ACKNOWLEDGEMENTS

First of all, I would like to dedicate this work with tremendous love to my parents for their support, encouragement, sacrifices and unconditional love throughout my entire life.

I thank the almighty for the divine grace I received throughout the research work, without which I could not have completed this work successfully.

I express my gratitude to **Prof. R. Sethuraman**, Vice Chancellor, and SASTRA University who provided all facilities and necessary encouragement during the course of study. I extend my sincere thanks to **Dr. G. Bhalachandran**, Registrar, SASTRA University for providing the opportunity to pursue this project.

I dedicate my whole hearted thanks to **Dr. B. Viswanathan**, Dean, SEEE, SASTRA University and **Prof. Dr. M. Narayanan**, Dean, Student affairs, SEEE, SASTRA University, for their moral support. I also thank **Dr. K. Thenmozhi**, Associate Dean (ECE) who motivated me during the project work.

I owe a debt of deepest gratitude to my mentors **Prof. N. S. Manigandan** and **Prof. Dr. K. Ramkumar** for their continuous support and guidance throughout the process during the pursuit of my project work. Their deep insight in the field and invaluable suggestions helped me in making progress through my project work.

I would like to thank **Prof. Dr. B. Ravindran**, Associate Professor, Department of Computer Science and Engineering, Indian Institute of Technology Madras, Chennai for giving me the opportunity to work under him.

I wish to thank all my teaching and non-teaching staff members of the Department of Electronics and Communication Engineering of SEEE for their support and cooperation. I also extend my gratitude to all teachers who taught me since my childhood days.

I would also like to thank **Mr. Prasana Parthasarathi**, **Mr. Sriram Ramprasad**, **Mr. A. Gowthaman**, fellow students and interns at IITM, Chennai for helping me in times of need and understanding.

I wish to thank **Mr. L. S. Aswath** and **Mr. Raghul Shanmuganathan** for helping me with the documentation of the report.

Last but not the least, I would like to mention the team ”Kekhron Mekhron” and all my friends who were with me in difficult times and helped me complete this project successfully.

Contents

ACKNOWLEDGEMENTS **i**

List of Figures **vi**

ABSTRACT **vii**

1 BACKGROUND AND MOTIVATION **1**

1.1 Introduction 1

1.2 Background and Motivation 3

1.3 Mathematical Prerequisites 5

2 MAPPING OF A STOCHASTIC ENVIRONMENT **9**

2.1 Introduction 9

2.2 Robotic Mapping Problem 11

2.3 Related Work 14

2.4 Occupancy Grid Mapping 17

2.5 Occupancy grid mapping algorithm 20

2.6 Experimental Results 23

3 HIGH DIMENSIONAL PATH PLANNING AND CONTROL **25**

3.1 Introduction 26

3.2 Related Work 27

3.3 Graph search as a path planner 28

3.4 Selection of Heuristic: 29
3.5 The Rapidly Exploring Random Tree Search 31
3.6 Proposed Solution: RRT+ 32
3.7 Path Smoothing using Bezier Curves. 37
3.8 Control of Mobile Robot. 40
3.9 Experimental Results. 41

4 Real-time Implementation of Particle Filter with Adaptive Sampling. 49
4.1 Introduction 49
4.2 Related Work 50
4.3 Particle Filter as a State Estimator. 52
4.4 Mobile Robot Localization 53
4.5 The Particle Filter Algorithm 55
4.6 Adaptive Sampling for Particle Filter 61
4.7 Experimental Results 63

5 CONSTRUCTION OF RELIABILITY MAP. 66
5.1 Introduction 66
5.2 Related Work 67
5.3 Reliability Mapping 68

LIST OF FIGURES

2.1 Impact of odometry error 12

2.2 The process of estimation of grid measure from sensor data 18

2.3 A procedural representation of occupancy grip mapping 19

2.4 Various stages of occupancy grid mapping. 22

2.5 Example of a map built using occupancy grid mapping in Stage simulator. 22

2.6 Pioneer 3DX driving autonomously to map the environment . . . 23

2.7 Probabilistic map obtained using occupancy grid algorithm. The gray level of each pixel refers to the likelihood of presence of an obstacle. White refers to free region and black refers to solid obstacle. 24

3.1 Schematic representation of construction of nodes in basic RRT. . 32

3.2 RRT build with adaptive branch lengths. 34

3.3 Construction of parametric Bezier curves. 38

3.4 Raw coordinates obtained by RRT+ planning. 39

3.5 Smoothed trajectory of raw RRT+ plan using Bezier curves. . . . 40

3.6 Comparison of RRT and RRT+ in synthetic map. 45

3.7 Costs are plotted against iterations showing the rate of convergence of both RRT and RRT+. 46

3.8 Running Time Ratio of RRT over RRT+ is plotted against iterations. Constant b is taken as 2 for RRT+. 47
3.9 Costs of path in fifty different environments for 2500 iterations with dynamic biasing ratio scheme. 47
3.10 RRT+ implemented for planning a path on probabilistic map of Computer Science department of IIT Madras(red line indicates the route of the planned path). 48

4.1 Flowchart showing the steps involved in particle filtering. 59
4.2 Working of Particle Filter. Adapted from [10] 60
4.3 Normal probability plot of error function. 62
4.4 Modelled probability density function of sensor variance. 63
4.5 Comparison between particle filter with arbitrary sampling and adaptive sampling. 64
4.6 Various stages of particle filter convergence in simulated environment. 65

5.1 Reliability map of an indoor environment showing the belief distribution between the sensors at different locations of the environment. 70

ABSTRACT

Mobile Robot Localisation denotes the robot′s ability to establish its own position and orientation within the frame of reference. The past decade of research has developed various algorithms and mathematical structures to tackle this problem in Robotics. The high computational cost of these models makes the real-time implementation a challenging task thereby leaving the domain entirely in research phase. The conventional format of Particle Filter operating on large state spaces faces poor convergence rates at real-time. Our attempt is to overcome this computational overhead by devising external robust sensor models to support the high dimensions of state space. Autonomous navigation has to be completely established to facilitate path planning, localisation, dynamic obstacle avoidance and mapping of the environment. An unknown environment is completely mapped, displaying the probabilistic distribution of presence of an obstacle over the region using occupancy grid method for mapping. Particles are sampled from any multi-modal distribution and spread uniformly over this map to implement Particle Filter. As stated, a modified version of Particle filter is formulated to localise at real-time.

With the help of dependable localisation, path planning is been done using Randomly exploring Rapid Trees (RRTs). A novel variation is brought in this algorithm that scales itself depending on the surrounding of Mobile Robot in the environment. A reduction in time complexity is witnessed by this adaptive modification resulting in better convergence rate as compared to other graph search

algorithms at higher dimensions. The smoothing of the discrete path generated is done by Bezier parametric curve.

A Kinect sensor tracks the dynamic obstacles in the environment by defining skeletons for features and doing background subtraction. A basic regression is done to predict the trajectory of moving obstacles. On collision with the planned trajectory a quick dynamic re-planning is done to establish a path between the broken branches of the tree and evade the obstacle. The control part involves traversing the path using simple inverse kinematics strategies.

Localisation is carried out on different terrains and synthetic data sets to develop a policy that could effectively adjust the belief of sensors depending on the type of environment encountered. With this an attempt is being made to build a reliability map of the world by exploiting the rewards obtained on sufficient exploration. A reliability map would provide information on adequate sensors to be activated at any instant to maintain minimum localisation error.

Mobile Robot platform: Pioneer 3DX, Corobot CL2

Hardware Abstraction Layer: Robot Operating System (ROS).

CHAPTER 1

BACKGROUND AND MOTIVATION

1.1 Introduction

Robotics is the science of perceiving and manipulating the physical world through computer- controlled mechanical devices. Examples of successful robotic systems include mobile platforms for planetary exploration, robotics arms in assembly lines, cars that travel autonomously on highways, actuated arms that assist surgeons. Robotics systems have in common that they are situated in the physical world, perceive their environments through sensors, and manipulate their environment through things that move. While much of robotics is still in its infancy, the idea of intelligent manipulating devices has an enormous potential to change society. The most striking characteristic of the new robot systems is that they operate in increasingly unstructured environments, environments that are inherently unpredictable. These types of environments are subjected to a most crucial and daunting factor: *Uncertainity*. Uncertainty arises if the robot lacks critical information for carrying out its task. It arises from five different factors:

- **Environment:** Physical worlds are inherently unpredictable. While the degree of uncertainty in well−structured environments such assembly lines is small, environments such as highways and private homes are highly dynamic and unpredictable.

- **Sensors:** Sensors are inherently limited in what they can perceive. Limitations arise from two primary factors. First, range and resolution of a sensor is subject to physical laws. For example, Cameras can$'$t see through walls, and even within the perceptual range the spatial resolution of camera images is limited. Second, sensors are subject to noise, which perturbs sensor measurements in unpredictable ways and hence limits the information that can be extracted from sensor measurements.

- **Robots:** Robot actuation involves motors that are, at least to some extent, unpredictable, due effects like control noise and wear and tear. Some actuators, such as heavy$-$duty industrial robot arms, are quite accurate. Others, like low-cost mobile robots, can be extremely inaccurate.

- **Models:** Models are inherently inaccurate. Models are abstractions of the real world. As such, they only partially model the underlying physical processes of the robot and its environment. Model errors are a source of uncertainty that has largely been ignored in robotics, despite the fact that most robotic models used in state of the art robotics systems are rather crude.

- **Computation:** Robots are real$-$time systems, which limits the amount of computation that can be carried out. Many state of the art algorithms (such as most of the algorithms described in this book) are approximate, achieving timely response through sacrificing accuracy. All of these factors give rise to uncertainty. Traditionally, such uncertainty has mostly been ignored in robotics. However, as robots are moving away from factory floors into increasingly unstructured environments, the ability to cope with uncertainty is critical for building successful robots.

1.2 Background and Motivation

Two key problems in mobile robotics are global position estimation and local position tracking. We define global position estimation as the ability to determine the robots position in an a priori or previously learned map, given no other information than that the robot is somewhere on the map. Once a robot has been localized in the map, local tracking is the problem of keeping track of that position over time. Both these capabilities are necessary to enable a robot to execute useful tasks, such as office delivery or providing tours to museum visitors. By knowing its global position, the robot can make use of the existing maps, which allows it to plan and navigate reliably in complex environments. Accurate local tracking on the other hand, is useful for efficient navigation and local manipulation tasks. Both these sub-problems are of fundamental importance to building truly autonomous robots. We believe that probabilistic approaches are among the most promising candidates to providing a comprehensive and real-time solution to the robot localization problem, but current methods still face considerable hurdles. Kalman-filter based techniques have proven to be robust and accurate for keeping track of the robots position. However, a Kalman filter cannot represent ambiguities and lacks the ability to globally re−localize the robot in the case of localization failures. The grid based Markov localization approach which can represent arbitrarily complex probability densities at fine resolutions. However, the computational burden and memory requirements of this approach are considerable. In addition, the grid-size and thereby also the precision at which it can represent the state has to be fixed beforehand.

Tracking or local techniques aim at compensating odometric errors occurring during robot navigation. They require, however, that the initial location of the robot is (approximately) known and they typically cannot recover if they lose track of the robot′s position (within certain bounds). Another family of approaches is called global techniques. These are designed to estimate the position of the robot even under global uncertainty. Techniques of this type solve the so-called wake-up robot problem, in that they can localize a robot without any prior knowledge about its position. They furthermore can handle the kidnapped robot problem, in which a robot is carried to an arbitrary location during it′s operation [14]. Global localization techniques are more powerful than local ones. They typically can cope with situations in which the robot is likely to experience serious positioning errors. In this paper we present a metric variant of Markov localization, a technique to globally estimate the position of a robot in its environment. Markov localization uses a probabilistic framework to maintain a position probability density over the whole set of possible robot poses. Such a density can have arbitrary forms representing various kinds of information about the robot′s position.

For example, the robot can start with a uniform distribution representing that it is completely uncertain about its position. It furthermore can contain multiple modes in the case of ambiguous situations. In the usual case, in which the robot is highly certain about its position, it consists of a unimodal distribution centered on the true position of the robot. Based on the probabilistic nature of the approach and the representation, Markov localization [15] can globally estimate the position of the robot, it can deal with ambiguous situations, and it can re-localize the robot in the case of localization failures. These properties are basic precondi-

tions for truly autonomous robots designed to operate over long periods of time. This approach has several advantages over previous ones, which predominately used Gaussians or coarse-grained, topological representations for approximating a robot′s belief. First, it provides more accurate position estimates, which are required in many mobile robot tasks (e.g., tasks involving mobile manipulation). Second, it can incorporate raw sensory input such as a single beam of an ultrasound sensor. Most previous approaches to Markov localization, in contrast, screen sensor data for the presence or absence of landmarks, and they are prone to fail if the environment does not align well with the underlying assumptions (e.g., if it does not contain any of the required landmarks). Most importantly, however, previous Markov localization techniques assumed that the environment is static. Therefore, they typically fail in highly dynamic environments, such as public places where crowds of people may cover the robot′s sensors for extended periods of time. To deal with such situations, our method applies altering technique that, in essence, updates the position probability density using only those measurements which are with high likelihood produced by known objects contained in the map. As a result, it permits accurate localization even in densely crowded, non-static environments.

1.3 Mathematical Prerequisites

- **Bayes Theorem:** Bayes′ theorem is stated mathematically as the following equation:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}, \tag{1.1}$$

  where:

A *and* B = are events.

$P(A)$ *and* $P(B)$ = are the probabilities of A and B independent of each other.

$P(A|B)$ = a conditional probability, is the probability of A given that B is true.

$P(B|A)$ = is the probability of B given that A is true.

- **Importance Sampling:** In statistics, importance sampling is a general technique for estimating properties of a particular distribution, while only having samples generated from a different distribution from the distribution of interest.

  Let $X : \Omega \to \mathbb{R}$ be a random variable in some probability space ($\Omega, \mathscr{F}$,P). We wish to estimate the expected value of X under P. If we have random samples $x_1, \ldots, x_n$, generated according to P, then an empirical estimate of E[X;P] is

$$\widehat{\mathbf{E}}_n[X;P] = \frac{1}{n}\sum_{i=1}^{n} x_i. \tag{1.2}$$

- **Thompson Sampling:** Consider a set of contexts $\mathscr{X}$, a set of actions $\mathscr{A}$, and rewards in $\mathbb{R}$. In each round, the player obtains a context x $\in \mathscr{X}$, plays an action a $\in \mathscr{A}$ and receives a reward r $\in \mathbb{R}$ following a distribution that depends on the context and the issued action. The aim of the player is to play actions such as to maximize the cumulative rewards.

  The elements of Thompson sampling are as follows:

  - a set Θ of parameters θ;
  - a prior distribution $P(\theta)$ on these parameters;
  - past observations triplets $\mathscr{D} = \{(x;a;r)\}$;

- a likelihood function $P(r|\theta, a, x)$;
- a posterior distribution $P(\theta|\mathscr{D}) \propto P(\mathscr{D}|\theta)P(\theta)$, where $P(\mathscr{D}|\theta)$ is the likelihood function.

Thompson sampling consists in playing the action $a^* \in \mathscr{A}$ according to the probability that it maximizes the expected reward, i.e.

$$\int \mathbb{I}[\mathbb{E}(r|a, x, \theta) = \max_{a'} \mathbb{E}(r|a', x, \theta)]P(\theta|\mathscr{D})\, d\theta, \tag{1.3}$$

where $\mathbb{I}$ is the indicator function.

- **Regression Analysis:** In statistics, regression analysis is a statistical process for estimating the relationships among variables. It includes many techniques for modeling and analyzing several variables, when the focus is on the relationship between a dependent variable and one or more independent variables. More specifically, regression analysis helps one understand how the typical value of the dependent variable (or criterion variable) changes when any one of the independent variables is varied, while the other independent variables are held fixed. Most commonly, regression analysis estimates the conditional expectation of the dependent variable given the independent variables that is, the average value of the dependent variable when the independent variables are fixed. In all cases, the estimation target is a function of the independent variables called the regression function. In regression analysis, it is also of interest to characterize the variation of the dependent variable around the regression function which can be described by a probability distribution.

- **Weighted Sampling:** A probability sample is a sample in which every unit in the population has a chance (greater than zero) of being selected in the sample, and this probability can be accurately determined. The combination of these traits makes it possible to produce unbiased estimates of population totals, by weighting sampled units according to their probability of selection.

  These various ways of probability sampling have two things in common:

  - Every element has a known nonzero probability of being sampled
  - Involves random selection at some point

CHAPTER 2

MAPPING OF A STOCHASTIC ENVIRONMENT

Robotic Environment mapping deals with the problem of obtaining spatial models of physical environment using mobile robots. A highly robust method of obtaining environment map is necessary for the building of truly autonomous robots.

2.1 Introduction

The past two decades have witnessed significant improvements in the area of robotic mapping. Despite that it still poses great challenges as there is no generic solution that can perfectly map all kinds of environment. Though many mapping techniques can efficiently map limited size, structured and static environments they perform poorly in large−scale, unstructured and dynamic environments.

Our attempts focus in mapping indoor environments probabilistically. Mapping algorithms can either be incremental and can be run in real time or require passing the data through the algorithm multiple times. In a similar fashion some algorithms demand exact pose information of the mobile robot while others can be implemented using odometry information. Some algorithms are equipped to handle correspondence problems between data recorded at different points in time, whereas others require features to carry signatures that makes them uniquely iden-

tifiable.

The mapping of an environment serves as the basic step for many robotic tasks like localization, planning and navigation. The data obtained from the robot fitted with sensors are fused to develop the structure of environment numerically over a large array. Sensors commonly used in this task include cameras, range finders using sonar, laser, and infrared technology, radar, tactile sensors, compasses, and GPS. There always exists a trade off between the number of sensors used, the amount of data obtained and the computational strength to randomly access data over huge data structures.

As the robotic mapping is characterized by uncertainty and sensor noise the probabilistic techniques are preferred over point estimates. Probabilistic algorithms approach the problem by explicitly modeling different sources of noise and their effects on the measurements. The usage of mathematical methodology to solve the situation can be justified by understanding the complexity of mapping as a perceptual interference problem.The basic principle underlying virtually every single successful mapping algorithm is Bayes rule shown in equation 2.1:

$$p(x|d) \;=\; \eta \; p(d|x) \; p(x) \tag{2.1}$$

Bayes rule is a typical example of probabilistic inference. A knowledge about quantity **x** (robot′s position) given the information about **d** (sensor measurement) can be obtained by multiplying the two terms: p(d|x) and p(x), where the term p(d|x) specifies the probability of observing the measurement **d** under the hypothesis **x**.

2.2 Robotic Mapping Problem

A key challenge in robotic mapping arises from the nature of the measurement noise. The problem becomes a relatively easier task if the noise in the different measurements are statistically independent. However in many cases of environment mapping the dependent nature of noise make the error evolve over time and affect the way the future sensor measurements are interpreted. This could be intuitively understood by looking at Figure 2.1 in which a small rotational error at one end of the corridor can lead to meters of error on the other side. Accommodating such systematic errors is key to building maps successfully, and it is also a key complicating factor in robotic mapping.

The second daunting aspect that complicates the robotic mapping is the dimensionality of the problem. The dimensionality of the problem can be understood as the number of features one needs to consider to describe an environment. It is always possible to describe the topology with major entities like corridors, intersections, doors and rooms. But a two dimensional floor plan which is an equally common representation of robotic maps, often requires thousands of numbers. But a detailed three dimensional visual map of a building may easily require millions of numbers. From a statistical point of view, each such number is a dimension of the underlying estimation problem. Thus, the mapping problem can be extremely high dimensional.

The third and probably the most difficult problem of all is the data association task or correspondence problem. The data association is a very crucial task in

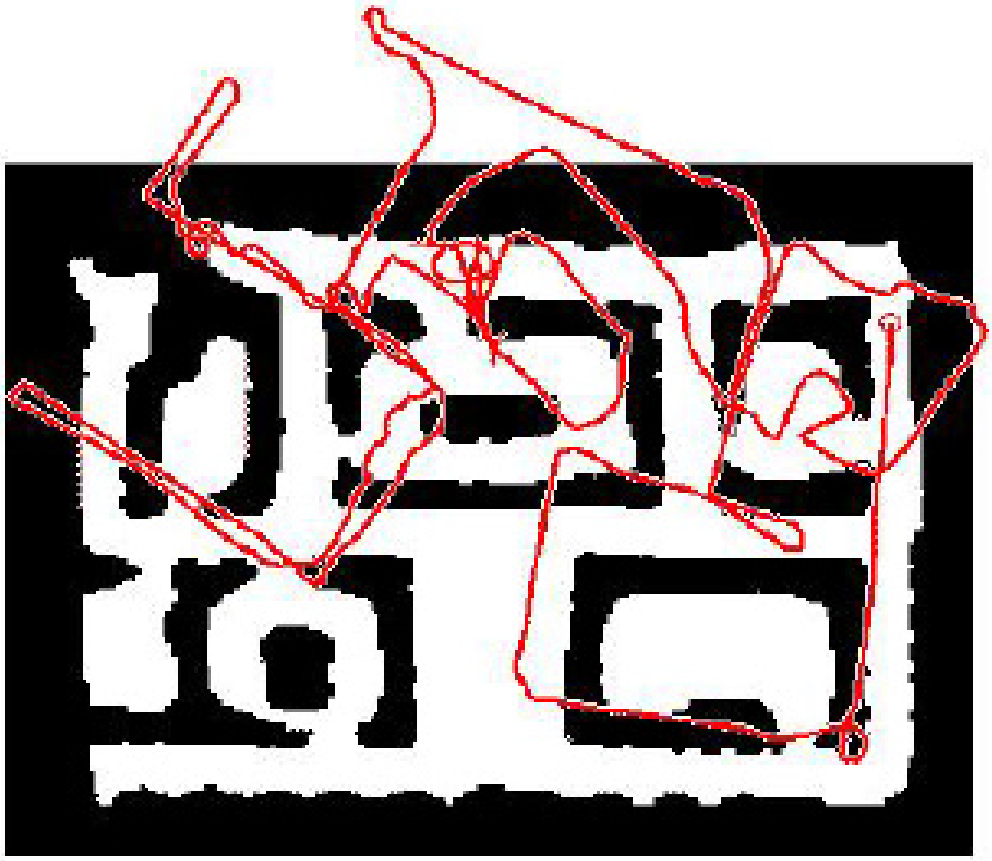

Figure 2.1 Impact of odometry error

mobile robot localization in which the sensor measurements taken at different point of time refer to the same physical entity in the real world. The loop closure in case of mobile robot navigation cannot be addressed unless the data association algorithm is formulated robustly. The accumulation of error over time goes high as the mobile robot closes the circle and hence the association problem becomes computationally expensive since we need to search for a perfect hypotheses in a space growing exponentially in time.

Fourth, environments change over time. Some changes may be relatively slow, such as the change of appearance of a tree across different seasons, or the structural changes that most office buildings are subjected to over time. Others are faster, such as the change of door status or the location of furniture items, such as chairs. Even faster may be the change of location of other agents in the environment, such as cars or people. The dynamism of robot environments creates a big challenge, since it adds yet another way in which seemingly inconsistent sensor measurements can be explained. To see, imagine a robot facing a closed door that previously was modeled as open. Such an observation may be explained by two hypotheses, namely that the door status changed, or that the robot is not where it believes to be. Unfortunately, there are almost no mapping algorithms that can learn meaningful maps of dynamic environments. Instead, the predominant paradigm relies on a static world assumption, in which the robot is the only time−variant quantity (and everything else that moves is just noise). Consequently, most techniques are only applied in relatively short time windows, during which the respective environments are static.

Fifth, the robot must choose their way while mapping. The task of generating robot motion in the pursuit of building a map is commonly referred to as robotic exploration. While optimal robot motion is relatively well−understood in fully modeled environments, exploring robots have to cope with partial and incomplete models. Hence, any viable exploration strategy has to be able to accommodate contingencies and surprises that might arise during map acquisition. For this reason, exploration is a challenging planning problem, which is often solved sub−optimally via simple heuristics. When choosing where to move, various quantities have to be traded off: the expected gain in map information, the time and energy it takes to gain this information, the possible loss of pose information along the way, and so on. Furthermore, the underlying map estimation technique must be able to generate maps in real−time, which is an important restriction that rules out many existing approaches.

2.3 Related Work

The history of mapping algorithms is broadly classified into the two types: topological and metric approaches. The geometric properties of the map are the primary concern of metric maps whereas connectivity to different locations are represented by topological maps. Elvis and Moravec′s algorithm named occupancy grid mapping [2] has left a mark in robotic mapping and it is widely used. Geometric structures like polyhedra is used to describe the environment in case of metric approaches as proposed by Chatila and Laumond [7]. Other instances of topological approaches include works done by Mataric [29] and Cuipers [22]. The most significant places in the environment are connected via arcs by the topo-

logical maps. The labels of these arcs contains details on how to manoeuvre from one location to another. In spite of these variations the distinction between metric and topological approaches is not very lucid. This basically stems from the fact that the geometric information is the basis for the working for topological approach. In general metric maps have better resolution than topological ones. This finer distinction comes at the expense of computational power and hence can be helpful to solve difficult data association problems.

Another classification in robotic mapping algorithms are world−centric and robot−centric. World−centric maps are represented in a global coordinate space. The components of the map do not contain details about the sensor measurements that lead to their discovery. On the other hand the robot−centric maps are represented in the measurement space which describe the sensor measurements a robot would receive at different locations. As there is no $'$translation$'$ of robot measurements into world coordinates the robot−centric maps look much simpler to build. But in practice, it is often difficult to extrapolate from individual measurements to measurements at nearby, unexplored places an extrapolation that is typically straightforward in world−centric approaches. Also there arises an ambiguity in robot−centric approaches if different places of the map look similar and if an obvious geometry of an map is missing from the measurement space. Considering all these factors, the dominant approaches to date generate world−centric maps.

Since the 1990s, the field of robot mapping has been dominated by probabilistic techniques. Few seminal papers by Smith, Self, and Cheeseman [32] introduced a powerful statistical framework for simultaneously solving the mapping prob-

lem and the induced problem of localizing the robot relative to its growing map. Since then, robotic mapping has commonly been referred to as SLAM or CML, which is short for simultaneous localization and mapping [16], and concurrent mapping and localization respectively. One family of probabilistic approaches employ Kalman filters to estimate the map and the robot location [8]. The resulting maps usually describe the location of landmarks, or significant features in the environment, although recent extensions exist that represent environments by large numbers of raw range measurements. An alternative family of algorithms is based on Dempsters expectation maximization algorithm. These approaches specifically address the correspondence problem in mapping, which is the problem of determining whether sensor measurement recorded at different points in time correspond to the same physical entity in the real world. A third family of probabilistic techniques seek to identify objects in the environment, which may correspond to ceilings, walls [36], doors that might be open or closed, of furniture and other objects that move. Many of these technique have counterparts in the computer vision and photogrammetry literature a connection that is still somewhat underexploited. Robot exploration in the context of mapping has also been studied extensively. Todays approaches are usually greedy, that is, they chose control by greedily maximizing information gain, sometimes under consideration of safety constraints [19]. However, this discussion of robotic exploration is beyond the scope of this dissertation and hence will not be carried out any further.

2.4 Occupancy Grid Mapping

The occupancy grid framework develops a spatial robot perception and substantiating mechanism the uses probabilistic sensor interpretation models and random field representation schemes. A variety of study mapping, localisation and mapping schemes are supported through actions performed directly on the occupancy grid representation. The map built by occupancy grid method results in a multidimensional (mostly 2D or 3D) tessellation in which the measure of each cell represents the likelihood of presence of an obstacle when scaled to real world dimensions. By definition, occupancy field **O(x)** is discrete state stochastic process over a set of continuous spatial coordinates $\mathbf{x} = (x_1,x_2,....,x_n)$, while the occupancy grid is a lattice process, defined over a discrete spatial lattice. The state variable attached with every grid of the tessellation s(C) is a discrete random variable that describes the degree of occupancy or emptiness of that state denoted as OC or EM.

Since the data from the environment reach the robot indirectly via the sensors the recovery of the spatial world problem can be considered as an estimation problem. The various steps involved involved in the estimating the grid measures are depicted in the Figure 2.2. The basic conditional probability is used to interpret the observation data obtained from the sensing device. The probability density function **p(r|z)**, which relates the reading **r** to the true parameter space range value **z**. Using this density function in the Bayesian estimation procedure to find the occupancy grid measure. After the above manipulations we will be able to obtain the deterministic environment model, using optimal estimators like maximum likeli-

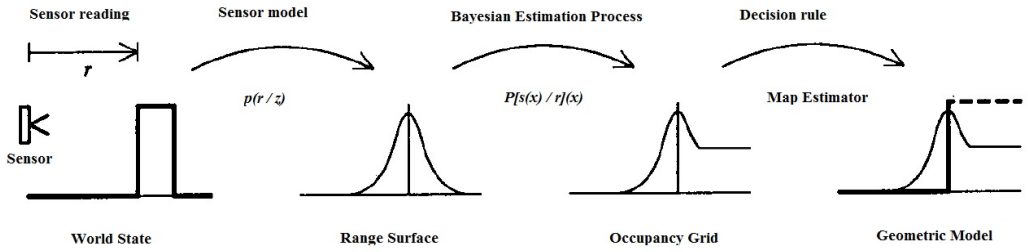

Figure 2.2 The process of estimation of grid measure from sensor data

hood estimator, majorizing estimator and maximum a posteriori estimator [1] to assign discrete states to cells, annotating them empty or occupied.

In order to facilitate the incremental setup in finding the occupancy grid probabilities, we use sequential updating formulation of Bayes′ theorem. According to that given a current estimate of state of a cell C_1, $P[s(C_1)=OC|\{r\}_t]$, based on the observations $\{r\}_t = [r_1, r_2, ..., r_t]$ and given a new observation r_{t+1} the improved estimate is given by 2.2. When this formulation is run recursively the prior estimate of the cell state $P[s(C_1)=OC|\{r\}_t]$ is obtained directly from the occupancy grid. The new cell estimate $P[s(C_1)=OC|\{r\}_{t+1}]$ is subsequently stored again in the map.

$$P(s(C_1) = OC \mid \{r\}_t) = \frac{p(\{r\}_{t+1} \mid s(C_1) = OC]P[s(C_1) = OC \mid \{r\}_t]}{\sum_{s(C_i)} P[s(C_1) \mid \{r\}_t]} \tag{2.2}$$

The occupancy grid method generates map from noisy sensor data with the assumption that robot pose is known. Occupancy grid method find approximate posterior estimates for the random variables involved. A framework representing the occupancy grid based robot mapping is shown in Figure 2.3.

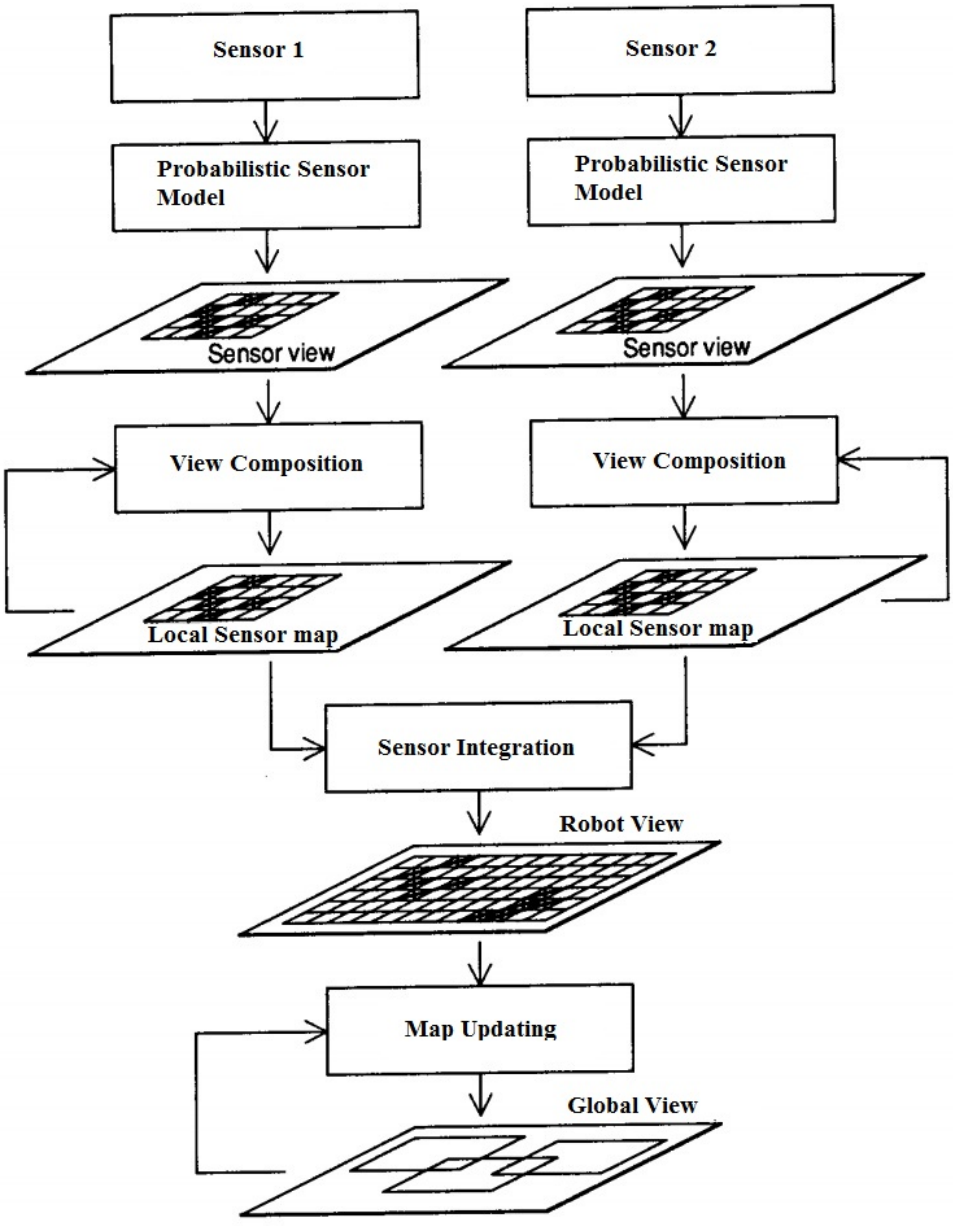

Figure 2.3 A procedural representation of occupancy grip mapping

2.5 Occupancy grid mapping algorithm

The aim of the occupancy mapping algorithm is to estimate the posterior probability over maps provided the data is given: p($m \mid z_{1:t}$,$x_{1:t}$) where **m** is the map $z_{1:t}$ is the set of the measurements from time 1 to t and $x_{1:t}$ is the set of the robot poses from time 1 to t. The controls and odometry data play no part in the occupancy grid mapping algorithm since the path is assumed known.

If we let m_i denote the grid cell with index i (often in 2d maps, two indices are used to represent the two dimensions), then the notation p(m_i) represents the probability that cell i is occupied. The computational problem with estimating the posterior p($m \mid z_{1:t}$, $x_{1:t}$) is the dimensionality of the problem: if the map contains 10,000 grid cells (a relatively small map), then the number of possible maps that can be represented by this gridding is $2^{10,000}$. Thus calculating a posterior probability for all such maps is infeasible.

The standard approach, then, is to break the problem down into smaller problems of estimating p($m_i \mid z_{1:t}$, $x_{1:t}$) for all grid cells m_i. Each of these estimation problems is then a binary problem. This breakdown is convenient but does lose some of the structure of the problem, since it does not enable modelling dependencies between neighboring cells. Instead, the posterior of a map is approximated by factoring it into p($m \mid z_{1:t}$, $x_{1:t}$) = $\prod_i p(\mathrm{m}_i \mid z_{1:t}, x_{1:t})$. Due to this factorization, a binary Bayes filter can be used to estimate the occupancy probability for each grid cell. It is common to use a log−odds representation of the probability that each grid cell is occupied. Occupancy grid algorithms represent the map **m** as a

fine−grained grid.

Algorithm: 1
Input: Sensor data and constants.

- A set K of Kinect sensor data.
- P is Pose data from the odometry model.
- S is the maximum sensor range.
- U update value of grid cell probability.
- G is the grid resolution
- Sc is the scaling constants

Output: Occupancy grid map of world WP initialized with median value

while *K and P is non empty* **do**
    Convert point cloud data into cartesian coordinates.
    Convert pose data into cartesian data.
    while *iterator* < *size of K* **do**
        Find slope of the line joining P and $K[iterator]$
        Find y intercept of the line joining P and $K[iterator]$
        if *Cells between P and* $K[iterator]$ *is non empty* **then**
            Decrease the value of each grid cell with U
            Convert the local grid cell values with Sc to get WP
        end
        Scale probabilities using $WP = \exp(WP) \ / \ (1 + \exp(WP)$
    end
end

Algorithm 1: Occupancy grid Algorithm

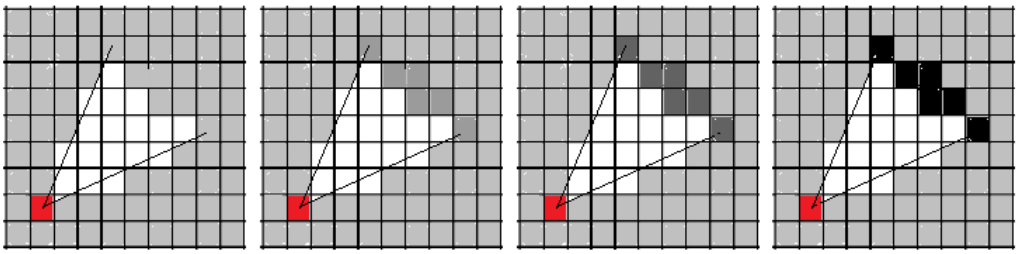

Figure 2.4 Various stages of occupancy grid mapping.

The Figure 2.4 shows how the probabilities of the grid cells build up on multiple visit to any location. Also Figure 2.5 shows the map built on a canonical environment in Stage simulator.

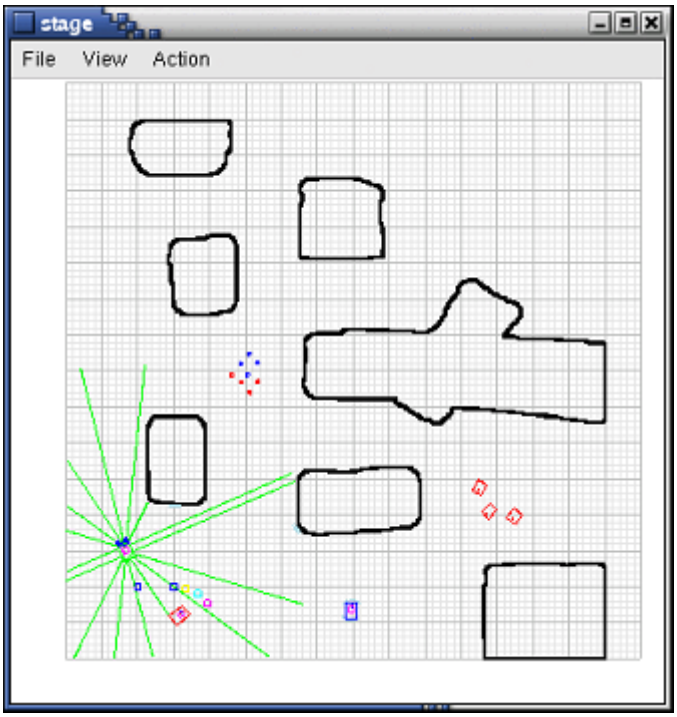

Figure 2.5 Example of a map built using occupancy grid mapping in Stage simulator.

2.6 Experimental Results

The mapping experiments are carried out in a Pioneer 3DX mobile robot platform at both the indoor and outdoor environments of IIT Madras campus. The Microsoft Kinect sensor and SONAR are used as the observation model. Optical encoders attached to the motors are used as the motion model. The odometry model is devised by employing the constants provided in optical encoder datasheet.

The Figure 2.6 shows the Pioneer robot autonomously running to map the Computer Science department of IIT Madras and Figure 2.7 shows the probabilistic map built using the occupancy grid algorithm.

Figure 2.6 Pioneer 3DX driving autonomously to map the environment

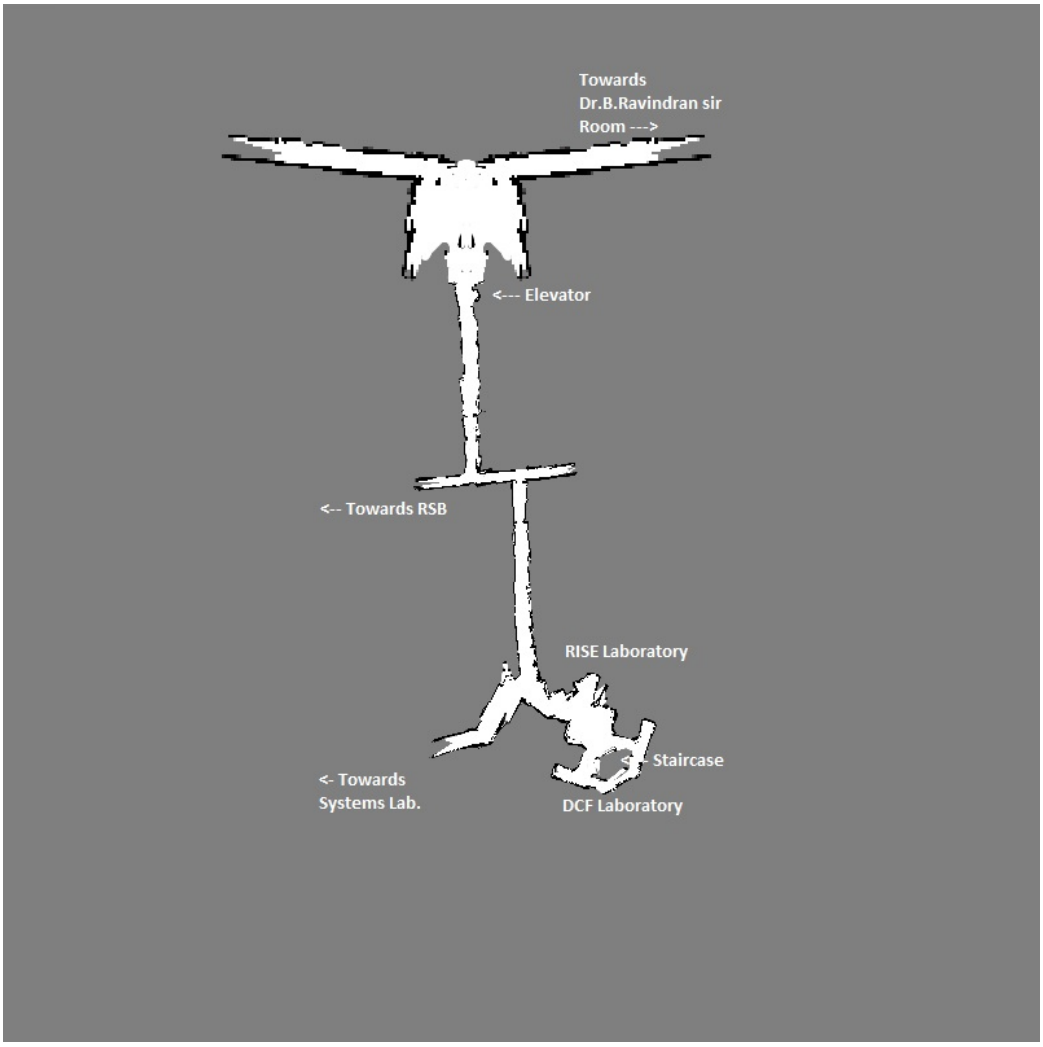

Figure 2.7 Probabilistic map obtained using occupancy grid algorithm. The gray level of each pixel refers to the likelihood of presence of an obstacle. White refers to free region and black refers to solid obstacle.

CHAPTER 3

HIGH DIMENSIONAL PATH PLANNING AND CONTROL

Every robotic task involves a mathematical function that maps states to action *i.e* a finite state machine. In order to generalize this state action pair over a trajectory the robot must exercise a phenomenon called planning. The transition matrix that makes the robot jump from one state to another is often subjected to uncertain inputs. The art of planning becomes a crucial job when ”backing out” of such undesirable states also become a part of it.

Current robotic systems maintain probabilistic models of their environment. Deterministic state transitions are a mismatch for these models in that they require a single datum to be extracted from the probabilistic models to use as an input to the state transition decision. For example, a particle filter would be processed to produce a choice for the current robot pose to use as a basis for action decisions. In some sense, this throws away the advantages of tracking multiple hypotheses in the particle filter. Also it is not possible to carry out actions leading to the different states at the same time. But it is not always that the state action pair is mutually exclusive. For example, a robot may try to handle a mug while moving to some location. In such cases the actions happen simultaneously and may lead to different resulting states at the different instances. Hence the reader must be careful to not to confuse with the standard assumptions and must take into account

the dimensionality of vectors that comprise a single state or action.

3.1 Introduction

With a broad understanding of planning in robotics the task at hand is to find a path between any two points of high dimensional non convex [31] probabilistic map. Path planning and navigation for mobile robots, in particular the case where the environment is known, is a well studied problem. However many path planning algorithms faces serious challenges when it is implemented online. The complexity of the graph search algorithm grows exponential with the size of the map. The entire map is translated as a graph consisting of only nodes and arcs.

The complexity of the planning algorithm also depends upon the heuristic used. The usage of appropriate heuristic for a search over any map cannot be generalized considering the fact that different graphs may fall under different mathematical properties. For instance, a greedy algorithm operating with euclidean distance as heuristic may perform well in simple low−dimensional convex worlds. At the same time, path planning in case of known environments allows us to explicitly choose a particular heuristic for a map thereby allowing us to hold the loss of generality with respect to that world. Dave Ferguson *et al.* [9] have worked on various nuances in choosing a heuristic that provides optimal policy for path planning. A planning strategy will always be followed by a control algorithm that works on executing the plan.

3.2 Related Work

Considering the planning problem for navigation as a graph search process, researchers have tried to come with new and better navigation technologies in the last years. The most famous ones like the Dijkstra′s algorithm [12] evaluates the moving cost from one node to any other node and sets the shortest moving cost as the connecting cost of two nodes. A little different from the Dijkstra algorithm, Best First Search algorithm has a different approach because it estimates the distance from current position to goal position, and it chooses the step that is closer to the goal position [24](LaValle, 2006). The difficulty was growing with the new path finding situations so the old path finding algorithm had to be improved to meet the new introduced requirements.

Very similar to any literature a new algorithm was proposed named the A$*$ algorithm by combining the advantages of Dijkstra and Best first search algorithm. A* uses a best first search and finds a least cost path from a given initial node to one goal node (out of one or more possible goals). As A* traverses the graph, it follows a path of the lowest expected total cost or distance, keeping a sorted priority queue of alternate path segments along the way.

Sampling based algorithms are the most favorite path planning algorithms when compared with probabilistically complete algorithms. Computationally they are less complex and and have the ability to find solutions without using explicit information about the obstacles in the configuration space. A comprehensive study and analysis on various sampling based algorithms have been done previously

[18] [13]. Rapidly exploring random tree is one of the quickest sampling based algorithm though it does not ensure optimality [33]. With that being the drawback many variations of RRT had come up out of which the development of RRT* [33] ensures asymptotic optimality, better space and time complexity.

3.3 Graph search as a path planner

The graph search algorithm can be generalized as a set of operators applied to the nodes of graph till the goal node is reached. Hence the flow of any graph search can be itemized as follows:

1. Define the start node.
2. We also define a process called *expanding* by which we apply a set of operators to each node to find the corresponding successor node.
3. Each node has a backpointer to its parent. These pointers are used to represent paths from the start node to the goal node when the search is complete.
4. A condition is examined during every iteration to find if one of the goal node is reached. If the goal node is reached the backpointers of every node is tracked to produce the solution. This process is continues until the goal node is attained.
5. In general every node follows the ideology of single parent multiple children.

With the insight gained in the generic working of any graph search algorithm we should understand and analyse the intricacies involved in programming them.

The various things which should be taken care of are:

1. Place the start node in a list called OPEN. This list will contain all the graphs nodes that are candidates for future explorations.

2. Consider two lists OPEN and CLOSED which contain a list of nodes of a graph that are already explored and unexplored respectively.

3. Pull out a node n from OPEN list and move it to the closed list.

4. Expand node n by generating all its successors. If node n has no successors, repeat step 2. If a successor of node n is not in the CLOSED list (i.e. the node has not been visited yet), it will be placed at the end of the OPEN list.

5. Check if goal is node is present in the cluster of successors. If the goal node is found then make use of the backpointers to reach the start node. If not, move to step 2. The search terminates here.

The way in which the nodes are expanded in step 2 determines the type of search. Selecting nodes for expansion in the sequence in which they are generated is called ”breadth first search” fashion whereas selection of a recently generated successor is called ”depth first search” fashion.

3.4 Selection of Heuristic:

As the search process advances the number of expanded cell increases and hence consume large running times and occupy huge computer memory. This is the consequence of a blind search. This deficiency can be eradicated by the usage of efficient heuristic. The information about the graph structure and the location

of goal node is always available. Thus the heuristic rules always help in focusing the search. Another metric called "evaluation function" is often used. The evaluation function tells about the node′s capability to generate a path towards the goal node. Using this as constraint the OPEN list is sorted and the most efficient node is picked during each iteration for exploration. In this way several criterion can be used to estimate the path cost. At times the path distance such as Euclidean metric, Manhattan metric or Mahalanobis metric is used a function whose minimization evaluates the path cost. Considering two vectors **X**=(x_1,x_2,...,x_n) and **Y**=(y_1,y_2,...,y_n), the Minkowski difference or p−norm in R_n may be defined as follows:

$$L_p = (\sum_{i=1}^{n} |x_i - y_i|^p)^{1/p} \tag{3.1}$$

Oftentimes Equation 3.4 is evaluated for values of **p** like p=1, p=2 or $p \to \infty$. The different values of **p** lead to following different metrics:

1. p=1, defines the Manhattan distance or L_1 norm:

$$L_1 = (\sum_{i=1}^{n} |x_i - y_i|) \tag{3.2}$$

2. p=2, defines the Euclidean distance or L_2 norm:

$$L_2 = \sqrt{(\sum_{i=1}^{n} |x_i - y_i|^2)} \tag{3.3}$$

3. $p \to \infty$, defines the Chebyshev distance or L_∞ norm:

$$L_k = (\sum_{i=1}^{n} |x_i - y_i|^k)^{1/k} \tag{3.4}$$

3.5 The Rapidly Exploring Random Tree Search

A rapidly exploring random tree (RRT) is an algorithm designed to efficiently search nonconvex, high−dimensional spaces by randomly building a space-filling tree. The tree is constructed incrementally from samples drawn randomly from the search space and is inherently biased to grow towards large unsearched areas of the problem.RRTs were developed by Steven M. LaValle and James J. Kuffner Jr. [25] [23]. In autonomous robot path planning, it is obvious to encounter obstacles (static and dynamic) and differential constraints (nonholonomic and kindodynamic) very frequently. RRT is an efficient tool for planning in such conditions.

RRTs can be viewed as a technique to generate open-loop trajectories for nonlinear systems with state constraints. An RRT can also be considered as a Monte-Carlo method to bias search into the largest Voronoi regions of a graph in a configuration space. Some variations can even be considered stochastic fractals. The high dimensionality of the search space can be understood as the minimum number of dimensions to be considered to define the pose of the robot and plan accordingly.

Basic RRT algorithm: The basic RRT algorithm is very easy to understand and implement. They family of RRTs perform better than any previously quoted graph search algorithm as the exploration is highly random. The tree generation

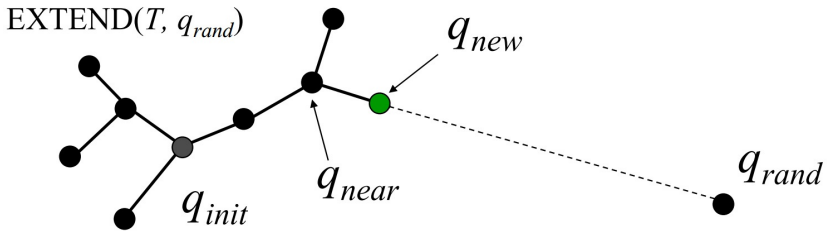

Figure 3.1 Schematic representation of construction of nodes in basic RRT.

is run over a metric space Q, for a continuous uninterupted path from q_{init} to the goal region $Q_{goal} \subset Q$ or a goal state q_{goal}. It is assumed that there is no explicit representation for the Q_{obs}, which is the fixed representation of obstacle region $Q_{obs} \subset Q$. The RRT is constructed over the vertices that are covered by Q_{free}, the complement of Q_{obs}. A state transition equation of the form $\dot{q} = f(q,u)$ is defined to express the nonholonomic constraints, where the vector u is selected from a set of inputs U. The vector can be obtained by differentiating f with respect to time. By integrating f over any fixed time interval the next state q_{next} can be found out from any given input state q and input $u \in U$. The basic RRT algorithm and a schematic representation of the tree construction is given in **Algorithm 2** and Figure 3.1:

3.6 Proposed Solution: RRT+

Though the basic format of RRT itself performs better than many graph search algorithms, the efficiency decreases as the size of the tree grows. The reason behind this phenomenon is the need to check every node of the tree during every iteration in order to find the closest node to the randomly chosen state. This search process evolves over time as the size of tree is a function of time and hence the

Input: Tree parameters.

- q_{init} is the given initial state or node.
- q_{goal} is the given goal node or state.
- Δt is a fixed time interval.

Output: RRT structure and the planned path.

$\Psi.init(q_{init})$;$q_{next} = q_{path} = \emptyset$
while q_{next} *is not equal to* q_{goal} **do**
  $q_{rand} \leftarrow RANDOM\_STATE()$;
  $q_{near} \leftarrow NEAREST\_NEIGHBOUR(q_{rand}, \Psi)$;
  $u \leftarrow SELECT\_INPUT(q_{rand}, q_{near})$;
  $q_{next} \leftarrow NEW\_STATE(q_{near}, u, \Delta t)$;
  $\Psi.ADD\_VERTEX(q_{next})$;
  $\Psi.ADD\_EDGE(q_{near}, q_{next}, u)$;
  Dictionary$[q_{parent}] = PARENT_NODE(q_{next})$
  if q_{next} *is equal to* q_{goal} **then**
    break
  end
end
while q_{path} *is not equal* q_{init} **do**
  $q_{path}[iterator] = PARENT\_NODE(q_{goal})$
  $q_{goal} = q_{path}[iterator]$
end
return Ψ, q_{path}

Algorithm 2: Basic RRT Algorithm

search is a O(n) process. In real time navigation the most optimal path consists of many continuous stretches. A continuous stretch can be defined as traversing a single long or short route in which there is no change in the control action. Such actions are generally taken in areas of the map that are very structured and uncluttered. If the vertices from the nodes of the tree are of fixed length throughout the planning process then there is a possibility that several open and obstacle-free regions of the environment may accumulate redundant nodes. Adaptively changing the size of the vertices depending on the structure of the map can solve this

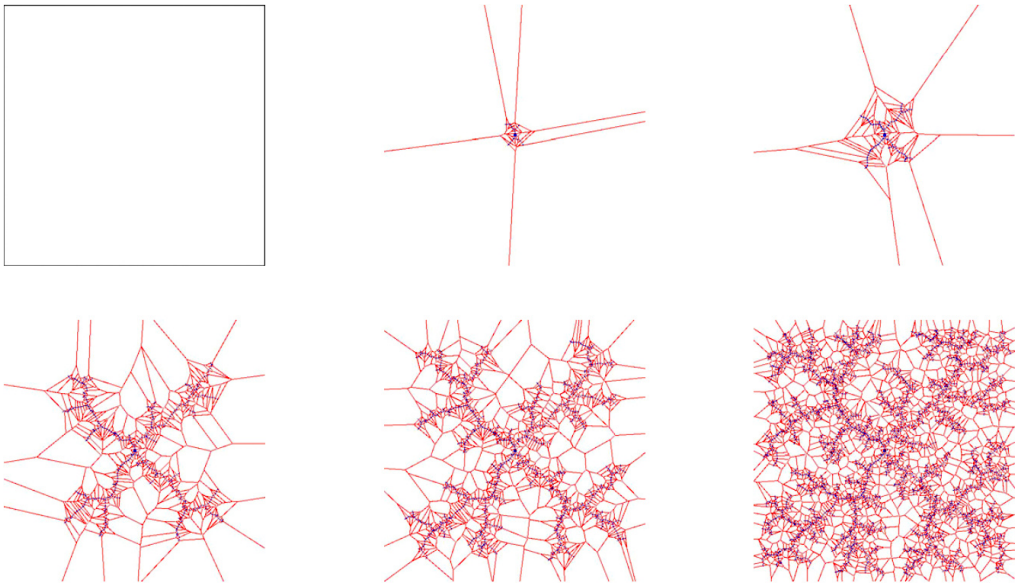

Figure 3.2 RRT build with adaptive branch lengths.

redundancy. For example, in large obstacle free regions like long corridor, free lanes etc. single long branch of the tree can be grown instead of many fixed sized branches that does the same job. Avoiding many such nodes can save computational time by reducing the complexity which is a function of number of nodes, O(n). Also the control program need to be invoked each time a current branch is broken and new node becomes the temporal destination.

The next step in the generation of truly optimal RRT is to optimize the path generated. This section describes the RRT+ algorithm and the two proposed key concepts –Intelligent Sampling and Path Optimization. Initially, RRT+ randomly searches the state space as RRT does. Similarly, the first path is found just like the RRT would try to find a path by random sampling in the configuration space. Once this first path is found, it then optimizes it by interconnecting the directly visible nodes. This optimized path yields biasing points for intelligent sampling. At these

biasing points, sampling takes place at regular intervals, which are governed by a constant b that in turn depends upon the biaisng ratio. This process is continued, as the algorithm progresses and the path is optimized continuously. Whenever, a shorter path is found, the biasing shifts towards the new path. This process is outlined in **Algorithm 3.**

In **Algorithm 3** the $Sample$ function samples a state $q_{rand} \in Q_{free}$. The $Nearest$ method returns the nearest neighbour from the tree, $q_{nearest} \in \Psi$ to the randomly chosen from point. The $Steer$ method finds the control input that drives the system from q_{rand} to $q_{nearest}$. This gives the next node q_{next} at an adaptive distance Δq from $q_{nearest}$ to q_{rand}. The process to reorder the map with varying resolution using image segmentation technique takes place inside the $Steer$ function. The function $Insertnode$ adds a new node q_{next} to the tree and connects this to the already existing parent node q_{parent}. The cost value is updated that sums up the current cost depending on Euclidean distance with the previous cost. In the $Rewire$ function, the cost of the node Q_{near} via the node q_{new} is compared with the previous older costs. If it is less for a particular node, its parent q_{parent} is changed to q_{new}. The results of these algorithms are stated and compared in Section 3.9. They are demonstrated in both real-time and synthetic environments and evaluated separately.

$\Psi \leftarrow InitialTree();$

$\Psi \leftarrow InsertNode(\emptyset, q_{init}, \Psi);$

while ***i*** *is less than* ***N*** **do**

  if *i=n+b,n+2b,...* **then**

    $q_{rand} \leftarrow Sample(i, q_{beacons});$

    else

      $q_{rand} \leftarrow Sample(i);$

      $q_{nearest} \leftarrow Nearest(\Psi, q_{rand});$

      $(x_{next}, u_{next}) \leftarrow Steer(q_{nearest}, q_{rand});$

      if *Obstaclefree*(x_{next})**then**

        $Q_{near} \leftarrow Near(\Psi, q_{next});$

        $z_{min} \leftarrow Chooseparent(Q_{near}, q_{nearest}, q_{near}, x_{next});$

        $\Psi \leftarrow InsertNode(q_{min}, q_{next}, \Psi);$

        $\Psi \leftarrow Rewire(\Psi, Q_{near}, q_{min}, q_{next});$

        if *InitialPathFound* **then**

          $n \leftarrow i;$

        end

        $(\Psi, directcost) \leftarrow PathOptimization(\Psi, q_{init}, q_{goal});$

        if *(directcost)* **then**

          $q_{beacons} \leftarrow PathOptimization(\Psi, q_{init}, q_{goal});$

        end

      end

    end

  end

  return Ψ

Algorithm 3: RRT+ Algorithm.

3.7 Path Smoothing using Bezier Curves.

A Bezier curve is a parametric curve frequently used in computer graphics and related fields. They are often used to model smooth curves. As the curve is completely contained in the convex hull of its control points, the points can be graphically displayed and used to manipulate the curve intuitively. Quadratic and cubic Bezier curves are most common. Higher degree curves are more computationally expensive to evaluate. When more complex shapes are needed, low order Bezier curves are patched together, producing a composite Bezier curve. The motivation is drawn from its application in computer graphics to smooth the high dimensional arcs and curvy edges.

In our situation, a set of coordinates from the trajectory of the planned path is obtained by the implementation of **Algorithm 3.** Due to various factors like the scaling factor that relates the probabilistic map, 2.7, with the original environment size and other approximations the trajectory has obtained due to raw planning is highly discontinuous and coarse. This trajectory has to be smoothed for reasons like − better efficiency of control actions executed by the mobile robot and minimized localisation error. Hence the list of coordinates are re−sampled to make them evenly spaced using an one dimensional median filter. The resampled list of way points are smoothed using the parametric curve. The performance metrics like − minimum localisation error, resistance to wobbling and smooth control actions are found to greatly improve after using this technique.

A Bezier curve is defined by a set of control points P_0 through P_n, where n is called its order (n = 1 for linear, 2 for quadratic, etc.). The first and last control points are always the end points of the curve; however, the intermediate control points (if any) generally do not lie on the curve.

Given points P0 and P1, a linear Bezier curve is simply a straight line between those two points. The curve is given by Equation 3.5, which is equivalent to linear interpolation.

$$\mathbf{B}(t) = \mathbf{P}_0 + t(\mathbf{P}_1 - \mathbf{P}_0) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1 \ , t \in [0,1] \tag{3.5}$$

A quadratic Bezier curve is the path traced by the function B(t), given points P0, P1, and P2 as in Equation 3.6.

$$\mathbf{B}(t) = (1-t)[(1-t)\mathbf{P}_0 + t\mathbf{P}_1] + t[(1-t)\mathbf{P}_1 + t\mathbf{P}_2] \ , t \in [0,1] \tag{3.6}$$

A schematic representation in Figure 3.3 shows how the parametric Bezier curves are constructed with the parameter t ranging between 0 and 1. The red line with a dotted end shows the path travelled by the locus.

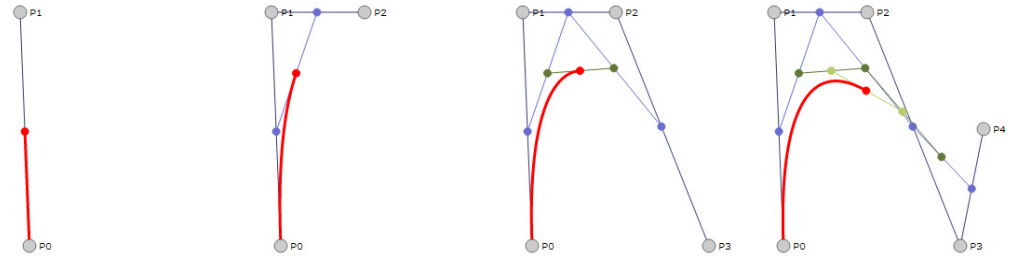

Figure 3.3 Construction of parametric Bezier curves.

Quadratic Bezier curve is predominantly preferred and the same is also used in smoothing the RRT path. The Figure 3.4 shows the raw trajectory or RRT and Figure 3.5 shows the smoothed version using Bezier curves.

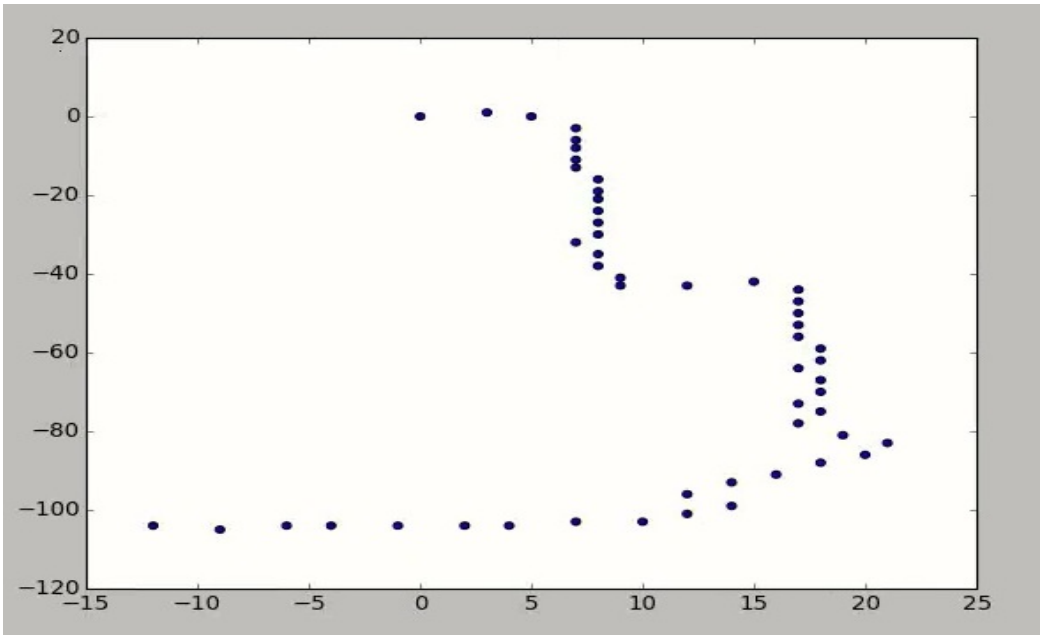

Figure 3.4 Raw coordinates obtained by RRT+ planning.

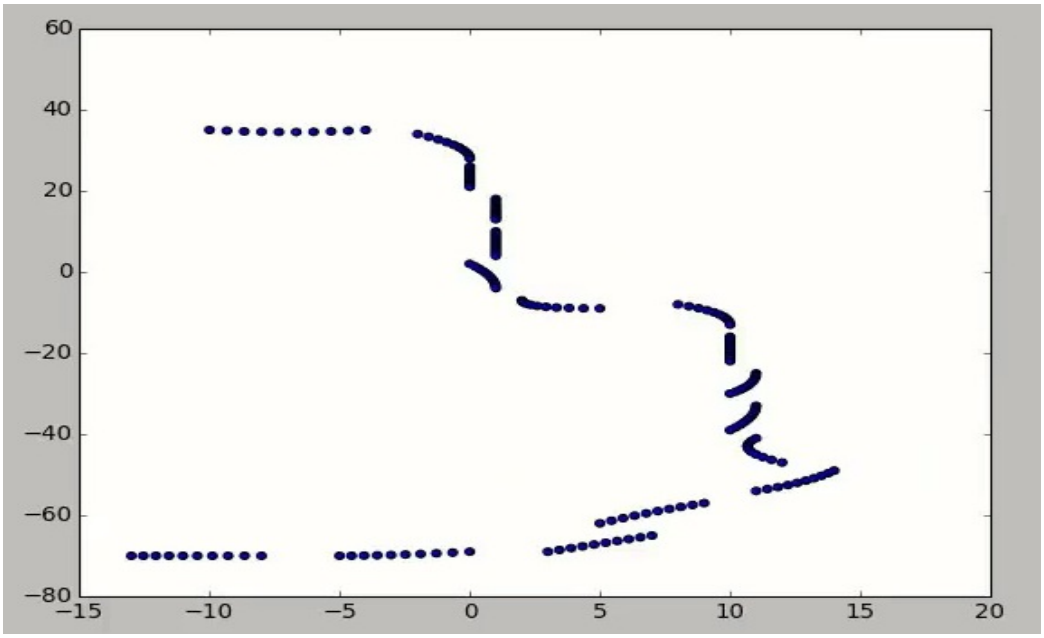

Figure 3.5 Smoothed trajectory of raw RRT+ plan using Bezier curves.

3.8 Control of Mobile Robot.

The control of mobile robot is also an integral portion of this chapter. It deals with building up velocity profile or/and a trajectory of control actions for physically manoeuvring the mobile robot along this planned path. Mathematically it involves calculating an appropriate rotational and translational matrix that is multiplied with the system matrix, which tells the current pose of the robot to arrive at a new state. The rigid body transformation matrix must account all the minimum number of dimensions that are mandatory for expressing the posture of mobile at any point.

Another chief aspect to be considered is the cost in executing a control action. In any inverse kinematic task, two points in a high dimensional space can be con-

nected by any order polynomial. The polynomial that consumes the least energy while execution must be chosen. A control input to a mobile robot is always abides the assumptions of closed loop system. Using the constants that relate the energy consumed for unit change in the odometry encoder or any feedback mechanism the total cost associated with a control action can be estimated. The task with the least estimate of cost is finalized and corresponding voltage values are supplied to motors and other tractable actuators.

As the direction of translation and angular rotation with the least cost is decided suitable velocity profiles can be constructed. A velocity profile considers the dynamics involved in the nature of rotation and translation exhibited by the mobile robot. It is not necessary that the robot should hit the environment with constant velocity through out the way point navigation. Since the situation is similar to a normal navigation task exhibited by a human similar motivations can be drawn. Depending upon the length of paths and angular arcs, variable velocity can be given to the actuators which is accelerated by a factor linearly dependent on the lengths.

3.9 Experimental Results.

The various results obtained by the experiments demonstrated in this section are summarized here. The performance measures of different parameters involved in deciding the better approach are compared and graphed. The proposed algorithm is tested to prove better results than the previous one in both real time and simulation environments.

Input: Set of nodes of planned path as Ψ, Robot$'$s pose as Γ
Output: Velocity Profile (Direction and Velocity vector.)
while $\Psi \neq \emptyset$ **do**
    $slo \leftarrow atan2(\Gamma_{ori}, \Psi_{next})$
    if *slo is less than 0* **then**
        $slo = slo + 2\ \pi$;
    end
    if Γ_{ori} *is less than slo* **then**
        if $slo - \Gamma_{ori}$ *is greater than* π **then**
            $Ang \leftarrow 2\pi$-$(slo - \Gamma_{ori})$;
        end
        else
            $Ang \leftarrow slo$ - Γ_{ori};
        end
    end
    if Γ_{ori} *is greater than slo* **then**
        if $\Gamma_{ori} - slo$ *is greater than* π **then**
            $Ang \leftarrow 2\pi$-$(\Gamma_{ori}$ - $slo)$;
        end
        else
            $Ang \leftarrow \Gamma_{ori}$ - slo;
        end
    end
    $Dist \leftarrow EuclideanDistance(\Gamma_{x,y}, \Psi_{x_{next}, y_{next}})$;
    RotateTranslate(Ang,Dist);
end

Algorithm 4: Mobile robot control for Way point navigation.

The algorithms of RRT and RRT+ are run on a synthetic map to display the visual distinction between both the algorithms. The results of the path planned in various environments are displayed in Figure 3.6. It can be seen clearly that RRT+ plans better and simple path in comparison with basic structure of RRT. Figure 3.6 showing path planned RRT and RRT+ are the implementations of them as described in **Algorithm 2** and **Algorithm 3**.

By profiling the performance of these two different algorithms various parameters have been graphed. The cost of a branch based on Euclidean metric is calculated for every iteration for both the algorithms. It can be seen from Figure 3.7 that the cumulative cost of RRT+ is much less than basic RRT. Also as expected the cost of construction of each branch decreases as we proceed due to the fact that the length of the branch decreases with time.

Figure 3.8 highlights the running time ratio of RRT to RRT+. The graph shows that the ratio is always greater than 1 for the implementation of both algorithms, proving the time efficiency of RRT+ over basic format of RRT.

Figure 3.9 demonstrates the behavior of RRT and RRT+ in fifty different obstacle environments. For each environment, the cost is plotted for a fixed number of iterations n. It can be seen that the graph of RRT+ consistently stays below the graph of RRT, showing that in all these environments the path costs found by RRT+ remains less than the path costs of RRT, hence proving the efficiency of RRT+.

The formulated algorithm is used to plan a path over the map in Figure 2.7 built using Occupancy grid method as explained in Chapter 2. The red circle and orange circle indicate the start and end nodes in the map. This path generated by RRT+ was smoothed using the Bezier curves in Section 3.7. Before smoothing the path nodes generated by RRT+ is resampled to remove the noisy ones. The resampled version of the planned nodes were passed through the bezier curves and interpolated to get more finely placed samples. The map of IIT Madras Computer Science department over which the planning is done is showed in Figure 3.10. The red line between the start and goal nodes indicate the structure of the path.

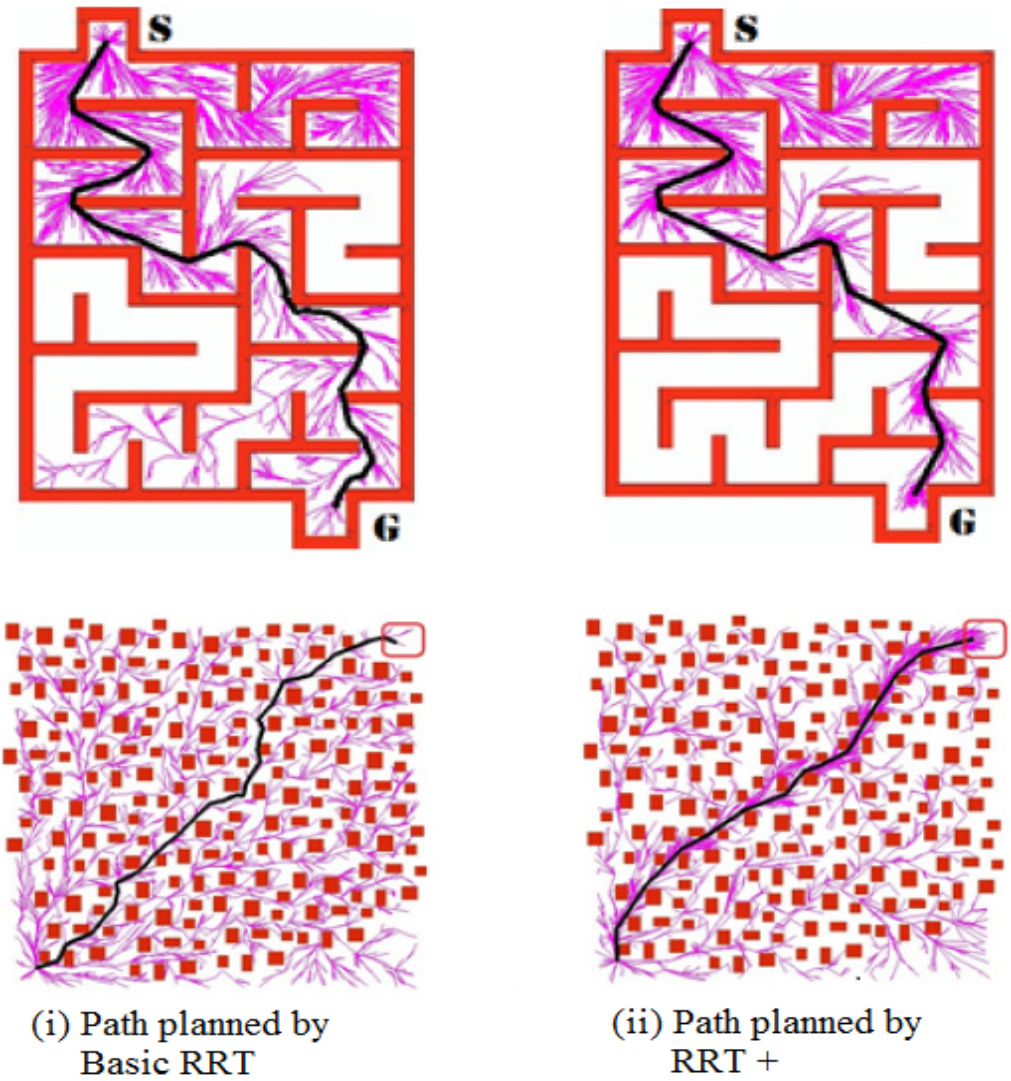

Figure 3.6 Comparison of RRT and RRT+ in synthetic map.

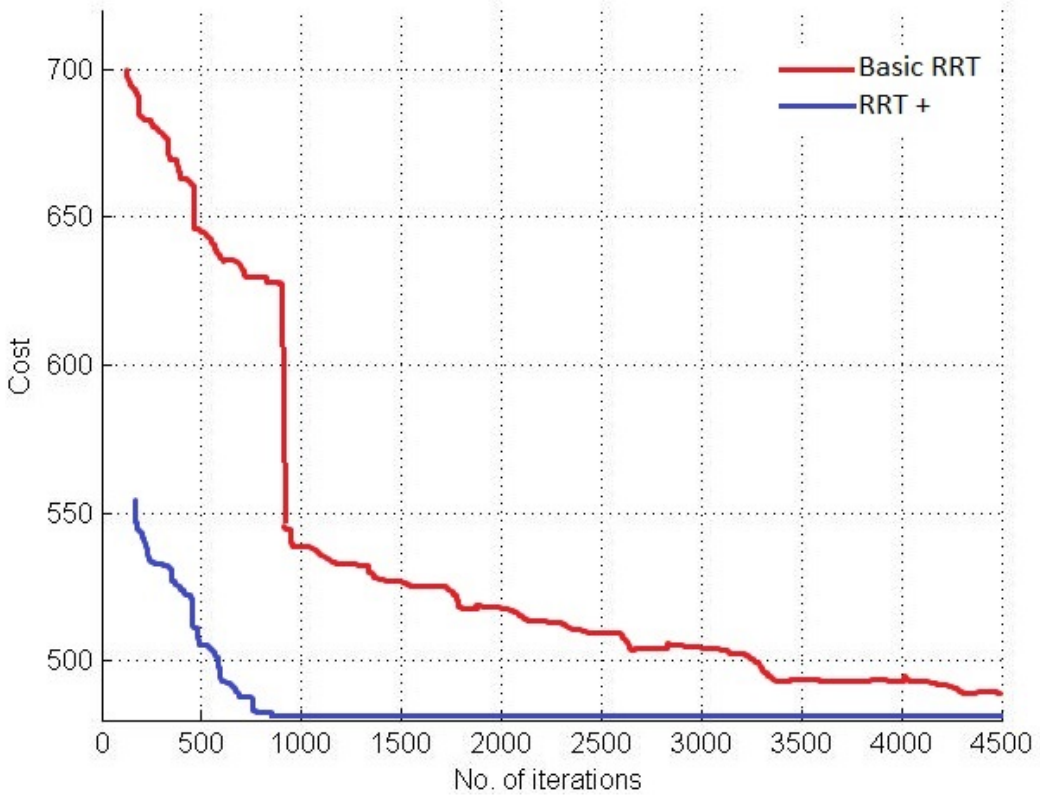

Figure 3.7 Costs are plotted against iterations showing the rate of convergence of both RRT and RRT+.

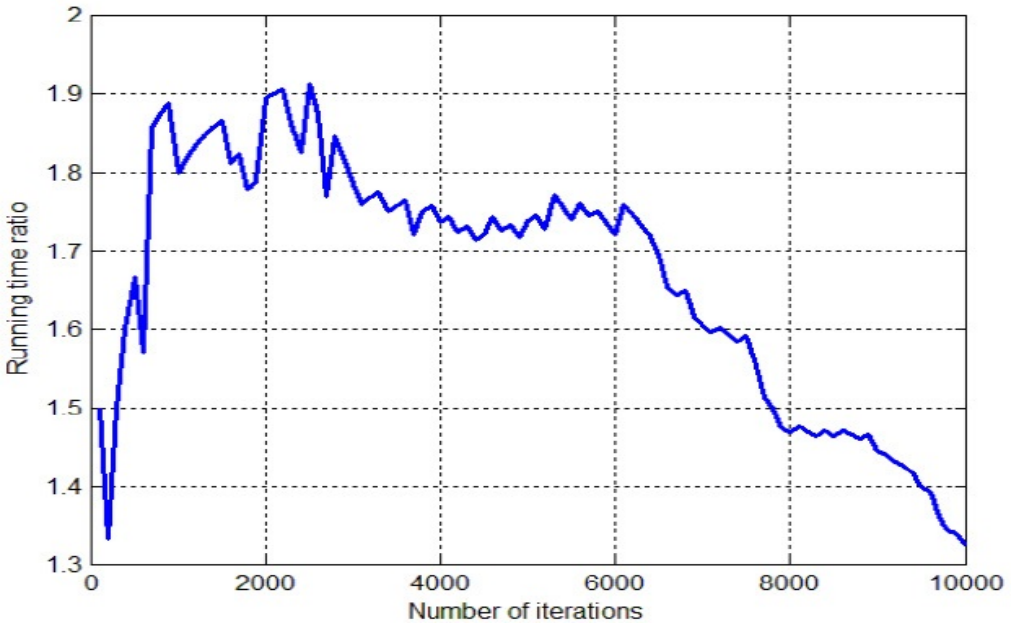

Figure 3.8 Running Time Ratio of RRT over RRT+ is plotted against iterations. Constant b is taken as 2 for RRT+.

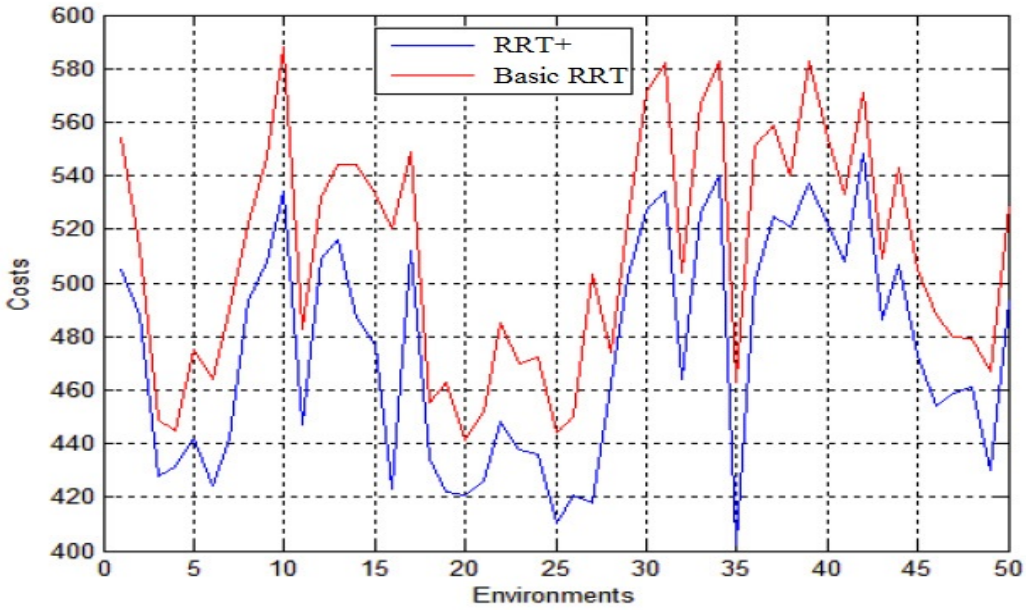

Figure 3.9 Costs of path in fifty different environments for 2500 iterations with dynamic biasing ratio scheme.

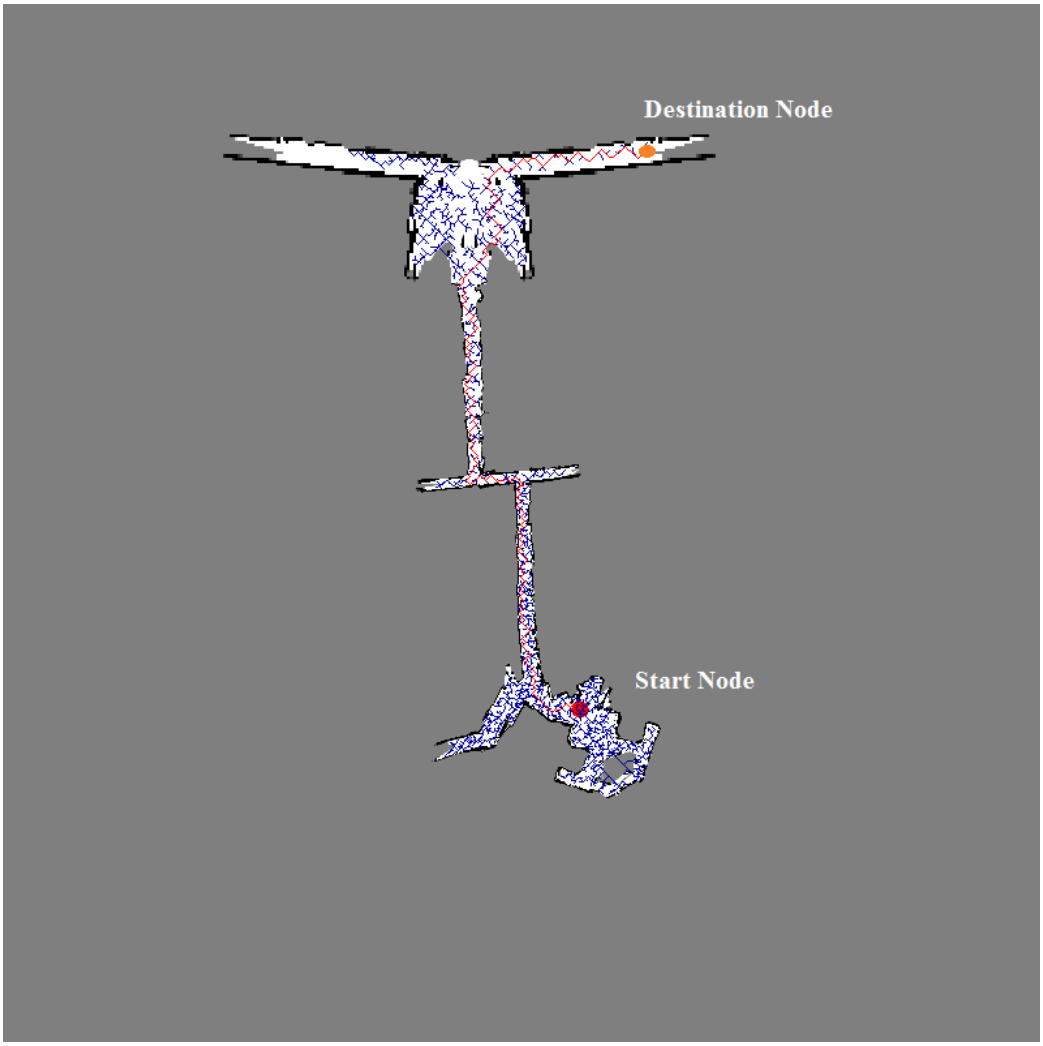

Figure 3.10 RRT+ implemented for planning a path on probabilistic map of Computer Science department of IIT Madras(red line indicates the route of the planned path).

CHAPTER 4

REAL-TIME IMPLEMENTATION OF PARTICLE FILTER WITH ADAPTIVE SAMPLING.

Operating robots autonomously in unknown and noisy environment requires building map of the environment and locating the position of the robot within the map from noisy measurements. This investigation presents a localization method for the wheeled mobile robot (WMR) operating in unknown and noisy environment using particle filter. To implement the particle filter, measurements on current orientation and position are obtained from on board sensors and sensor fusion technique is used to minimize the errors and uncertainty to determine the pose of the robot. As a result, the environment map and robot position can be determined with greater accuracy than existing methods in literature. The results are illustrated using simulation on unicycle robot model.

4.1 Introduction

The primary motivation for localisation and mapping is drawn from the control mechanism of human body that is responsible for navigation. When a mobile navigating object is blindfolded there is no feedback mechanism and the target to be achieved by any control action remains uncertain. It is natural for both the

sensors and actuators to accumulate non-linear errors and drive away the system from estimated state at any point. In order to prevent this error that evolves over several parameters, the error needs to be estimated and modelled. Because it is not possible to keep accumulating error and compensate once in every finite period. Such a task is a *NP* hard problem and becomes more difficult at higher dimensions.

To avoid such daunting situations it is necessary to work under a closed loop system with some tracking model that constantly compensates this accumulating error. An inspiration was drawn from the vision and other feedback systems in human beings that works in a similar way by providing feedback to the navigation task. Several state estimation filters have been proposed with these motivations. Each state estimator has its own set of pros and cons. Few are strictly dependent on the model of system whereas few are independent. Certain estimation filters are applicable only to gaussian density error arising in the sensors and system model and some filters are independent of the type of noise, unimodal or multimodal. Few filters work only if the model response is of first order and few work for any type of system response.

4.2 Related Work

In autonomous robotics, localization has a key role in tracking the robot, which in turn helps the robot to traverse through the unknown terrain. Localization without a known map is a tedious work. It also brings many environment specific constraints like slippage, surface friction, wind turbulence (in case of aerial robots) etc. This problem is defined in many robotics literature as SLAM, *Simultaneous Localization And Mapp*

[26]. Simultaneous Localization and mapping has been a trending area of research in the field of Robotics and Artificial Intelligence since 90s [3] [28]. Various researchers have proposed many different approaches and solutions for this problem, but still this problem is open only because of its environment dependencies. The methods proposed by many literature are environment dependent [28]A Solution to the Simultaneous Localization and Map Building (SLAM) Problem..

Localization can be classified into two categories

- Relative or local Localization [30]Robot Localization Using Relative and Absolute Position Estimates
- Absolute or Global Localization [30]

The former method makes use of the sensors like wheel encoders, gyroscopes, accelerometers, etc. While the later method makes use of laser range finder, satellite Data, GPS etc. along with sensor fusion techniques. Relative Localization which uses odometry data from wheel encoders is a dead reckoning measurement which is prone to accumulation of error over the period of time. This might result in huge error after certain period of time. Thus the position given by the odometry model is largely deviated from the actual position and orientation of the robot. J.Borenstein and L.Feng has done a considerable work to improve odometry by a calibration technique called UMB mark test [6]. They have also proposed a new technique called Gyrodometry which manipulates the odometry according the measurements from wheel encoders and gyroscope. Thrun et.all [5] proposed a beacon based method for solving SLAM problem. All the works discussed above doesnt provide the complete solution for the SLAM problem. Hence prob-

abilistic estimation techniques have attained good attention in recent years for Robot localization problem. The well-known Kalman filter works well for linear system estimation with Gaussian noise. But the mobile robot is a nonlinear system with non-Gaussian noise for which Extended Kalman filter (EKF) can be used. EFK utilizes the data from various on-board sensors like laser range finder, wheel encoder, inertial measurement unit, camera (2D depth) etc. using sensor fusion techniques [27]. EKF uses least mean square method for error reduction in state hypothesis of the robot. EKF is a sequential algorithm which tracks the robots pose with the help of odometry model and current observations. It finds features from the environment and uses them to keep track of the system state.

4.3 Particle Filter as a State Estimator.

Although EKF works fine for the nonlinear and non-Gaussian systems, it has several intrinsic problems like feature extraction, data association, slip etc. which might cause the filter to diverge. The features in the environment has a major role in the convergence of the filter. As long as the features are visible for the robot, it helps the filter to give a better estimate. In the absence of features in the environment, the error state estimation will be accumulated as the robot will be traversing blind (kidnapped robot problem) which increases the belief on odometer more. This would cause the filter to deviate. A better feature extraction algorithm will help reducing this effect. Similarly, the slippage in the robot will not be recorded in odometry, but it inculcates bigger deviations in state estimation. Data Association is one such factor on which the filter relies more for convergence. It is nothing but the association of the features from current scan to the previously seen fea-

tures. EKF uses data association to calculate the innovation factor from which the Kalman gain is calculated[]. Data association is the most difficult and tricky part in EKF algorithm. Single wrong data association would lead to a huge divergence in the state estimation. This effect would propagate in further measurements and the system finds it difficult to fallback.

Hence a better nonlinear filtering technique which has higher performance than the existing nonlinear filters is required. A sequential Monte Carlo algorithm based filter referred as Particle Filter was developed by [4]. The particle filter overcomes the disadvantages of EKF and provide better tracking of the robots pose. The robot is fixed with sensors like optical wheel encoders, Microsoft kinect sensor, hokuyo laser range finder, phidget spatial sensor, camera, proximity sensor, SONAR and GPS. The robots odometry is calculated from the wheel encoder data. As stated earlier the odometry tends to have accumulated error because of the use of dead reckoning sensor for state updates. The Laser range finder is an absolute range measurement device used for observations for every state estimation.

4.4 Mobile Robot Localization

Mobile robot localization is the problem of determining the pose of a robot relative to a given map of the environment. Mobile robot localization is the most basic perceptual problem in robotics. This is because nearly all robotics tasks require knowledge of the location of the robots and the objects that are being manipulated, although not necessarily within the global map.Localization can be seen as a problem of coordinate transformation. Maps are described in a global

coordinate system, which is independent of a robots pose. Localization is the process of establishing correspondence between the map coordinate system and the robots local coordinate system.

The localization of mobile robot is a self affordance that will make the robot know about its position and orientation in the global or local enviroment. The various tasks like robotic mapping and mobile robot path planning discussed in the previous chapter can be realized only when there is a strong estimate of the robot pose that could be relied upon. Any compromise or uncertainity in this basic step will lead to unsuccessful implementation of complex robotics task.

Knowing the pose $x_t = (x\ y\ \theta)^T$ of the robot is sufficient to determine this coordinate transformation, assuming that the pose is expressed in the same coordinate frame as the map. Unfortunately and herein lies the problem of mobile robot localization − the pose can usually not be sensed directly. Put differently, most robots do not possess a (noisefree!) sensor for measuring pose. The pose has therefore to be inferred from data. A key difficulty arises from the fact that a single sensor measurement is usually insufficient to determine the pose. Instead, the robot has to integrate data over time to determine its pose.

Hence any robotic task that requires the knowledge of robot pose should be preceded by the mobile robot localization. Various state space estimation filters are used in order to achieve this. The localization algorithm runs throughout the planning, control and mapping process to ensure highly reliable experimentation and implementation.

4.5 The Particle Filter Algorithm

The particle filter algorithm for Wheeled Mobile Robot (WMR) localization using motion model and observation model is developed in this section. Particle filter is used to estimate the pose and orientation of WMR that has multimodal, Gaussian and non Gaussian data distribution. Particle Filter uses samples (known as particles) to represent the multimodal data distribution and each sample is associated with weight that influences the quality of the sample. It involves prediction and update steps sequentially.

Just like Bayes filter, particle filter is an alternative non parametric implementation. They approximate the posterior by a finite number of parameters. However, they differ in the way these parameters are generated, and in which they populate the state space. In particle filter the samples of a posterior distribution are called particles and are denoted by Equation 4.1.

$$\chi = x_t{}^{[1]}, x_t{}^{[2]}, ..., x_t{}^{[M]} \tag{4.1}$$

Each particle $x_t{}^{[m]}$ $(with\ 1 \leq m \leq M)$ is a concrete instantiation of the state at time t, that is, a hypothesis as to what the true world state may be at time t. Here M denotes the number of particles in the particle set χ_t. In practice, the number of particles M is often a large number, e.g., M = 1000. In some implementations M is a function of t or of other quantities related to the belief $bel(x_t)$.

$$x_t{}^{[m]} \sim p(x_t, z_{1:t}, u_{1:t}) \tag{4.2}$$

As a consequence of 4.2, the denser a subregion of the state space is populated by samples, the more likely it is that the true state falls into this region. As we will discuss below, the property 4.2 holds only asymptotically for $M \uparrow \infty$ for the standard particle filter algorithm. For finite M, particles are drawn from a slightly different distribution. In practice, this difference is negligible as long as the number of particles is not too small (e.g., $M \geq 100$).

Algorithm Particlefilter(χ_{t-1}, u_t, z_t)

for *m=1 to M* **do**

  sample $x_t^{[m]} \sim p(x_t, z_{1:t}, u_{1:t})$

  $w_t^{[m]} = p(z_t \mid x_t^{[m]})$

  $\bar{\chi}_t = \bar{\chi}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$

end

for *m=1 to M* **do**

  draw i with probabiliy $\propto w_t^{[i]}$

  add $x_t^{[i]}$ *to* χ_t

end

return χ_t

Algorithm 5: Particle Filter Algorithm

The particle filter algorithm constructs the belief $bel(x_t)$ recursively from the belief $bel(x_{t-1})$ one time step earlier. Since beliefs are represented by sets of particles, this means that particle filters construct the particle set χ_t recursively from the set χ_{t-1}. A brief version of the particle filter implemented is given in **Algorithm 5**. The input of this algorithm is the particle set χ_{t-1}, along with the most recent control u_t and the most recent measurement z_t. The algorithm then first constructs a temporary particle set $\bar{\chi}$ which is reminiscent (but not equivalent)

to the belief $bel(\bar{x}_{t-1})$. It does this by systematically processing each particle $x_t{}^{[m]}$ in the input particle set χ_{t-1} as follows.

- The *sample* line inside first for loop generates a hypothetical state $x_t{}^{[m]}$ for time t based on the particle $x_{t-1}{}^{[m]}$ and the control u_t. The resulting sample is indexed by m, indicating that it is generated from the m−th particle in χ_{t-1}. This step involves sampling from the next state distribution $p(x_t \mid u_t, x_{t-1})$. To implement this step, one needs to be able to sample from $p(x_t \mid u_t, x_{t-1})$. The ability to sample from the state transition probability is not given for arbitrary distributions $p(x_t \mid u_t, x_{t-1})$. However, many major distributions in this book possess efficient algorithms for generating samples. The set of particles resulting from iterating *sample* step M times is the filters representation of bel(xt).

- The subsequent line calculates for each particle $x_t{}^{[m]}$ the so-called importance factor, denoted $w_t{}^{[m]}$. Importance factors are used to incorporate the measurement z_t into the particle set. The importance, thus, is the probability of the measurement z_t under the particle $x_t{}^{[m]}$, that is, $w_t{}^{[m]} = p(z_t \mid x_t{}^{[M]})$. If we interpret $x_t{}^{[m]}$ as the weight of a particle, the set of weighted particles represents (in approximation) the Bayes filter posterior $bel(x_t)$.

- The real trick of the particle filter algorithm occurs in the re−sampling stage in **Algorithm 5**. These lines implement what is known as re-sampling or importance resampling. The algorithm draws with replacement M particles from the temporary set $\bar{\chi}_t$. The probability of drawing each particle is given by its importance weight. Resampling transforms a particle set of M particles into another particle set of the same size. By incorporating the

importance weights into the resampling process, the distribution of the particles change: whereas before the resampling step, they were distribution according to $bel(x_t)$, after the resampling they are distributed (approximately) according to the posterior $bel(x_t) = \eta\, p(z_t \mid x_t^{[m]}) \bar{bel}(x_t)$. In fact, the resulting sample set usually possesses many duplicates, since particles are drawn with replacement. More important are the particles that are not contained in χ_t: those tend to be the particles with lower importance weights.

The resampling step has the important function to force particles back to the posterior $bel(x_t)$. In fact, an alternative (and usually inferior) version of the particle filter would never resample, but instead would maintain for each particle an importance weight that is initialized by 1 and updated multiplicatively:

$$w_t^{[m]} = p(z_t \mid x_t^{[m]}) w_{t-1}^{[m]} \tag{4.3}$$

Such a particle filter algorithm would still approximate the posterior, but many of its particles would end up in regions of low posterior probability. As a result, it would require many more particles; how many depends on the shape of the posterior. The resampling step is a probabilistic implementation of the Darwinian idea of *survival of the fittest*: It refocuses the particle set to regions in state space with high posterior probability. By doing so, it focuses the computational resources of the filter algorithm to regions in the state space where they matter the most. A flowchart and schematic representation of the working of one dimensional Particle filter is displayed in Figure 4.1 Figure 4.2 thus providing an intuitive understanding (standard notations used in this chapter follow).

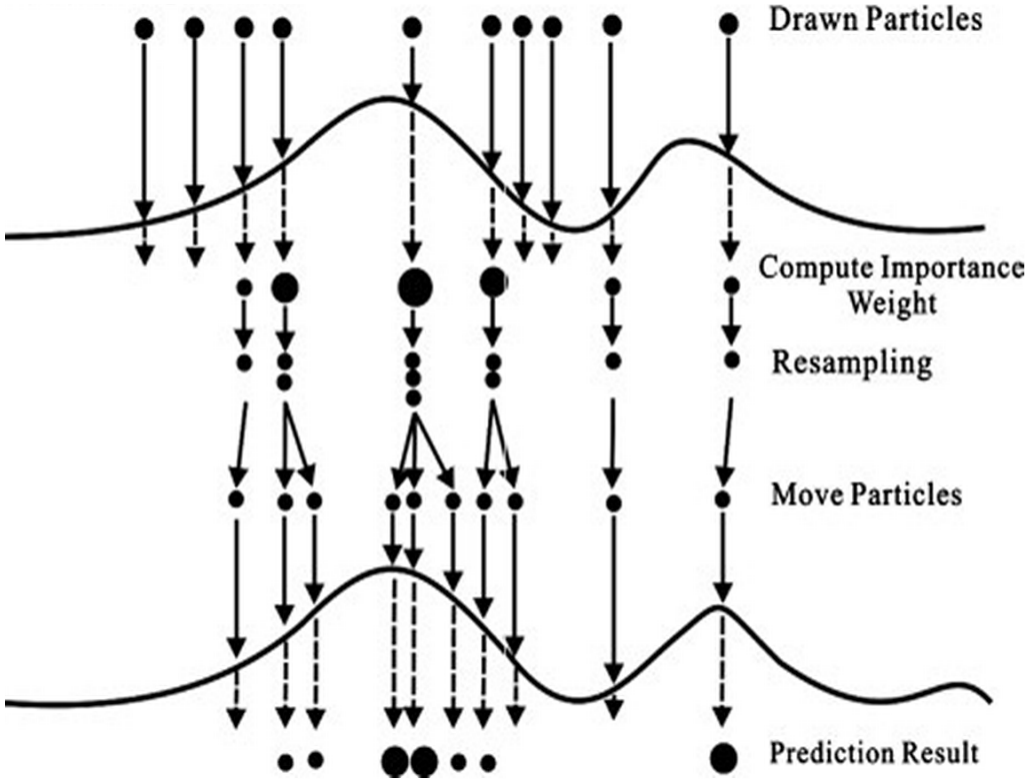

Figure 4.1 Flowchart showing the steps involved in particle filtering.

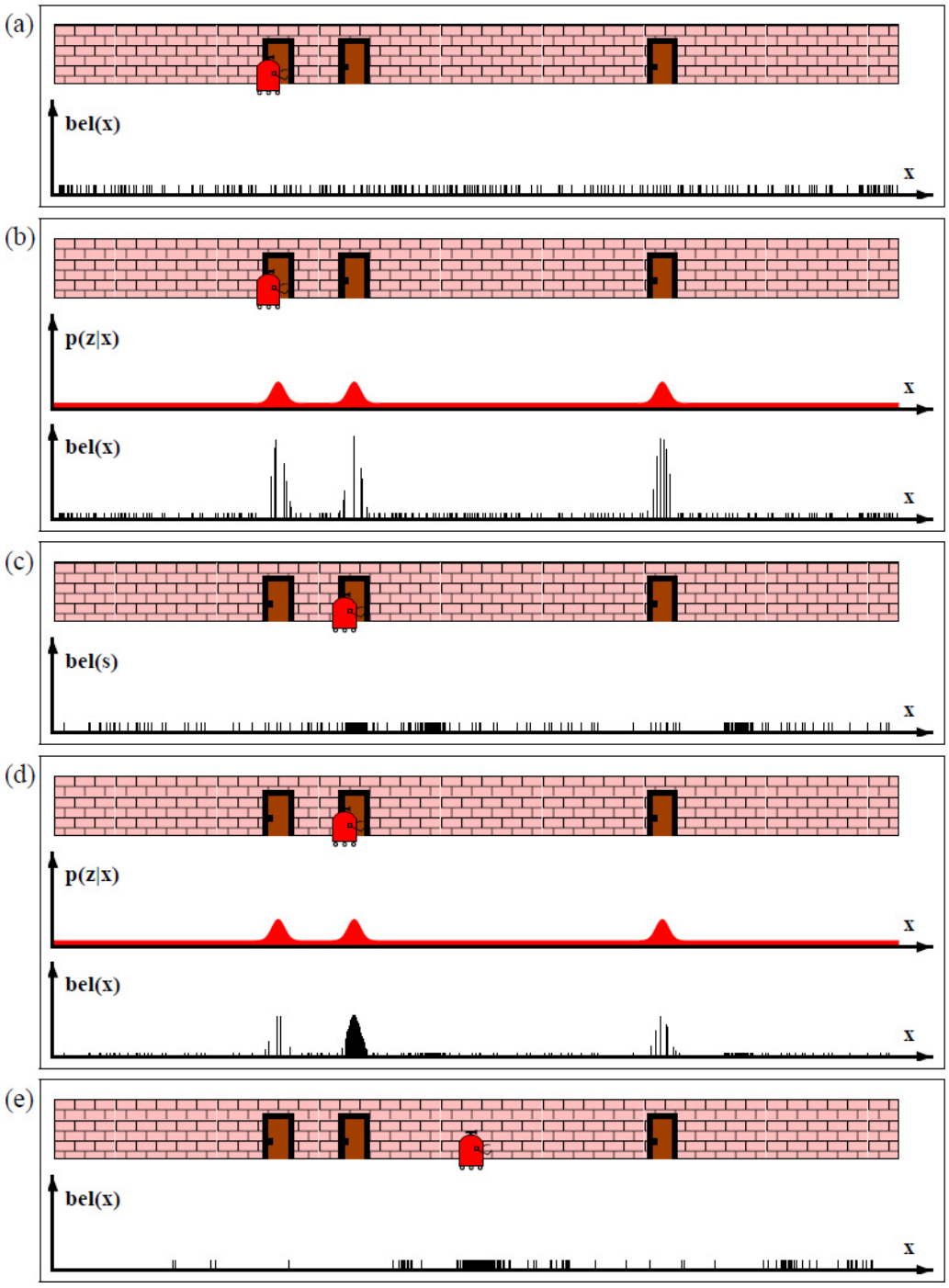

Figure 4.2 Working of Particle Filter. Adapted from [10]

4.6 Adaptive Sampling for Particle Filter

The robot sensors and actuators are subjected to noise which follow a particular distribution. Since most of the error source in nature are Gaussian we normally prefer a standard normal density function from which we derive samples. This is generally used in particle filter to randomly throw particles all over the environment and also as a system noise that is added to every particle after motion update during each iteration. But this procedure seems inefficient as we intrinsically assume an approximation that considers any type of unknown noise associated with the robotic system as Gaussian. Due to this approximation the filter may never converge as the system noise characteristic does not match with the assumed one.

To avoid this approximation and generalized distribution for sampling, we have devised an algorithm that adaptively alters the distribution it maintains. The noise characteristics of various sensors present in the robot are modelled and corresponding distribution is established. This noise distribution may not necessarily follow Gaussian or any standard distribution. The noise model is built by traversing the sensors and actuators randomly over any known environment. The exact measurements of the environment are manually taped and recorded. The various estimates made by the sensors are used to build a distribution by assigning the exact values as the mean of the distribution.

A normal probability plot is used to analyse the variance or error in the data. The normal probability plot is a graphical technique to identify substantive departures from normality. The normal probability plot is formed by plotting the sorted

data vs. an approximation to the means or medians of the corresponding order statistics. Some users plot the data on the vertical axis; others plot the data on the horizontal axis. Deviations from a straight line suggest departures from normality. This also serves as a test for the variance in the measurement to follow normal distribution or not. Once the nature of variance is analysed, the corresponding histogram is used to fit an unconventional probability density function.

The newly structured density function is used as a basis for sampling particles and adding uncertainity during motion update. It can be intuitively understood how this density function performs better than any other arbitrary assumption. This holds true due to the fact that there is no approximation involved as we are using the exact noise model of the sensors. To understand better, the noise model of error is constructed and the density function is shown in Figure 4.4. The Figure 4.3 shows the normal probability plot and it is found to be linear.

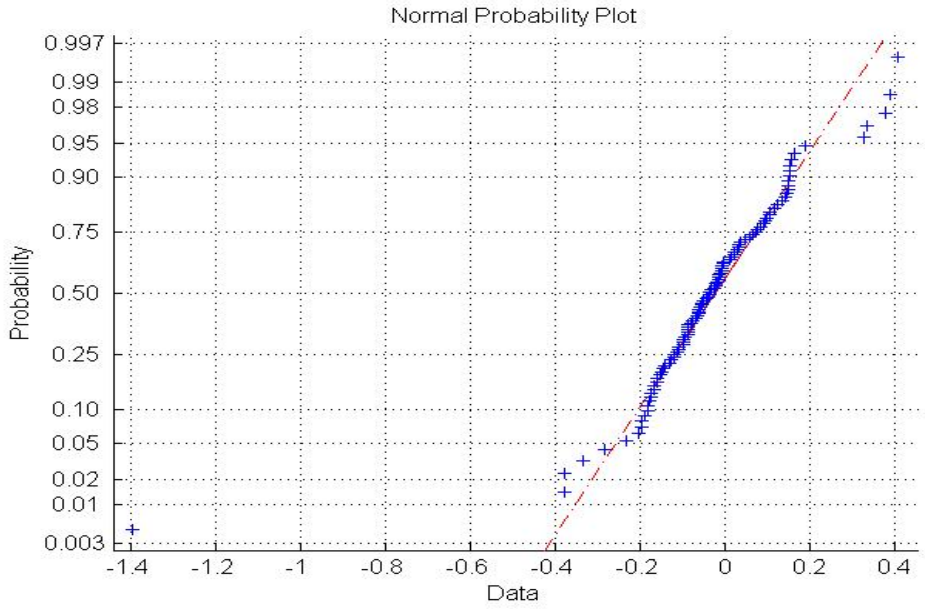

Figure 4.3 Normal probability plot of error function.

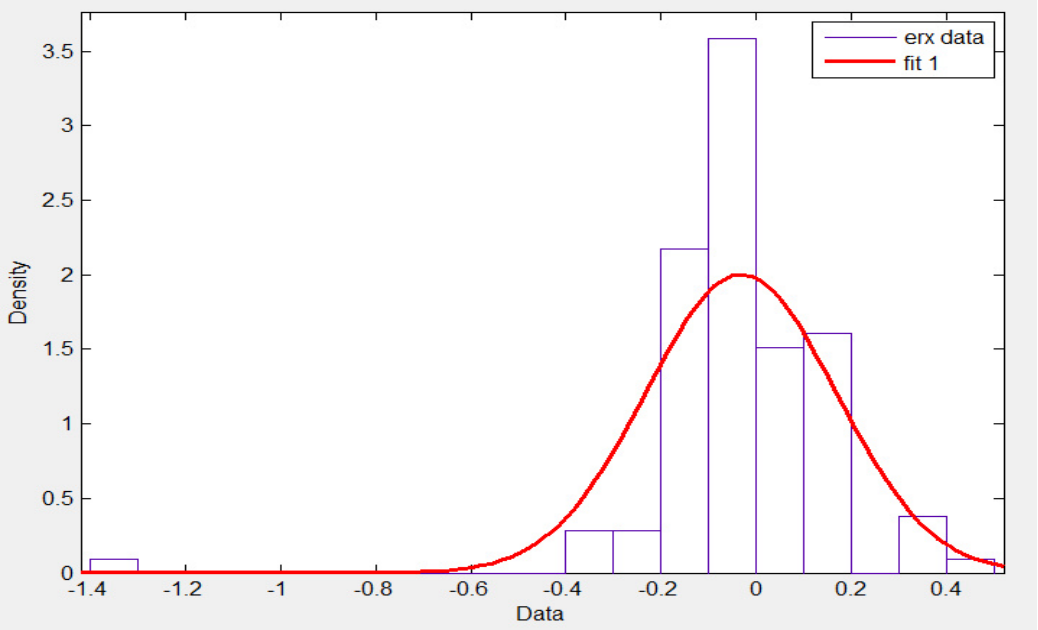

Figure 4.4 Modelled probability density function of sensor variance.

4.7 Experimental Results

With the technique of adaptive sampling the particle filter produced faster convergence rates as expected. This could be verified from the graph as shown in Figure 4.5. The graph shows the relationship between the state estimation variance and the number of re-sampling instances. By employing particle filter with adaptive sampling localization is attempted in various datasets and synthetic maps. One such famous robotic test environment is shown in Figure 4.6 where the estimation was found to attain optimality at a faster rate.

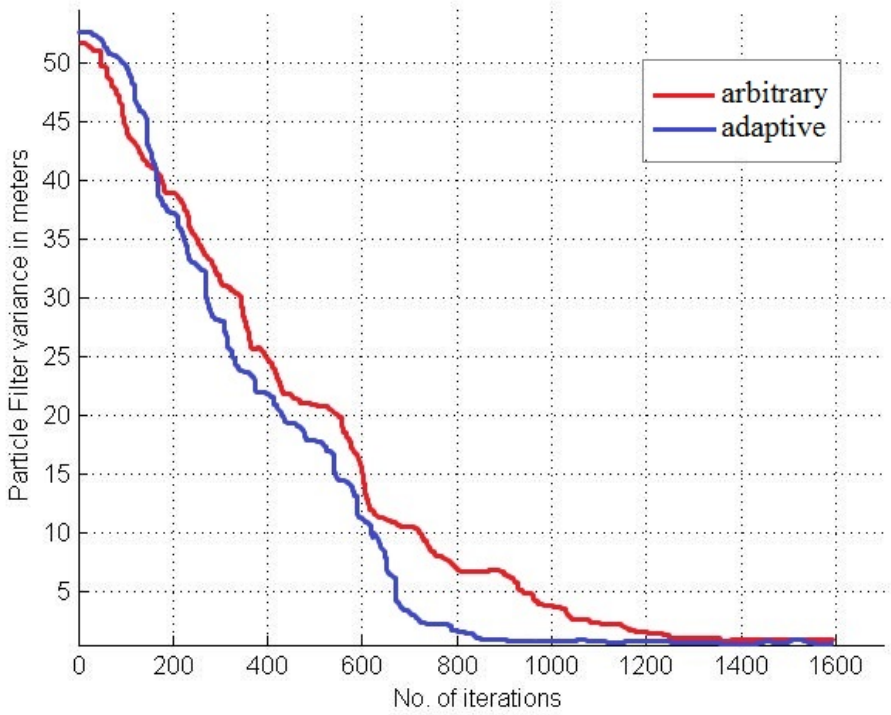

Figure 4.5 Comparison between particle filter with arbitrary sampling and adaptive sampling.

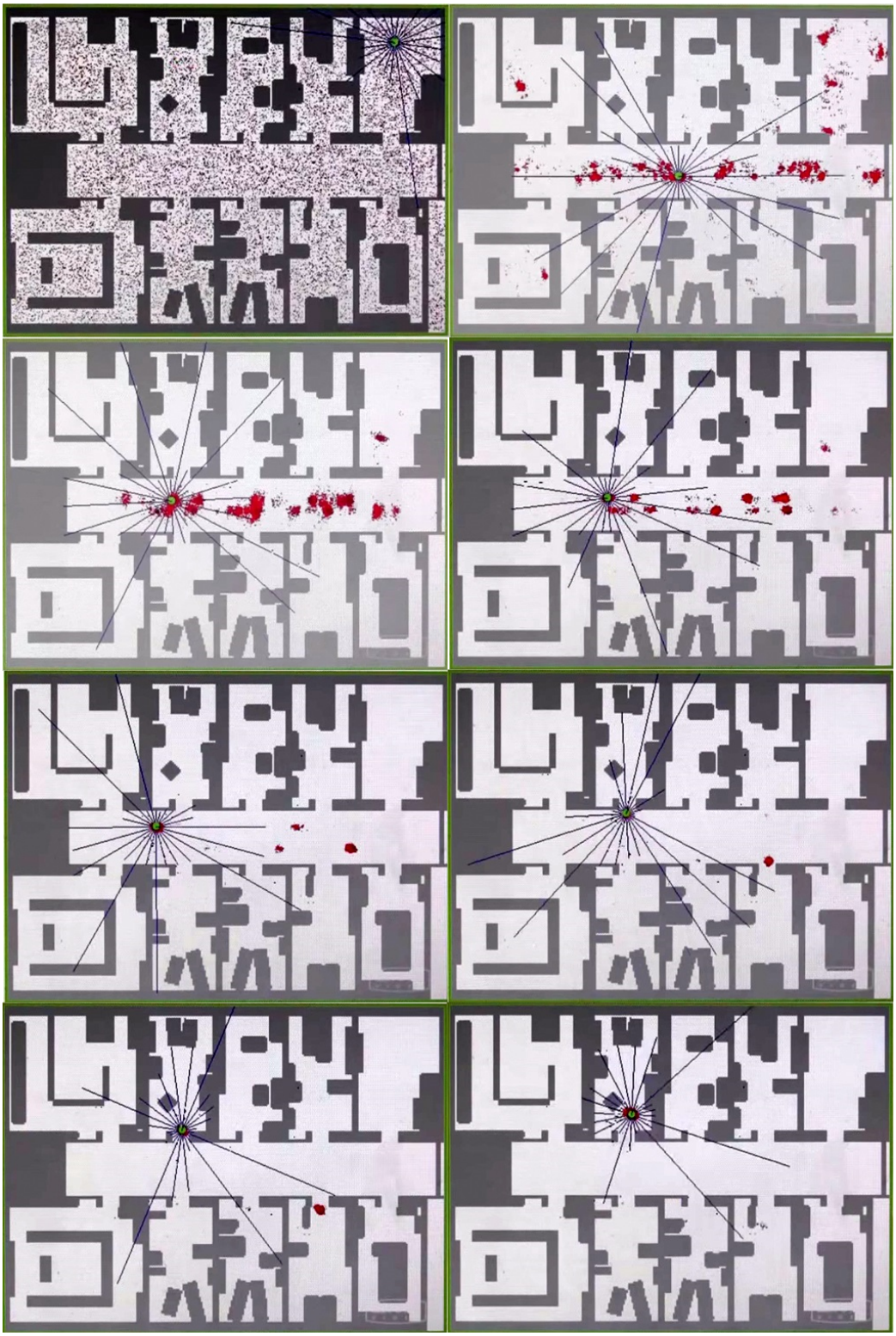

Figure 4.6 Various stages of particle filter convergence in simulated environment.

CHAPTER 5

CONSTRUCTION OF RELIABILITY MAP.

So far, many robotic tasks has been implemented and corresponding results have been obtained. From the nature of the algorithms and the intense of mathematics involved it could be seen that almost all the process consumes large amount of energy. Along with many other optimization problems that try to work for the betterment of a particular core robotic task, the overall energy consumed in completing a task is also an indispensable metric and must be seriously considered.

5.1 Introduction

A mobile robot subjected to random exploration in an environment will be fitted with so many sensors that are sufficient enough to perform sensor fusion techniques and other dictated problems. Currently, it is possible to guarantee the completion of a task when the robot is fault-tolerant and the task remains in the fault-tolerant workspace. The traditional definition of fault-tolerant workspace does not consider different reliabilities for the different sensors of the robot at different positions of the environment.

However, the aim of this work is to extend the concept of a fault-tolerant workspace to address the reliability of different sensors at various states of the system. Such an extension can provide an overall as well as a environment dependent view about the sensor with which several conclusions can be drawn. This

area in robotic mapping is currently an open problem and hopes to attract many research avenues. As a result, reliable fault-tolerant workspaces are introduced by using the novel concept of conditional reliability maps. Such a reliable fault-tolerant workspace can be used to improve the performance of robots provided that the confidence of each sensor remains reliable with respect to an already explored environment.

5.2 Related Work

Robots are increasingly applied for more advanced and complex tasks in various applications. This requires further study to improve their performance. For example, assistant robots have been designed for helping disabled or elderly people [34] [20]. These robots are required to be safe and reliable to prevent human injury and task failure. In some tasks, if the robot is unable to accomplish the tasks, then there will be the possibility of serious human injury. Furthermore, safety and reliability of surgical robots are critical to prevent patient injury because of robot failure [17] [35]. This is true for most applications of fault-tolerant robots from space exploration to hazardous material disposal where the robot failure can result in a catastrophic outcome [11]. Fault tolerance is important because it increases the dependability of the robots. However, robots are being applied to more advanced tasks where different levels of safety and reliability are required. Therefore, understanding the reliability of the robots is paramount. Currently, it is possible to guarantee the completion of a task when the robot is fault-tolerant and the task remains in the fault-tolerant workspace (FTW).

Research on fault-tolerant robotic systems has focused on the control of the manipulators (e.g. fault analysis and fault-tolerant motion planning or fault-tolerant controllers), belief of any sensor at every state in the environment or the design of the manipulators. Out of this the approach on sensor belief for environment states is an entirely new topic to be added to this list.

5.3 Reliability Mapping

The need to construct a truly reliable map can be understood by the following motivational example. The inspiration is drawn as usual from the control mechanism of the human body. The reliability map is a graph that is maps the degree to which a particular sensor should be trusted in order to estimate the pose of the robot. In case of human beings, the data obtained from the vision system while navigating in a dark room is useless. The prediction or observation made by the sensor system are completely unreliable. The navigation exhibited by the human in such a situation will be similar to the one negotiating the environment blindfolded. Hence in such situations the amount of energy spent in receiving totally useless data can be conserved. For human beings, this amount of energy is negligible. But considering a similar situation in case of a mobile, turning off unwanted sensor node depending on the vicinity of the robot can be of extreme use.

In order to build a reliability map the robot must get trained by the environment. A reliability map of an unknown environment is built by learning about the environment and making it known. The robot is left to randomly walk throughout the environment with any state estimation filter running in parallel. The estimator will correct the position of the robot with the inputs from various models of the

sensor. During this correction step the filter may assign variable beliefs to each sensor which depend on the factors like innovation between prediction and measurement models. The channel which supplies more reliable data when tested by different association techniques like [21] [37] are believed higher than the others.

For example, if the environment contains many reflecting surfaces like black glasses with lamination then the laser range finder may provide highly erroneous data. This is because the laser is not strongly reflected by illuminating and glassy surfaces and hence the data becomes unreliable. In such locations of the environment it is better to turn off the laser range finder and work with the remaining sensors. Another example where equal distribution of beliefs may fail is terrains in the environment that are rugged and uneven. In surfaces like them due to bumps and pits the odometry model may indicate a distance that is more or less than the absolute displacement. This is because the wheels were not rotated smoothly in coherent with the terrain. But the observation systems like laser range finder, kinect and sensors like GPS, accelerometer etc indicate a displacement that varies hugely from the odometry feedback and lies close to the absolute value. In these locations of the map the entire belief is shifted towards the vision system.

With this understanding, the distribution of beliefs among the sensors are recorded for every filter estimated states to which the robot has traversed. The beliefs obtained for every sensor is normalised and the corresponding values of red, blue and green are scaled and distributed over the map. The resultant colour produced by this combination is equivalent to the weight distribution of each sensor respectively. The belief distribution among the three sensors are laser range finder,

odometry and inertial measurement unit. The reliability map obtained by this procedure is shown in Figure 5.1

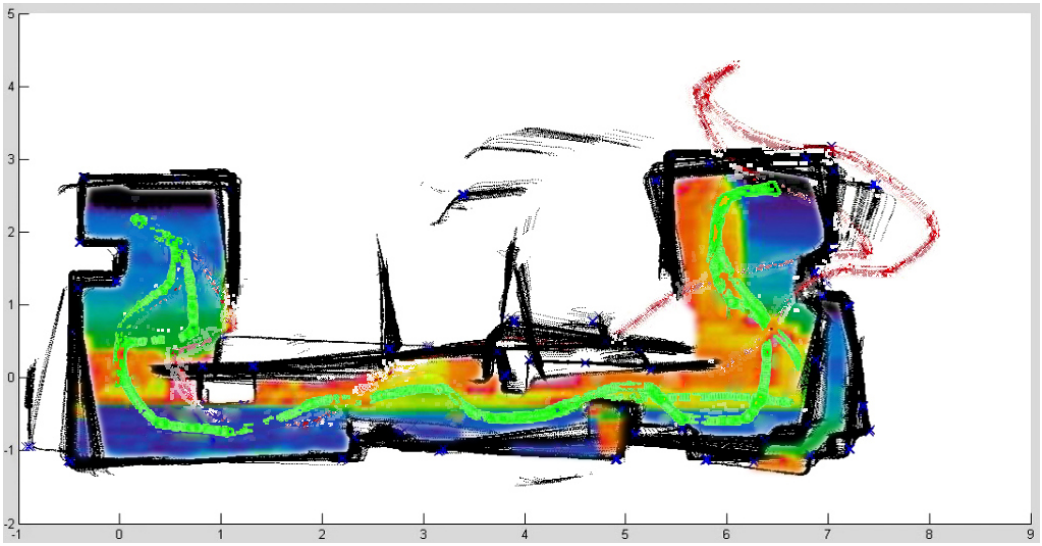

Figure 5.1 Reliability map of an indoor environment showing the belief distribution between the sensors at different locations of the environment.

Bibliography

[1] HENRY LANDAU ZEPH LANDAU JAMES POMMERSHEIM AARON ABRAMS, SANDY GANZELL and ERIC ZASLOW. Optimal estimators for algorithmic applications.

[2] A.Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, RA - 3(3):249–265, 1987.

[3] Josep Aulinas. The slam problem: A survey.

[4] J. Blanco, J.L.; Gonzalez and J.A. Fernandez-Madrigal. Optimal filtering for non-parametric observation models: Applications to localization and slam. *The International Journal of Robotics Research (IJRR)*.

[5] Feng L. Borenstein, J. Gyrodometry: a new method for combining data from gyros and odometry in mobile robots.

[6] J. Borenstein1 and L. Feng. Umbmark: A method for measuring, comparing, and correcting dead-reckoning errors in mobile robots.

[7] R. Chatila and J.-P. Laumond. Position referencing and consistent world modeling for mobile robots. *IEEE International Conference on Robotics and Automation*, 1985.

[8] M. Csorba. Simultaneous localisation and map building. *PhD thesis*.

[9] Maxim Likhachev Dave Ferguson and Anthony Stentz. A guide to heuristic-based path planning.

[10] Sebastian Thrun Dieter Fox and Wolfram Burgard. Probabilistic robotics. *Textbook*.

[11] J. Hwang E. Wu, M. Diftler and J. Chladek. A fault tolerant joint drive system for the space shuttle remote manipulator system. *IEEE International Conference on Robotics and Automation.*, 1991.

[12] Kirkby S. Eklund, P.W. and S. Pollitt. A dynamic multi-source dijkstra' algorithm for vehicle routing.

[13] Simic M. Elbanhawi, M. Sampling-based robot motion planning: A review.

[14] S.P. Engelson and D.V. McDermott. Error correction in mobile robot map-learning. *ICRA. Proceedings of the international conference on robotics.*, RA - 3(3):2555–2560, 1992.

[15] Wolfram Burgard Sebastian Thrun. Frank Dellaert, Dieter Fox. Monte carlo localization for mobile robots. *Proc. of the IEEE International Conference on Robotics and Automation*, 1999.

[16] H. Durrant-Whyte G. Dissanayake and T. Bailey. A computationally efficient solution to the simultaneous localisation and map building (slam) problem. *ICRA Workshop: Mobile Robot Navigation and Mapping*, April 2000.

[17] A. D. Greer T. Fielding G. Feil G. R. Sutherland, I. Latour and P. Newhook. An image-guided magnetic resonance-compatible surgical robot. *Neurosurgery*, pages 286–290, 2008.

[18] Roland Geraerts. On experimental research in sampling-based motion planning.

[19] H. H. Gonzalez-Banos and J. C. Latombe. Navigation strategies for exploring indoor environments. *International Journal of Robotics Research*, 2002.

[20] M. Heerink J. Broekens and H. Rosendal. Assistive social robots in elderly care: a review. *Gerontechnology*, pages 8:94–103, 2009.

[21] Juan D. Tardos Jose Neira. Data association in stochastic mapping using the joint compatibility test. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*.

[22] B. Kuipers and Y.-T. Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Journal of Robotics and Autonomous Systems*, 8:4763, 1991.

[23] James J. Kuffner Jr. LaValle, S.M. Randomized kinodynamic planning.

[24] S.M. (2006) LaValle. Planning algorithms.

[25] Steven M. LaValle and James J. Kuffner Jr. Rapidly-exploring random trees: A new tool for path planning.

[26] H.F. Leonard, J.J.; Durrant-whyte. Simultaneous map building and localization for an autonomous mobile robot.

[27] Gabriele Ligorio and Angelo Maria Sabatini. Extended kalman filter-based methods for pose estimation using visual, inertial and magnetic sensors: Comparative analysis and performance evaluation. *Sensors 2013*.

[28] Steven Clark Hugh F. Durrant-Whyte M. W. M. Gamini Dissanayake, Paul Newman and M. Csorba. A solution to the simultaneous localization and map building (slam) problem.

[29] M. J. Mataric. A distributed model for mobile robot environment-learning and navigation. *also available as MIT AI Lab Tech Report AITR-1228*, January 1990.

[30] Stergios I. Roumeliotis Puneet Goel and Gaurav S. Sukhatme. Robot localization using relative and absolute position estimates.

[31] Sean Quinlan. Efficient distance computation between non-convex objects.

[32] M. Self R. Smith and P. Cheeseman. Estimating uncertain spatial relationships in robotics. *Autonomous Robot Vehnicles,*, page 167193, 1990.

[33] Emilio Frazzoli. Sertac Karaman. Sampling-based algorithms for optimal motion planning.

[34] A. Tapus and M. J. Mataric. Guest editorial: special issue on socially assistive robotics. *Autonomous Robots*, pages 24:121–122, 2008.

[35] C. A. Nelson S. M. Farritor X. Zhang, A. Lehman and D. Oleynikov. Cooperative robotic assistant for laparoscopic surgery: Cobrasurge. *International Conference on Intelligent Robots and Systems*.

[36] D. Chakrabarti W. Burgard Y. Liu, R. Emery and S. Thrun. Using em to learn 3d models with mobile robots. *In Proceedings of the International Conference on Machine Learning (ICML)*, 2001.

[37] M.Q.-H. Yangming Li, Shuai Li ; Quanjun Song ; Hai Liu ; Meng. Fast and robust data association using posterior based approximate joint compatibility test. *IEEE transactions on Industrial Informatics*.