i=1}^{M} P(\mathbf{m}_i \mid \mathbf{X}_{k-1}^R, \mathbf{Z}_{k-1}, \mathbf{U}_{k-1}, \mathbf{x}_0^R, \mathbf{d}_{k-1}) \tag{A.8}$$

From (eqyuations (A.3) and (A.4) we may have

$$P(\mathbf{m} \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) =$$

$$P(\mathbf{m}_n \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) \prod_{i=not-observed}^{M} P(\mathbf{m}_i \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) \tag{A.9}$$

That is equal to the product of the individual landmark posteriors

$$P(\mathbf{m} \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) = \prod_{i=1}^{M} P(\mathbf{m}_i \mid \mathbf{X}_k^R, \mathbf{Z}_k, \mathbf{U}_k, \mathbf{x}_0^R, \mathbf{d}_k) \tag{A.10}$$

Appendix B

FAST-SLAM Algorithm with Unkown Data Association

$\mathbf{x}_k = \mathbf{x}_{aux} = \mathbf{0}$

for n = 1 to Þ // loop over all particles

retrieve n-th particle $\{ {}^{n}x^{R}_{k-1}, {}^{n}M_{k-1}, {}^{n}\mu_{k-1,1}, {}^{n}\sigma_{k-1,1}, \kappa, {}^{n}\mu_{k-1,{}^{n}M_{k-1}}, {}^{n}\sigma_{k-1,{}^{n}M_{k-1}} \}$ from $\mathbf{x}_{k-1}$

draw ${}^{n}x^{R}_{k} \sim P(x^{R}_{k} \mid \mathbf{u}_k, {}^{n}x^{R}_{k-1})$ // Sample new pose

for $i = 1$ **to** ${}^{n}M_{k-1}$ // loop over potential data association

$$\Lambda_k = \frac{\partial h(\mathbf{x})}{\partial \mathbf{z}_{k,i}}\Big|_{\mathbf{x}={}^{n}\mathbf{x}^{R}_{k}, \mathbf{z}_{k,i}={}^{n}\boldsymbol{\mu}_{k-1,i}}$$

$$\hat{\mathbf{z}}_k = h({}^{n}\mathbf{x}^{R}_{k}, \mu_{k-1,i})$$

$$\tilde{\mathbf{Z}}_k = \Lambda_k \, {}^{n}\Sigma_{k-1,i} \, \Lambda^{T}_{k} + \mathbf{R}_k$$

$${}^{n}p_{k,i} = \frac{1}{\sqrt{|2\pi\tilde{\mathbf{Z}}_{k,i}|}} \exp\left(-\frac{1}{2}\, \tilde{\mathbf{z}}^{T}_{k,i}\, \tilde{\mathbf{Z}}^{-1}_{k,i}\, \tilde{\mathbf{z}}_{k,i}\right)$$

end for

${}^{n}p_{{}^{n}M_{k-1}+1} = p_0$

$\hat{\mathbf{d}}_{k,j} =$ argmax ${}^{n}p_{k,i}$ **or** draw random $\hat{\mathbf{d}}_{k,j}$ with probability $\propto {}^{n}p_{k,i}$ // pick a data association

if $\hat{\mathbf{d}}_{k,j} = {}^{n}M_{k-1} + 1$ // is it a new feature?

${}^{n}M_k = {}^{n}M_{k-1} + 1$

${}^{n}\mu_{k,\hat{d}_k} = h^{-1}({}^{n}\mathbf{x}^{R}_{k}, \hat{\mathbf{z}}_{k,\hat{d}_k})$

${}^{n}\Sigma_{k,\hat{d}_k} = (\Lambda^{T}_{k,\hat{d}_k} \, \mathbf{R}^{-1}_{k} \, \Lambda^{T}_{k,\hat{d}_k})^{-1}$

else // or is a known feature?

${}^{n}M_k = {}^{n}M_{k-1}$

$K_{k,\hat{d}_k} = {}^{n}\Sigma_{k-1,\hat{d}_k} \, \Lambda^{T}_{k,\hat{d}_k} \, \tilde{\mathbf{Z}}^{-1}_{k,\hat{d}_k}$

${}^{n}\mu_{k,\hat{d}_k} = {}^{n}\mu_{k-1,\hat{d}_k} + K_{k,\hat{d}_k}(z_k - \hat{z}_{k,\hat{d}_k})$

${}^{n}\Sigma_{k,\hat{d}_k} = [I - K_{k,\hat{d}_k} \, \Lambda_{k,\hat{d}_k}] \, {}^{n}\Sigma_{k-1,\hat{d}_k}$

end if

for i=1 **to** ${}^{n}M_k$ do // handle unobserved features

if $d_k \neq \hat{d}_k$

$${}^{n}\mu_{k,m_i} = {}^{n}\mu_{k-1,m_i}$$

$${}^{n}\Sigma_{k,m_i} = {}^{n}\Sigma_{k-1,m_i}$$

end if
end for

$${}^{n}\hat{w}_k = {}^{n}p_{k,\hat{d}_k}$$

add $\{{}^{n}x_k^R, {}^{n}M_k, {}^{n}\mu_{k,1}, {}^{n}\sigma_{k,1}, \ldots, {}^{n}\mu_{k,{}^{n}M_{k-1}}, {}^{n}\sigma_{k,{}^{n}M_{k-1}}\}$ to $\mathbf{x}_{aux}$ // save weighted particle

end for
for n=1 **to** Þ // resample Þ new particles

draw random particle from x_{aux} with probability $\propto {}^{n}\hat{w}_k$
add new particles to $\mathbf{x}_k$
end for
return $\mathbf{x}_k$

Appendix C

Simulation Program Code for EKF-SLAM and Particle Filtering

The code is consisting of these files:

a_add

```
unction a=a_add(a1,a2)
%
% A=A_ADD(A1,A2)
% Add two angles so that the result
% always remains between +/- pi
%

a=a1+a2;

% get to within 2pi of the right answer
a=a - (2.0*pi*fix(a/(2.0*pi)));

% then leave a between +/- pi
while a>pi
  a=a-(2.0*pi);
end

while a< -pi
  a=a+(2.0*pi);
end
```

a_sub

```
function a=a_sub(a1,a2)
%
% A=A_SUB(A1,A2)
% Subtract two angles so that the result
% always remains between +/- pi
%

% simple function to get angle subtraction consistent


a=a1-a2;
```

```
% get to within 2pi of the right answer
a=a - (2.0*pi*fix(a/(2.0*pi)));

% then leave a between +/- pi
while a>pi
 a=a-(2.0*pi);
end

while a< -pi
 a=a+(2.0*pi);
end
```

get_beacons

```
function beacons=get_beacons
%
%  beacons=get_beacons
%
% HDW 28/04/00
% function to graphically input beacon locations
% beacons is 2*N matrix containing x,y locations of beacons


% set up the figure

globals;
figure(PLAN_FIG)
clf
v=[0 WORLD_SIZE 0 WORLD_SIZE];
axis(v);
hold on;
bin=1;
nbeacons=0;
beacons=zeros(1,2);

% now get beacons graphically until return
while bin
  [x,y]=ginput(1);
  bin= ~isempty(x);
  if bin
    nbeacons=nbeacons+1;
    plot(x,y,'go')
    beacons(nbeacons,1)=x;
    beacons(nbeacons,2)=y;
  end
```

```
end
hold off
```

get_control

```
function [xnext,unext]=get_control(x,u,perr,oerr,dt)
%
% [xnext,unext]=get_control(x,u,perr,oerr)
%
% A function to compute a new control vector for the vehicle
% and to do a one step vehicle prediction with this vector

globals;

unext=zeros(3,1);
xnext=zeros(4,1);

unext(1)=u(1);
unext(2)=KP*perr + KO*(oerr);
unext(3)=u(3)+dt;

xnext(1)=x(1) + dt*unext(1)*cos(x(3)+unext(2));
xnext(2)=x(2) + dt*unext(1)*sin(x(3)+unext(2));
xnext(3)=x(3) + dt*unext(1)*sin(unext(2))/WHEEL_BASE;
xnext(4)=x(4)+dt;
```

get_err

```
function [perr,oerr,index,dmin]=get_err(x,path,prev_idx)
%
%  [perr,oerr,index]=get_err(x,path,last_pt)
%
% A function to compute the position and orientation
% error of the vehicle with respect to a path.
%
% prev_idx is the index of the point in the path the
% vehicle was closest too the previous time step.
%
% If the vehicle follows the track with reasonable accuracy,
% the index of the closest point for the current time step
% will always be ahead of, or equal to, the previous index.
% In addition, the distance between the vehilce and the path
% should monotonically decrease as the index is incremented
% from prev_idx to the closest point on the path.  After this
% point, all subsequent indexs will result in an increased
% distance between the vehilce and the path.
```

```
[rows,npath]=size(path);
globals;

%dmin=WORLD_SIZE;
%dmin=WORLD_SIZE*WORLD_SIZE;
dx=path(1,prev_idx)-x(1);
dy=path(2,prev_idx)-x(2);
dmin=dx*dx + dy*dy;
index = prev_idx;

for i=prev_idx+1:(npath-1)
        dx=path(1,i)-x(1);
        dy=path(2,i)-x(2);

%       d=sqrt(dx*dx + dy*dy);
        d=dx*dx + dy*dy;

        if d <= dmin
                dmin=d;
                index=i;
        else
                break;
        end
end
dmin = sqrt(dmin);

dpx=path(1,index+1)-path(1,index);
dpy=path(2,index+1)-path(2,index);
dvx=x(1)-path(1,index);
dvy=x(2)-path(2,index);
perr=dpy*dvx - dpx*dvy;
phiest=atan2(path(2,index+1)-path(2,index),path(1,index+1)-path(1,index));
oerr=(phiest-x(3));
```

get_path

```
function path=get_path(beacons)
%
%  path=get_path(beacons)
%
% HDW 28/04/00
% Function to graphically acquire spline points
% and compute spline fit path for vehicle
% plan is in integer figure handle
% beacons is a 2*N array of x,y beacon locations
```

```
% path is a 2*M array of x,y path points
% get_path uses a call to 'globals' for various fixed values
% set up the figure

globals;
figure(PLAN_FIG)
clf
v=[0 WORLD_SIZE 0 WORLD_SIZE];
axis(v);
hold on;
plot(beacons(:,1),beacons(:,2),'go')
pin=1;
npoints=0;
points=zeros(2,1);
xi=[];

% get input points graphically
% and then set up basis x values
% interpolation can get confused so be careful !
while pin
  [x,y]=ginput(1);
  pin= ~isempty(x);
  if pin
    npoints=npoints+1;
    plot(x,y,'rx')
    points(1,npoints)=x;
    points(2,npoints)=y;
    % now find a basis for x
    if npoints > 1
          dx=points(1,npoints)-points(1,npoints-1);
        dy=points(2,npoints)-points(2,npoints-1);
        length=sqrt(dx*dx + dy*dy);
        xincs=points(1,npoints-1):LINC*dx/length:points(1,npoints);
        xi=[xi xincs];
    end
  end
end
% now we have all the basis points, interpolate the
% path in y. A better method would be to do this along
% the arc length; would need to think how to do this !
yi=interp1(points(1,:),points(2,:),xi,'spline');
plot(xi,yi,'r')
path=[xi;yi];
hold off
```

ginit

```
% values for global variables
% these are defined in globals.m
PLAN_FIG=1;                    % default handle for plan figure
WORLD_SIZE=500;  % 0-100 meters in each direction
LINC=0.1;                % 0.1m length for spline interpolation
DT=0.1;                        % Sample interval for controller
TEND=5000*DT;    % Total maximum run time for simulator
VVEL=2;                        % Vehicle velocity (assumed constant)

KP=1;                          % vehicle position error gain
KO=1;                          % vehicle orientation error gain

WHEEL_BASE=1;            % vehicle wheel base (m)
WHEEL_RADIUS=0.3;  % nomial wheel radius (m)
R_OFFSET=0.0;            % radar offset (m)
R_MAX_RANGE=100.0;% maximum range (m)
R_RATE=12.566;           % rotation rate (rads/s)

GSIGMA_RANGE=0.25;      % Range SD (m) used for observation generation
GSIGMA_BEARING=0.0174;% Bearing SD (rads) used for observation generation
GSIGMA_WHEEL=0.1;       % wheel variance used for control generation
GSIGMA_STEER=0.035;     % steer variance used for control generation

SIGMA_Q=0.25;                  % Multiplicative Wheel Noise SD (percent)
SIGMA_W=0.1;                   % Additive Wheel Noise SD (rads/s)
SIGMA_S=0.01;                  % Mutiplicative steer noise SD (percent)
SIGMA_G=0.0087;          % Additive steer noise SD (rads)
SIGMA_R=0.005;           % wheel radius SD noise (m)
SIGMA_RANGE=0.3;               % Range Variance (m)
SIGMA_BEARING=0.035; % bearing variance (rads)

fact=0.1;
SIGMA_Q=SIGMA_Q*fact;
SIGMA_W=SIGMA_W*fact;
SIGMA_S=SIGMA_S*fact;
SIGMA_G=SIGMA_G*fact;
SIGMA_R=SIGMA_R*fact;
```

Globals

```
% definitions of global variables
% values for these are set in ginit.m
global PLAN_FIG;         % handle for main plan figure
global WORLD_SIZE;       % size of the world (for display purposes only).
global LINC;             % increment along which spline path is evaluated.
```

```
global DT;                          % Sample interval for controller
global TEND;        % Total maximum run time for simulator
global VVEL;                % vehicle velocity

global KP;                          % vehicle position error gain
global KO;                          % vehicle orientation error gain

global WHEEL_BASE;          % vehicle wheel base (m)
global WHEEL_RADIUS; % nominal wheel radius
global R_OFFSET;            % radar offset along centre axis
global R_MAX_RANGE;  % maximum radar range
global R_RATE;                      % rotation rate of radar

global GSIGMA_RANGE      % Range SD (m) used for observation generation
global GSIGMA_BEARING % Bearing SD (rads) used for observation generation
global GSIGMA_WHEEL      % wheel variance used for control generation
global GSIGMA_STEER   % steer variance used for control generation

global SIGMA_Q                      % Multiplicative Wheel Noise SD (percent)
global SIGMA_W                      % Additive Wheel Noise SD (rads/s)
global SIGMA_S                      % Mutiplicative steer noise SD (percent)
global SIGMA_G              % Additive steer noise SD (rads)
global SIGMA_R              % wheel radius SD noise (m)
global SIGMA_RANGE          % Range Variance (m)
global SIGMA_BEARING % bearing variance (rads)
```

Ekfilter

```
function [xest, Pest, xpred, Ppred,innov,innvar]=kfilter(obs,u,xinit,Pinit,beacons)

%
% [xest, Pest, xpred, Ppred,innov,innvar]=kfilter(obs,u,xinit,Pinit,beacons)

% HDW 3/1/95 modified 31/05/00
% complete navigation filter
% each row of obs contains an observation of range bearing and beacon index
% each row of u contains omega gamma and time
% obs and u should have the smae number of rows

globals;

[temp,N_OBS]=size(obs);
if temp ~= 4
  error('observation dimension of 4 expected')
end
```

```
[temp,N_U]=size(u);
if temp ~= 3
 error('control input vector of dimension 3 expected')
end
if N_U ~= N_OBS
 error('control and observation sequences of different length')
end

[XSIZE,temp]=size(xinit);
if temp ~= 1
 error('xinit expected to be a column vector')
end

[s1,s2]=size(Pinit);
if((s1 ~= XSIZE)|(s2 ~=XSIZE))
 error('Pinit not of size XSIZE')
end

% make some space
xpred=zeros(XSIZE,N_U);
xest=zeros(XSIZE,N_U);
innov=zeros(2,N_U);
innvar=zeros(2,2,N_U);
Ppred=zeros(XSIZE,XSIZE,N_U);
Pest=zeros(XSIZE,XSIZE,N_U);
% returns from pred and update are in the form of column vectors
xe=xinit;
Pe=Pinit;
time=0;
for i=1:N_OBS
  dt=u(3,i)-time;
  time=u(3,i);
  [xp Pp]=pred(xe,Pe,dt,u(:,i));
  [xe Pe in ins]=update(xp,Pp,obs(:,i),beacons);
  xpred(:,i)=xp;
  xest(:,i)=xe;
  innov(:,i)=in;
  Ppred(:,:,i)=Pp;
  Pest(:,:,i)=Pe;
  innvar(:,:,i)=ins;
end
```

obs_seq

```
function obs=obs_seq(state,beacons)
```

```
% HDW 15/12/94, modified 17/05/00
% function to gather a complete observation sequence
% inputs are

globals;

% state vector, containing x,y,phi,t
[XSIZE,N_STATES]=size(state);
if XSIZE ~= 4
 error('Incorrect state dimension')
end

obs=zeros(4,N_STATES);

for i=1:N_STATES-1
 start_loc=state(:,i);
 end_loc=state(:,i+1);
 obs(:,i)=rad_sim(start_loc,end_loc,beacons);
end
```

onestep

```
% A script file to look in detail at one step in the
% filter calculation. Requires complete filter to have been
% run previously

dt=utrue(3,tstep)-utrue(3,tstep-1);
time=utrue(3,i);
[xp1 Pp1]=pred(xest(:,tstep-1),squeeze(Pest(:,:,tstep-1)),dt,utrue(:,tstep));
[xe1 Pe1 in1 ins1]=update(xp1,Pp1,obs(:,tstep),beacons);
```

p_obs

```
function [obs_p,state_p]=p_obs(obs,state)

% function to place vehicle centered observations in
% global coordinates for checking of consistency

globals;
[emp,OSIZE]=size(obs);
[temp,SSIZE]=size(state);

if (SSIZE ~= OSIZE)
 error('Unmatched state and observation dimensions')
end
```

```
obs_p=zeros(2,SSIZE);
state_p=zeros(2,SSIZE);


numobs=0;
for i=1:SSIZE
  if obs(3,i) ~= 0
    numobs=numobs+1;
    obs_p(1,numobs)=state(1,i)+(obs(1,i).*cos(obs(2,i)+state(3,i)));
    obs_p(2,numobs)=state(2,i)+(obs(1,i).*sin(obs(2,i)+state(3,i)));
    state_p(1,numobs)=state(1,i);
    state_p(2,numobs)=state(2,i);
  end
end

obs_p=obs_p(:,1:numobs);
state_p=state_p(:,1:numobs);
```

plots

```
replot

figure(2)
clf
hold on;
v=[0 WORLD_SIZE 0 WORLD_SIZE];
axis(v);
%plot(beacons(:,1),beacons(:,2),'go')
[obs_p, state_p]=p_obs(obs,xtrue);
plot(xtrue(1,:),xtrue(2,:),'b',obs_p(1,:),obs_p(2,:),'rx')
legend('True Vehicle Path','Beacon Observations')
title('True Vehicle Path and Beacon Observations')
xlabel('X distance (m)')
ylabel('Y distance (m)')
hold off


figure(3)
clf
hold on
plot(xtrue(4,:),abs(xtrue(1,:)-xest(1,:)),'r',xtrue(4,:),sqrt(squeeze(Pest(1,1,:))),'b')
legend('True Error','Estimated Error')
xlabel('Time (s)')
ylabel('Error (m)')
title('X Direction Error')
hold off
```

```
figure(4)
clf
hold on
plot(xtrue(4,:),abs(xtrue(2,:)-xest(2,:)),'r',xtrue(4,:),sqrt(squeeze(Pest(2,2,:))),'b')
legend('True Error','Estimated Error')
xlabel('Time (s)')
ylabel('Error (m)')
title('Y Direction Error')
hold off

figure(5)
clf
hold on
plot(xtrue(4,:),abs(xtrue(3,:)-xest(3,:)),'r',xtrue(4,:),sqrt(squeeze(Pest(3,3,:))),'b')
legend('True Error','Estimated Error')
xlabel('Time (s)')
ylabel('Error (rads)')
title('Orientation Error')
hold off

figure(6)
clf
hold on
plot(xtrue(4,:),xest(4,:),'r',xtrue(4,:),WHEEL_RADIUS+sqrt(squeeze(Pest(4,4,:))),'b',xtrue(4,:
),WHEEL_RADIUS-sqrt(squeeze(Pest(4,4,:))),'b')
legend('Estimated Wheel Radius','Estimated Wheel Radius Error')
title('Estimated Wheel Radius and Wheel Radius Error')
xlabel('Time (s)')
ylabel('Wheel Radius (m)');
hold off

%[inn,sinn]=proc_innov(innov,innvar,obs);
[inn,sinn]=track_innov(innov,innvar,obs,xest,utrue);

figure(7)
clf
hold on
plot(inn(3,:),inn(1,:),'bx')
plot(inn(3,:),inn(1,:),'b')
plot(inn(3,:),sinn(1,:),'g')
plot(inn(3,:),-sinn(1,:),'g')
title('Long-Track Innovation and Innovation Standard Deviation')
xlabel('Time (s)')
ylabel('Innovation (m)');
hold off
```

```
figure(8)
clf
hold on
plot(inn(3,:),inn(2,:),'bx')
plot(inn(3,:),inn(2,:),'b')
plot(inn(3,:),sinn(2,:),'g')
plot(inn(3,:),-sinn(2,:),'g')
title('Cross-Track Innovation and Innovation Standard Deviation')
xlabel('Time (s)')
ylabel('Innovation (m)');
hold off

[rows,cols]=size(xest);

rho=zeros(6,cols);

for i=1:cols
  %x-y
  rho(1,i)=Pest(1,2,i)/sqrt(Pest(1,1,i)*Pest(2,2,i));
  %x-phi
  rho(2,i)=Pest(1,3,i)/sqrt(Pest(1,1,i)*Pest(3,3,i));
  %y-phi
  rho(3,i)=Pest(2,3,i)/sqrt(Pest(2,2,i)*Pest(3,3,i));
  %x-R
  rho(4,i)=Pest(1,4,i)/sqrt(Pest(1,1,i)*Pest(4,4,i));
  %y-R
  rho(5,i)=Pest(2,4,i)/sqrt(Pest(2,2,i)*Pest(4,4,i));
  %Phi-R
  rho(6,i)=Pest(3,4,i)/sqrt(Pest(3,3,i)*Pest(4,4,i));

end

figure(9)
clf
hold on
plot(xtrue(4,:),rho(1,:))
xlabel('Time (s)')
ylabel('Correlation Coefficient')
title('X-Y Correlation Coefficient')
hold off

figure(10)
clf
hold on
plot(xtrue(4,:),rho(2,:),'b',xtrue(4,:),rho(3,:),'r')
```

```
legend('X to Orientation Correlation','Y to Orientation Correlation')
xlabel('Time (s)')
ylabel('Correlation Coefficient')
title('Position to Orientation Correlation Coefficient')
hold off

figure(11)
clf
hold on
plot(xtrue(4,:),rho(4,:),'b',xtrue(4,:),rho(5,:),'r')
legend('X to Wheel Radius Correlation','Y to Wheel Radius Correlation')
xlabel('Time (s)')
ylabel('Correlation Coefficient')
title('Position to Wheel Radius Correlation Coefficient')
hold off

figure(12)
clf
hold on
plot(xtrue(4,:),rho(6,:),'b')
xlabel('Time (s)')
ylabel('Correlation Coefficient')
title('Orientation to Wheel Radius Correlation Coefficient')
hold off
```

pred

```
function [xpred, Ppred]=pred(xest,Pest,dt,u)
%
%[xpred, Ppred]=pred(xest,Pest,dt,u)
%
% HDW 16/12/94 modified 31/05/00
% Function to generate a one-step vehicle prediction from
% previous estimate and control input.

% definitions and checks

globals;

[XSIZE,temp]=size(xest);
if temp ~= 1
  error('xest expected to be a column vector')
end

[s1,s2]=size(Pest);
if((s1 ~= XSIZE)|(s2 ~=XSIZE))
```

```
 error('Pest not of size XSIZE')
end

% set parameters
B=WHEEL_BASE;
% SD's to variances */
sigma_q=SIGMA_Q*SIGMA_Q;
sigma_w=SIGMA_W*SIGMA_W;
sigma_s=SIGMA_S*SIGMA_S;
sigma_g=SIGMA_G*SIGMA_G;
sigma_r=SIGMA_R*SIGMA_R; % wheel radius variance

% make some space
xpred=zeros(XSIZE,1);
Ppred=zeros(XSIZE,XSIZE);

% first state prediction
xpred(1)=xest(1) + dt*xest(4)*u(1)*cos(xest(3)+u(2));
xpred(2)=xest(2) + dt*xest(4)*u(1)*sin(xest(3)+u(2));
xpred(3)=xest(3) + dt*xest(4)*u(1)*sin(u(2))/B;
xpred(4)=xest(4);

% state transition matrix evaluation
F=[1 0 -dt*xest(4)*u(1)*sin(xest(3)+u(2)) dt*u(1)*cos(xest(3)+u(2));
  0 1  dt*xest(4)*u(1)*cos(xest(3)+u(2)) dt*u(1)*sin(xest(3)+u(2));
  0 0       1                    dt*u(1)*sin(u(2))/B;
  0 0       0                         1         ];

% source error transfer matrix
G=dt*[cos(xest(3)+u(2)) -sin(xest(3)+u(2)) 0;
  sin(xest(3)+u(2))  cos(xest(3)+u(2)) 0;
  sin(u(2))/B       cos(u(2))/B  0;
      0              0          1];

% source error covariance
sigma=[(sigma_q*(xest(4)*u(1))^2)+(sigma_w*(xest(4))^2) 0 0;
     0 (sigma_s*(xest(4)*u(1)*u(2))^2)+(sigma_g*(xest(4)*u(1))^2) 0;
     0                  0                          sigma_r];

   % stabalistation noise

stab=[0.000 0 0 0; 0 0.000 0 0; 0 0 0 0; 0 0 0 0];
% Now compute prediction covariance
Ppred=F*Pest*F' + G*sigma*G'+stab;
```

proc_innov

```
function [binnov, binnovsig]=proc_innov(innov,innovar,obs,bnum)
%
% function [binnov, binnovsig]=proc_innov(innov,innovar,obs,bnum)
%
% a function to process innovation results, extracting (optionally)
% innovations for specific beacons and standarad deviations

[temp,nsamps]=size(obs);
binnov=zeros(3,nsamps);
binnovsig=zeros(2,nsamps);
j=0;

if nargin==4
        for i=1:nsamps
        if obs(3,i) == bnum
        j=j+1;
        binnov(1:2,j)=innov(:,i);
        binnov(3,j)=obs(4,i);
        binnovsig(1,j)=sqrt(innovar(1,1,i));
        binnovsig(2,j)=sqrt(innovar(2,2,i));
        end
        end
else
        for i=1:nsamps
        if obs(3,i) ~= 0
        j=j+1;
        binnov(1:2,j)=innov(:,i);
        binnov(3,j)=obs(4,i);
        binnovsig(1,j)=sqrt(innovar(1,1,i));
        binnovsig(2,j)=sqrt(innovar(2,2,i));
        end
        end
end
binnov=binnov(:,1:j);
binnovsig=binnovsig(:,1:j);
```

rad_sim

```
function [obs,b]=rad_sim(start_loc,end_loc,beacons)

% [obs,b]=rad_sim(start_loc,end_loc,beacons)
%
% HDW 15/12/94, modified 17/05/00
% function to simulate radar observations
% takes a start and end location for the vehicle,
```

```
% the vehicle parameter set and a map of beacons.
% Returns an observation if one is made during this
% time slot.
% output observation vector is a row vector containing
% range, bearing and index.

globals;
obs=zeros(4,1);

% The location vectors start_loc and end_loc each consist
% of a vector of x,y,phi,t, describing the location of the
% vehicle at a specific time.
[XSIZE,temp]=size(start_loc);
if((XSIZE ~= 4)+(temp ~= 1))
  error('Incorrect size for start_loc')
end
[XSIZE,temp]=size(end_loc);
if((XSIZE ~= 4)+(temp ~= 1))
  error('Incorrect size for end_loc')
end

% The beacon map consists of an array of x,y locations
% for the beacons
[N_BEACONS,temp]=size(beacons);
if (temp ~= 2)
  error('Incorrect Size for beacon map')
end

% The algorithm now proceeds by finding the two bounding
% lines from radar to max range. If the beacon lies between these
% two lines, then it will be observed

% first find location and aim of radar at start and end
radx1=start_loc(1) + (R_OFFSET*cos(start_loc(3)));
rady1=start_loc(2) + (R_OFFSET*sin(start_loc(3)));
radphi1=a_add(start_loc(4)*R_RATE,0.0); % gets normalized angle
radx2=end_loc(1) + (R_OFFSET*cos(end_loc(3)));
rady2=end_loc(2) + (R_OFFSET*sin(end_loc(3)));
radphi2=a_add(end_loc(4)*R_RATE,0.0); % gets normalized angle

% construct line segments for checking beacon view
dx1=R_MAX_RANGE*cos(radphi1+start_loc(3));
dy1=R_MAX_RANGE*sin(radphi1+start_loc(3));
dx2=R_MAX_RANGE*cos(radphi2+end_loc(3));
dy2=R_MAX_RANGE*sin(radphi2+end_loc(3));
% To be observed, the beacon must lie between these two lines
```

```
% the test is simply that the dot products are positive and
% negative respectively. The nearest beacon within max range is
% recorded.
range=R_MAX_RANGE;
b=0;
for i=1:N_BEACONS
 r=sqrt((beacons(i,1)-radx1)^2+(beacons(i,2)-rady1)^2);
 d1=((beacons(i,2)-rady1)*dx1) - ((beacons(i,1)-radx1)*dy1);
 d2=((beacons(i,2)-rady2)*dx2) - ((beacons(i,1)-radx2)*dy2);
 if((r<R_MAX_RANGE)*(d1>0)*(d2<0)*(r<range))
   range=r;
   b=i;
 end
end

% if a beacon is found, assume it was seen from start_loc:
if b>0
 obs(1)=sqrt((radx1-beacons(b,1))^2+(rady1-beacons(b,2))^2);
 obs(2)=a_sub(atan2(beacons(b,2)-rady1,beacons(b,1)-radx1),start_loc(3));
 obs(3)=b; % beacon index
 obs(4)=start_loc(4); % time stamp
 % noise model
 obs(1)=obs(1)+ GSIGMA_RANGE*randn(size(obs(1)));
 obs(2)=obs(2)+ GSIGMA_BEARING*randn(size(obs(2)));
else
 obs(1)=0.0;
 obs(2)=0.0;
 obs(3)=0;
 obs(4)=0;
end
```

replot

```
figure(13)
clf
v=[0 WORLD_SIZE 0 WORLD_SIZE];
axis(v);
hold on;
plot(xtrue(1,:),xtrue(2,:),'r',beacons(:,1),beacons(:,2),'go')
legend('True Trajectory','Beacon Locations')
title('Vehicle Trajectory and Beacon Locations')
xlabel('X distance (m)')
ylabel('Y distance (m)')
hold off
```

run

```
dt=0.1;
TEND=1000*dt;
% initialise
x=[];
u=[];
x(1)=path(1,1);
x(2)=path(2,1);
x(3)=atan2(path(2,2)-path(2,1),path(1,2)-path(1,1));
x(4)=0;
u(1)=4; % velocity set at 2 m/s
u(2)=0;
u(3)=0;
xlog=x;
ulog=u;
perrlog=0;
oerrlog=0;
% loop control
for t=0:dt:TEND
  % find error
  [perr,oerr,index,d]=get_err(x,path);
  % compute next state
  [x,u]=get_control(x,u,perr,oerr,dt);
  xlog=[xlog;x];
  ulog=[ulog;u];
  if d >10
    sprintf('simulation terminating at time %f\n',t)
    break;
  end
  %perrlog=[perrlog;perr];
  %oerrlog=[oerrlog;oerr];
end
```

run_filt

```
globals;
ginit;

disp('Press return to run filter');
pause;

%initial_errors=[5; 5; 0.2; 0.1]
% initialise filter
xinit=xtrue(:,1);
xinit(4,1)=WHEEL_RADIUS;
%xinit=xinit+initial_errors;
```

```
Pinit= [0.024 0.0   0.0     0.0;
       0.0   0.024 0.0     0.0;
       0.0   0.0   0.00025 0.0;
       0.0   0.0   0.0     0.0001];

%run filter
[xest,Pest,xpred,Ppred,innov,innvar]=kfilter(obs,uz,xinit,Pinit,beacons);

disp('Completed Filtering')
```

run_obs

```
% this is an example run of the vehicle state filter
% initialise global values

globals;
ginit;

% first get all observations
disp('Beginning observation simulations');
obs=obs_seq(xtrue,beacons);
disp('Completed Simulation');

% place observations in a global coordinate system
[obs_p, state_p]=p_obs(obs,xtrue);
figure(PLAN_FIG);
hold on
plot(obs_p(1,:),obs_p(2,:),'rx');
% plot(state_p(1,:),state_p(2,:),'r+');
hold off
```

run_pfilter

```
% A Simple Particle Filter for localization
%
% author : Mike Montemerlo (CMU) (mmde@cs.cmu.edu)
%
% This code depends on the robot trajectory and sensor simulator
% code written by Hugh Durrant-Whyte
%
% First, run set_up
% then, run_obs
% then, run_pfilter
% All particle filter parameters are at the top of this file.
```

```
globals;

% number of particles
NUM_PARTICLES       =       1000;

% process every nth observation
PERCEPTION_SKIP     =       5;

% resample every nth observation processed
RESAMPLE_SKIP       =       4;

% wait for keyboard returns between resamplings
INTERACTIVE         =       1;

% parameters of robot motion model
SIGMA_VELOCITY      =       0.1;
SIGMA_STEERING      =       0.035;

% parameters of robot sensor model
SIGMA_RANGE         =       0.25
SIGMA_BEARING       =       (1.0 * pi / 180.0);
SIGMA_RADIUS        =       0.001;

% initialization parameters
SIGMA_X             =       0.024 * 100;
SIGMA_Y             =       0.024 * 100;
SIGMA_THETA         =       0.00025 * 10;
INITIAL_RADIUS      =       0.30;

% global localization parameters
DO_GLOBAL_LOC       =       0;
GLOBAL_PARTICLES    =       10000;
GLOBAL_FACTOR       =       20.0;

% main program start

% Initialize random number generator
randn('state',sum(100*clock));

% set the number of particles appropriately
if DO_GLOBAL_LOC,
  NUM_P = GLOBAL_PARTICLES;
else
  NUM_P = NUM_PARTICLES;
end
```

```
% Initialize particles
path = zeros(size(obs,2), 4);
drpath = zeros(size(obs,2), 3);
newu = zeros(NUM_P, 3);
tempp = zeros(NUM_P, 4);
dobs = zeros(NUM_P, 2);
weights = ones(NUM_P, 1);

particles=zeros(NUM_P,4);
if DO_GLOBAL_LOC,
   % initialize particles uniformly over configuration space
   particles(:,1) = rand(NUM_P, 1) .* 500;
   particles(:,2) = rand(NUM_P, 1) .* 500;
   particles(:,3) = rand(NUM_P, 1) .* (2 * pi) - pi;
   particles(:,4) = INITIAL_RADIUS + randn(NUM_P, 1) .* SIGMA_RADIUS;
   SIGMA_RANGE = SIGMA_RANGE * GLOBAL_FACTOR;
   SIGMA_BEARING = SIGMA_BEARING * GLOBAL_FACTOR;
else
   % initialize particles using gaussian
   particles(:,1) = xtrue(1,1) + randn(NUM_P, 1) .* SIGMA_X;
   particles(:,2) = xtrue(2,1) + randn(NUM_P, 1) .* SIGMA_Y;
   particles(:,3) = xtrue(3,1) + randn(NUM_P, 1) .* SIGMA_THETA;
   particles(:,4) = INITIAL_RADIUS + randn(NUM_P, 1) .* SIGMA_RADIUS;
end

% initialize bookkeeping
drx = xtrue(1,1);
dry = xtrue(2,1);
drtheta = xtrue(3,1);
obs_count = 0;
perception_count = 1;
last_resample = -1;
time = 0;
shrunk = 0;

if INTERACTIVE,
   % draw the results
   figure(1);
   plot(xtrue(1,:),xtrue(2,:),'b');
   hold on
   plot(path(1:i,1),path(1:i,2),'r');
   plot(particles(:,1),particles(:,2), 'gx');
   plot(beacons(:,1),beacons(:,2),'kx');
   legend('True Path', 'Estimated Path (PF)');
   hold off
   axis([0 500 0 500]);
```

```
   disp('Press return to continue');
   pause;
end

% loop over all of the actions and observations
for i=1:size(obs,2),
   % compute delta t
   dt = uz(3,i) - time;
   time = uz(3,i);

   % compute dead reckoning path (for display purposes only)
   tdrx = drx + dt .* INITIAL_RADIUS .* uz(1,i) .* cos(drtheta + uz(2,i));
   tdry = dry + dt .* INITIAL_RADIUS .* uz(1,i) .* sin(drtheta + uz(2,i));
   tdrtheta = drtheta + dt .* INITIAL_RADIUS .* uz(1,i) .* sin(uz(2,i)) / WHEEL_BASE;
   drx = tdrx;
   dry = tdry;
   drtheta = tdrtheta;
   drpath(i,:) = [drx dry drtheta];

   % particle filter action update
   newu(:,1) = uz(1,i) + SIGMA_VELOCITY .* randn(NUM_P,1);
   newu(:,2) = uz(2,i) + SIGMA_STEERING .* randn(NUM_P,1);
   newu(:,3) = SIGMA_RADIUS .* randn(NUM_P,1);
   tempp(:,1) = particles(:,1) + dt .* (particles(:,4) + newu(:,3)) .* newu(:,1) .*
cos(particles(:,3) + newu(:,2));
   tempp(:,2) = particles(:,2) + dt .* (particles(:,4) + newu(:,3)) .* newu(:,1) .*
sin(particles(:,3) + newu(:,2));
   tempp(:,3) = particles(:,3) + dt .* (particles(:,4) + newu(:,3)) .* newu(:,1) .* sin(newu(:,2)) /
WHEEL_BASE;
   tempp(:,4) = particles(:,4) + newu(:,3);
   particles = tempp;

   % particle filter perception update
   if(obs(3,i) ~= 0)
      obs_count = obs_count + 1;
      if rem(obs_count, PERCEPTION_SKIP) == 0,
         dobs(:,1) = (((beacons(obs(3,i),1) - particles(:,1)).^2 + (beacons(obs(3,i),2) -
particles(:,2)).^2).^0.5 - obs(1,i)) ./ SIGMA_RANGE;
         dobs(:,2) = (a_sub(a_sub(atan2(beacons(obs(3,i),2) - particles(:,2), beacons(obs(3,i),1)
- particles(:,1)), particles(:,3)), obs(2,i))) ./ SIGMA_BEARING;
         weights = weights .* exp(-0.5 * (dobs(:,1).^2 + dobs(:,2).^2));
         if max(weights > 0),
            weights = weights ./ max(weights);
         else
            weights(:,1) = 1.0;
         end
```

```
      perception_count = perception_count + 1;
   end
end
  % keep track of the mean particle
path(i,:) = mean(particles, 1);

% If global localization is complete, turn the number of particles back down
% WARNING : THIS IS A COMPLETE HACK
% See paper by Dieter Fox on Adaptive Particle Filters for the real way to do this
if DO_GLOBAL_LOC & std(particles(:,1)) < 10.0 & ~shrunk,
   NUM_P = NUM_PARTICLES;
   particles = particles(1:NUM_P,:);
   newu = newu(1:NUM_P,:);
   tempp = tempp(1:NUM_P,:);
   dobs = dobs(1:NUM_P,:);
   weights = weights(1:NUM_P,:);
   shrunk = 1;
end

% particle filter resampling - low variance algorithm
if rem(perception_count, RESAMPLE_SKIP) == 0 & last_resample ~= perception_count,
   last_resample = perception_count;
   fprintf(1, 'resample at time %d\n', i);
   weight_sum = sum(weights);
   cumulative_sum = cumsum(weights, 1);
   step_size = weight_sum / NUM_P;
   position = rand * weight_sum;
   which_particle = 1;
   for j = 1:NUM_P,
      position = position + step_size;
     if position > weight_sum
        position = position - weight_sum;
        which_particle = 1;
      end
      while position > cumulative_sum(which_particle)
        which_particle = which_particle + 1;
      end
      tempp(j,:) = particles(which_particle,:);
   end
   particles = tempp;
   weights = ones(NUM_P, 1);

   if INTERACTIVE,
      % draw the results
      figure(1);
      plot(xtrue(1,:),xtrue(2,:),'b');
```

```
        hold on
        plot(drpath(1:i,1),drpath(1:i,2),'k--');
        plot(path(1:i,1),path(1:i,2),'r');
        plot(particles(:,1),particles(:,2), 'gx');
        plot(beacons(:,1),beacons(:,2),'kx');
        legend('True Path', 'Dead Reckoning', 'Estimated Path (PF)');
        hold off
        axis([0 500 0 500]);
        disp('Press return to continue');
        pause;
      end
   end
end

% draw the results
figure(1)
plot(xtrue(1,:),xtrue(2,:),'b');
hold on
plot(drpath(1:i,1),drpath(1:i,2),'k--');
plot(path(1:i,1),path(1:i,2),'r');
plot(particles(:,1),particles(:,2), 'g.');
plot(beacons(:,1),beacons(:,2),'k*');
legend('True Path', 'Dead Reckoning', 'Estimated Path (PF)');
hold off
axis([0 500 0 500]);
xlabel('X direction (m)')
ylabel('Y direction (m)')

err = abs(xtrue' - path);
figure(2)
plot((err(:,1).^2+err(:,2).^2).^0.5)
grid on
xlabel('Time (s)')
ylabel('RMS Position Error (m)')

figure(3)
grid on
plot(err(:,3))
grid on
xlabel('Time (s)')
ylabel('Orientation Error (radians)')
```

set_up

```
% script file to set up a vehicle run for subsequent
% filtering and localisation algorithms
```

```
globals;        % define global variables
ginit;          % set global variables

% first step is to input beacons
disp('Input Beacon Locations via mouse. Press return to end');
beacons=get_beacons;
[n_beacons,temp]=size(beacons);
buf=sprintf('%d Beacons read\n',n_beacons);
disp(buf);

% Next, input path spline points and compute path
disp('Input path spline points via mouse. Press return to end');
path=get_path(beacons);
[temp,n_path]=size(path);
buf=sprintf('%d Path points of total Length %f meters read\n',n_path,n_path*LINC);
disp(buf);
disp('Press Return to continue');
pause;

% do controller to build true path and control vectors
% initialise
time=0:DT:TEND;
[temp,tsize]=size(time);

xtrue=zeros(4,tsize);  % needed for reseting between runs
utrue=zeros(3,tsize);  % and again
xtrue(1,1)=path(1,1);  % initial x
xtrue(2,1)=path(2,1);  % initial y
xtrue(3,1)=atan2(path(2,2)-path(2,1),path(1,2)-path(1,1)); % initial phi
xtrue(4)=0;                                    % time=0;
utrue(1)=VVEL;                                 % velocity set at 2 m/s
utrue(2)=0;                                    % initial steer is zero
utrue(3)=0;                                    % time=0

% loop control
buf=sprintf('Beginning Simulation, please wait\n');
disp(buf)

index = 1;
for i=1:(tsize-1)
  % find error
  [perr,oerr,index,d]=get_err(xtrue(:,i),path,index);

  % compute next state
  [xtrue(:,i+1),utrue(:,i+1)]=get_control(xtrue(:,i),utrue(:,i),perr,oerr,DT);
```

```
  if d >10  % test for end of path
    break;
  end
end

% shorten vectors to end length
xtrue=xtrue(:,1:i);
utrue=utrue(:,1:i);
utrue(1,:)=utrue(1,:)/WHEEL_RADIUS; % make speed into rads/s

% add noise if required
uz(1,:)=utrue(1,:)+GSIGMA_WHEEL*randn(size(utrue(1,:)));
uz(2,:)=utrue(2,:)+GSIGMA_STEER*randn(size(utrue(2,:)));
uz(3,:)=utrue(3,:);

buf=sprintf('simulation terminating at time %f\n',i*DT);
disp(buf);
figure(PLAN_FIG)
hold on
plot(xtrue(1,:),xtrue(2,:),'g')
hold off
```

track_innov

```
function [binnov, binnovsig]=proc_innov(innov,innovar,obs,x,u)
%
% function [binnov, binnovsig]=proc_innov(innov,innovar,obs,bnum)
%
% a function to process innovation results, extracting (optionally)
% innovations for specific beacons and standarad deviations

[temp,nsamps]=size(obs);
binnov=zeros(3,nsamps);
binnovsig=zeros(2,nsamps);
j=0;

        for i=1:nsamps
        if obs(3,i) ~= 0
        j=j+1;
      ivec=innov(1:2,i); % innovation
      svec=innovar(1:2,1:2,i); % innovation variance
      angle=x(3,i)+u(2,i); % vehicle heading angle=orientation+steering
      T=[cos(angle) sin(angle); -sin(angle) cos(angle)]; % transform
      tvec=T*ivec;
      tsvec=T*svec*T';
        binnov(3,j)=obs(4,i);
```

```
        binnovsig(1,j)=sqrt(tsvec(1,1));
      binnovsig(2,j)=sqrt(tsvec(2,2));
      binnov(1:2,j)=tvec;
        end
        end
binnov=binnov(:,1:j);
binnovsig=binnovsig(:,1:j);
```

update

```
function [xest, Pest, innov, S]=update(xpred,Ppred,obs,beacons)
%
% function [xest, Pest, innov, S]=update(xpred,Ppred,obs,beacons)
%
% HDW 3/1/95 Modified 31/05/00
% function to update vehicle location

globals;

[temp,N_OBS]=size(obs);
if temp ~= 4
 error('observation dimension of 4 expected')
end

[XSIZE,temp]=size(xpred);
if temp ~= 1
 error('xpred expected to be a column vector')
end

[s1,s2]=size(Ppred);
if((s1 ~= XSIZE)|(s2 ~=XSIZE))
 error('Ppred not of size XSIZE')
end

sigma_range=SIGMA_RANGE*SIGMA_RANGE;
sigma_bearing=SIGMA_BEARING*SIGMA_BEARING;

% first put observation in cartesian vehicle-centred coords
zv=[R_OFFSET+(obs(1)*cos(obs(2)));
   obs(1)*sin(obs(2))];
T=[cos(obs(2)) -sin(obs(2));
  sin(obs(2)) cos(obs(2))];
sigma_o=[sigma_range 0;
      0 sigma_bearing*(obs(1)^2)];
sigma_z=T*sigma_o*T';
```

```
% then in base coordinates
zb=[xpred(1)+(zv(1)*cos(xpred(3)))-(zv(2)*sin(xpred(3)));
   xpred(1)+(zv(1)*sin(xpred(3)))+(zv(2)*cos(xpred(3)))];
Tx=[1 0 -((zv(1)*sin(xpred(3)))+(zv(2)*cos(xpred(3)))) 0;
   1 0 -((zv(1)*cos(xpred(3)))+(zv(2)*sin(xpred(3)))) 0];
Tz=[cos(xpred(3)) -sin(xpred(3));
   sin(xpred(3))  cos(xpred(3))];
sigma_b=Tx*Ppred*Tx' + Tz*sigma_z*Tz';

% now try and match observation to beacon map
% index=match(zb,sigma_b,beacons);
index=obs(3);
if (index)
  dx=beacons(index,1)-xpred(1);
  dy=beacons(index,2)-xpred(2);
  T=[ cos(xpred(3)) sin(xpred(3));
    -sin(xpred(3)) cos(xpred(3))];
  H=[-cos(xpred(3)) -sin(xpred(3)) (-(dx*sin(xpred(3)))+(dy*cos(xpred(3)))) 0;
     sin(xpred(3)) -cos(xpred(3)) (-(dx*cos(xpred(3)))-(dy*sin(xpred(3)))) 0];
  zpred=T*[dx;dy];
  S=H*Ppred*H' + sigma_z;
  W=Ppred*H'* inv(S);
  Pest=Ppred-W*S*W';
  innov=[zv(1)-zpred(1); zv(2)-zpred(2)];
  xest=xpred+W*innov;
else
  xest=xpred;
  Pest=Ppred;
  S=zeros(2,2);
  innov=zeros(2,1);
end
```