FEATURE BASED ADAPTIVE MOTION MODEL FOR BETTER LOCALIZATION

by

Rohan Bhargava

Submitted in partial fulfillment of the
requirements for the degree of
Master of Computer Science

at

Dalhousie University
Halifax, Nova Scotia
October 2014

DALHOUSIE UNIVERSITY

FACULTY OF COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "FEATURE BASED ADAPTIVE MOTION MODEL FOR BETTER LOCALIZATION" by Rohan Bhargava in partial fulfillment of the requirements for the degree of Master of Computer Science.

Dated: October 1, 2014

Supervisors:

Dr. Thomas Trappenberg

Dr. Mae Sato

Readers:

Dr. Evangelos E. Milios

A. External

DALHOUSIE UNIVERSITY

DATE: October 1, 2014

AUTHOR: Rohan Bhargava

TITLE: FEATURE BASED ADAPTIVE MOTION MODEL FOR BETTER LOCALIZATION

DEPARTMENT OR SCHOOL: Faculty of Computer Science

DEGREE: M.C.Sc. CONVOCATION: October YEAR: 2014

Signature of Author

Table of Contents

List of Tables

List of Figures

Abstract

In the 21st century, we are moving ahead in making robots a ubiquitous part of our everyday life. The need for a robot to interact with the environment has become a necessity. The interaction with the world requires a sense of it's pose. Humans clearly are very good in having a sense of their location in the world around them. The same task for robots is very difficult due to the uncertainties in the movement, limitation in sensing of the environment and complexities in the environment itself. When we close our eyes and walk we have a good estimate of our location but the same can't be said for robots. Without the help of external sensors the problem of localization becomes difficult. Humans use their vestibular system to generate cues about their movement and update their position. The same can be done for robots by using acceleration, velocity or odometry as cues to a motion model.

The motion model can be represented as a distribution to account for uncertainties in the environment. The parameters to the model are typically static in the current implementation throughout the experiment. Previous work has shown that by having an online calibration method for the model has improved localization. The previous work provided a framework to build adaptive motion model and targeted land based robot and sensors.

The work presented here builds on the same framework to adapt motion models for Autonomous Underwater Vehicle. We present detailed results of the framework in a simulator. The work also proposes a method for motion estimation using side sonar images. This is used as a feedback to the motion model. We validate the motion estimation approach with real world datasets.

Acknowledgements

I would like to thank my colleagues at Hallab for their encouragement and support throughout my work. I would like to express my special appreciation and thanks to my supervisors Dr. Thomas Trappenberg and Dr. Mae Sato. I would like to thank them for encouraging my research and for allowing me to grow as a research scientist. A special thanks to my family for all the sacrifices they have made for me to pursue my research interests. Last but certainly not the least my deepest gratitude goes to my late grandfather, Dr. D.N. Bhargava. His words of wisdom and encouragement from above has a great impact on what I am. I dedicate my thesis to him. Thank you

Chapter 1

Introduction

1.1 Motivation

The core of human environment interaction is the ability of a person to know its position in a the surrounding environment. The process of estimating the robot's position and orientation in the world is termed Localization [34]. As humans we use our visual and vestibular system to know our surroundings and localize ourselves. A common approach to determine the location of a robot is through the use of a Global Positioning System (GPS), which uses a series of satellites in low earth orbit that use differential positioning to determine a location for a receiver. Another approach is to provide a prior map of the environment and with the help of sensors a robot perceives the world and localizes itself in it. These techniques are highly dependant upon external sensors.

If we close our eyes and walk, we still have a sense of our movement. This is provided by our vestibular system and help us localize ourselves. These localization estimates can get noisy with time. Similarly in robots the major challenge is to localize without the help of external sensors. Sometimes sensing an external environment can be impossible or incomplete due to unreliability and inaccessibility of the sensors. In this case the localization estimate can be updated using self-generated cues such as acceleration and velocity.

The model which predicts localization based on just acceleration, velocity or odometry is called the motion model. In my thesis I propose an algorithm to learn and adapt the motion model with time. It can also be termed an online self calibration problem.

A need for a precise motion model is seen in Autonomous Underwater Vehicle(AUV). The AUV's are primarily dependant upon Inertial Navigation Systems (INS) which gives an estimate of the velocity, position and orientation. An example of such a vehicle is Hugin 4500 (Figure 1.1). The absence of external sensors such as

GPS gives rise to the need of having an adaptive motion model.

Input to a motion model such as velocity, acceleration, odometry have uncertainties associated with them. These uncertainties can arise due to noisy and incomplete sensing of the environment, uncertain movements of a robot in the environment and changes in the environment itself. To address these issues the motion and sensor models are represented probabilistically [34]. This helps in having a probability distribution over a space of guesses instead of relying on a single best guess. To update the state of a robot probabilistically there are algorithms such as Kalman Filter [20], Particle Filter [14] which are based on Bayes filter.

The motion and sensors models are represented by a distribution which is defined by its parameters. The process of determining parameters to kinematic model is termed as calibration [5] [38]. Generally the parameters to the models are hand tuned and are derived by conducting calibration experiments. Such calibration methods are impractical for two reasons. Firstly these processes are labour intensive and require prior information about the environment and robot. Secondly, changes in robot (e.g.-: general wear and tear) and environment (e.g.-: moving from fresh water to sea water) leads to changes in the parameters.

The changes in underwater environments such as change in density or temperature of water require recalibration of a robot while it is in operation. In such cases manual recalibration is not possible. The inaccuracies in the models will affect results for higher level tasks such as path planning [23] and Simultaneous Localization and Mapping (SLAM) [34] [15]. The need for online recalibration requires an adaptive motion model and the work presented in my thesis address this problem.

1.2 Previous Work

In this section I give an overview of the methods proposed for online calibration. Roy and Thrun [29] proposed an online calibration method for land robots which can be performed without human intervention. They approached the calibration process as maximum likelihood estimation problem which gives an estimate of parameters for the given data. The calibration parameters are iteratively estimated by comparing sensor readings. The algorithm proposed worked well for systematic drifts and the results showed the position error reduced approximately by 83%.

Figure 1.1: Hugin 4500 autonomous underwater vehicles. When submerged, the vehicle uses dead reckoning, incorporating Doppler velocity log and compass input to maintain an estimate on current positioning.

Alizar and Parr [10] continued the work further by estimating non-systematic drifts. They proposed an algorithm to learn the right parameters of a motion model for a land robot using their Expectation Maximization(EM) Framework [7]. This is an unsupervised machine learning technique that alternates between the expectation and maximization step. In the Expectation Step it creates an expectation of the log-likelihood using the current estimate of the parameters. In the Maximization step the parameters are computed by maximizing the expected log-likelihood found in the Expectation Step. With very little user input accurate motion models were learnt.

Yap [39] took Alizar and Parr's [10] work further and proposed an algorithm to learn the motion and sensor models for land robots. They used the same EM framework to learn the right parameters for the models. To calculate the likely trajectory of a robot both the algorithms implemented particle filtering [27] [4] and smoothing [8]. The algorithm started with an estimate of initial parameters and iteratively optimized the parameters based on the data collected during robot's operation. The work was directed towards land based robots.

The INS systems present in AUV suffer from drift i.e. small errors in measurement of acceleration and angular velocity that are integrated into progressively larger error.

To compensate for drift, systems such as Doppler Velocity Log (DVL), surface GPS etc. are used [22] [24]. Hegrenaes et.ãl [16] pointed out that there are situations where these systems fail or readings from these sensors need to be discarded due to poor quality. An example of such a situation is the non-feasibility of the vehicle to surface. In such situations they proposed to use the self-generated velocity estimates to aid INS systems. This shows the need to have good estimates from our motion model which can be used to aid INS systems.

1.3 Contribution

In my thesis I use the same framework proposed by Yap for adaptive motion model. The expectation maximization framework is used to learn a velocity based motion model. We provide a detailed study of the adaptive motion model. The changes in the environment are simulated by changing the parameters to the motion model. The performance of adaptive and static are compared with change in parameters as well as when drift is present in the system. The estimated parameter values are also compared to ground truth parameter values which were not done in any of the work before. In the second section of the thesis I propose an algorithm to estimate motion using side sonar images. This is the first step towards building adaptive motion model for AUV.

The remainder of the thesis is structured as follows. Chapter 2 will explain the motion model for AUV as well as give an insight on particle filtering and smoothing. Chapter 3 will give an overview of EM and show how this framework is used to adapt parameters for a motion model. Chapter 4 consists results of a simulated experiment to show the effectiveness of the algorithm. Chapter 5 explains how landmarks are extracted from side sonar images. The reliability of the landmarks are shown by extracting motion information. The algorithm to perform that is described in the same chapter. The results of comparing motion estimation using side sonar images to Doppler velocity log (DVL) is shown in Chapter 6.

Chapter 2

Background

In the chapter we start by discussing how motion models are probabilistically represented as well as give insight about motion models for AUV. This helps us in understanding, how motion model captures the probabilistic movements of robots. We then discuss a probabilistic state estimation algorithm such as particle filter [27] [4] which is at the heart of my algorithm as well as many other robotics systems. Lastly we discuss about particle smoothing [8] [9] which gives an estimate of ground truth by calculating the distribution of past states with taking into account all the evidence to date.

2.1 Motion Models

A motion model captures the relationship between the control input and the change in the robot's pose. Thrun, Burgard and Fox [34] models the motion of a robot probabilistically because the same control inputs will not reproduce identical motion every time. A good motion model will capture the errors such as drift that are encountered during the motion of a robot. The motion model is a necessary ingredient of many algorithms such as localization, mapping etc.

Let $X = (x, y, \theta)$ be the initial pose of the robot in x-y space. Mathematically the motion model can be described as $P(X^{'}|X, u)$, where $X^{'}$ is the pose after executing the motion command u. Based on the control input Thrun et al. [34] divided the motion model in two classes

1. **Odometry based motion model** -:
   This class of motion models are used for robots equipped with wheel encoders. The encoders gives us an estimate of the distance moved by a robot. This data is more accurate than velocity.

2. **Velocity based motion model** -:

These models calculate the new position based on velocities and time elapsed. These models are implemented for Autonomous Underwater Vehicle(AUV) and Unmanned Aerial Vehicles(UAV).

Both odometry as well as velocity are subject to drift and slippage therefore the same control commands will not generally reproduce the same motion.

The velocity motion model proposed by Thrun et al. [34] assumes that the wheeled robot can be controlled through two velocities, rotational and translational velocity. The translational velocity at time t is denoted by v_t and rotational velocity by w_t. Hence the control input u_t can be represented by

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

The assumption is that positive rotational velocities w_t induce a counterclockwise rotation whereas positive translational velocities v_t correspond to forward motion. The set of equations to compute the next state of a robot for a velocity motion model per unit time step are

$$x_t = x_{t-1} + v_{t-1}/w_{t-1}\sin(\theta_{t-1}) + v_{t-1}/w_{t-1}\cos(\theta_{t-1} + w_{t-1}\delta t) \tag{2.1}$$

$$y_t = y_{t-1} + v_{t-1}/w_{t-1}\cos(\theta_{t-1}) - v_{t-1}/w_{t-1}\sin(\theta_{t-1} + w_{t-1}\delta t) \tag{2.2}$$

$$\theta_t = \theta_{t-1} + w_{t-1}\delta t \tag{2.3}$$

To represent AUV's motion, 6 independent coordinates are necessary to determine the position and orientation of the rigid body. The notations used for marine vehicles are described in Table 2.1.

The pose of AUV can be represented as $s = (x, y, z, \theta, \phi, \psi)$. The first three coordinates correspond to the position of an AUV along the x, y, z axes while the last three coordinates describe the orientation commonly termed as roll, pitch and yaw. Fossen [12] in his book describes the motion of a marine vehicle in 6 DOF using two coordinate systems as shown in Figure 2.1. X_0, Y_0, Z_0 represent the

DOF		forces and moments	linear and angular vel.	positions and Euler angles
1	motions in the x-direction (surge)	X	u	x
2	motions in the y-direction (sway)	Y	v	y
3	motions in the z-direction (heave)	Z	w	z
4	rotation about the x-axis (roll)	K	p	ϕ
5	rotation about the y-axis (pitch)	M	q	θ
6	rotation about the z-axis (heave)	N	r	ψ

Table 2.1: Notation used for marine vehicles. Table from [12]

moving coordinate frame and is called the body-fixed reference frame. The earth-fixed reference frame is denoted by X, Y, Z. The origin of the body-reference frame is denoted by O and is chosen to coincide with the center of gravity denoted by CG.

To estimate the position of an AUV we need to calculate the velocity at which the AUV is currently moving. The velocity can be computed in two ways, static motion model or dynamic motion model.

In the static model the velocity is calculated from a lookup table as further explained below. In the dynamic model we are computing forces and moments on the fly but the parameters to these forces are considered to be static. The parameters such as density, temperature of water can change with time and lead to an inaccurate estimate of velocity in both the models. The change in density of the water can lead to changed in the trim of the AUV. Similarly, with temperature the buoyancy of the water can change. Hence the velocity needs to be adapted and Chapter 3.3 explains how it is done in my algorithm. A detailed description on how the velocity is calculated in both the models is explained in rest of the chapter.

Hegrenaes [17] points out that a way to implement a simple static motion model

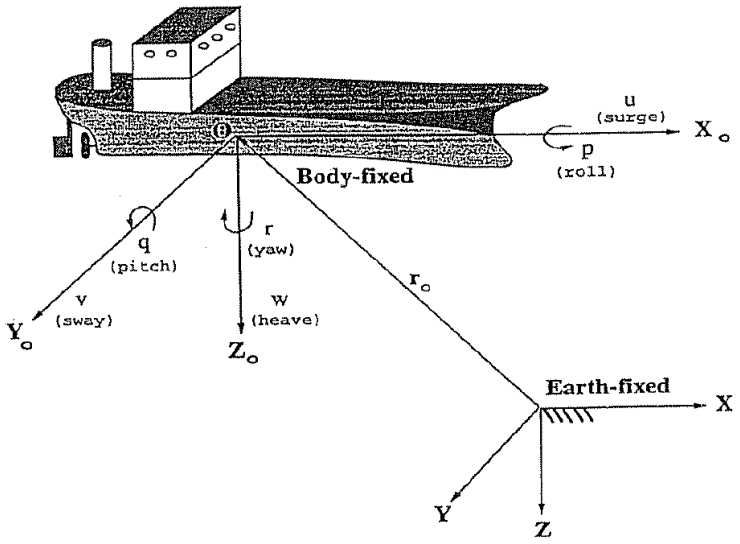


Figure 2.1: The coordinate system of an AUV described in Body-fixed and earth-fixed reference frames. Figure from [12]

as table look-up based on experimental data.

$$u_r = f(n_s) \tag{2.4}$$

u_r, n_s are the water relative linear velocity in x direction and control system set point respectively. In a similar manner an expression can be established for v_r.

Another way to implement the motion model is through dynamics. The equation to implement the motion model is [12]

$$M_{RB}\mathcal{V}^{\cdot} + C_{RB}(\mathcal{V})\mathcal{V} = \tau_{RB} \tag{2.5}$$

Here $\mathcal{V} = [u, v, w, p, q, r]^T$ is the body fixed linear and angular velocity and $\tau_{RB} = [X, Y, Z, K, M, N]$ is the vector of external forces and moments. M_{RB} is the rigid body inertia matrix and C_{RB} is Coriolis and centripetal matrix.

The right hand side of the vector 2.5 represents the external forces and moments acting on the vehicle. Fossen [12] classifies the forces into 1) Radiation-induced forces 2) Environmental Forces 3) Propulsion Forces. Table 2.2 shows the examples of various forces acting on an AUV.

τ_{RB} can be represented as the sum of these forces.

$$\tau_{RB} = \tau_H + \tau_E + \tau \tag{2.6}$$

Radiation Induced forces	Added Inertia, Hydrodynamic damping, Restoring Force
Environmental Forces	Ocean currents, Waves, Wind
Propulsion Force	Thruster/ Propeller Force, Control surface/ rudder force

Table 2.2: Examples of forces acting on an AUV. Table taken from [12]

Here τ_H is the radiation induced forces and moments, τ_E is used to describe the environmental forces and moments and τ is the propulsion forces and moments. Equations 2.5 and equation 2.6 can be combined to yield the following representation of 6 DOF dynamic equations of motion:

$$M\mathcal{V}^{\cdot} + C(\mathcal{V})\mathcal{V} + D(\mathcal{V})\mathcal{V} + g(\eta) = \tau_E + \tau \tag{2.7}$$

where

$M \triangleq M_{RB} + M_A$; $C(\mathcal{V}) \triangleq C_{RB}(\mathcal{V} + C_A(\mathcal{V})$

M_A is the added inertia matrix $C_A(\mathcal{V})$ is the matrix of hydrodynamic Coriolis and centripetal terms. $g(\eta)$ is the restoring force. Equation 2.8 can be used to calculate acceleration from the set of dynamic motion equations. Lammas [22] terms it as a navigation equation of the underwater vehicle.

$$\mathcal{V}^{\cdot} = M^{-1}(\tau - C(\mathcal{V})\mathcal{V} - D(\mathcal{V})\mathcal{V} - g(\eta)) \tag{2.8}$$

The parameters of matrices M and C are determined experimentally. Using the navigation equation we can determine the position of the AUV.

2.2 Particle Filter

Particle filters is an integral part of my algorithm to learn the right parameters of a motion model and the way it is used is explained in rest of the section. A particle Filter is a state estimation algorithm based on a sampling method for approximating a distribution. Thrun [34] defines particle filter as an alternative non-parametric implementation of the Bayes filter. It also can be called as a Sequential Monte Carlo(SMC) algorithm. The first attempt to use SMC was seen in simulations of growing polymers by M.N Rosenbluth and A.W. Rosenbluth [28]. Gordon et.ãl. [14] provided the first true implementation of sequential Monte Carlo algorithm.

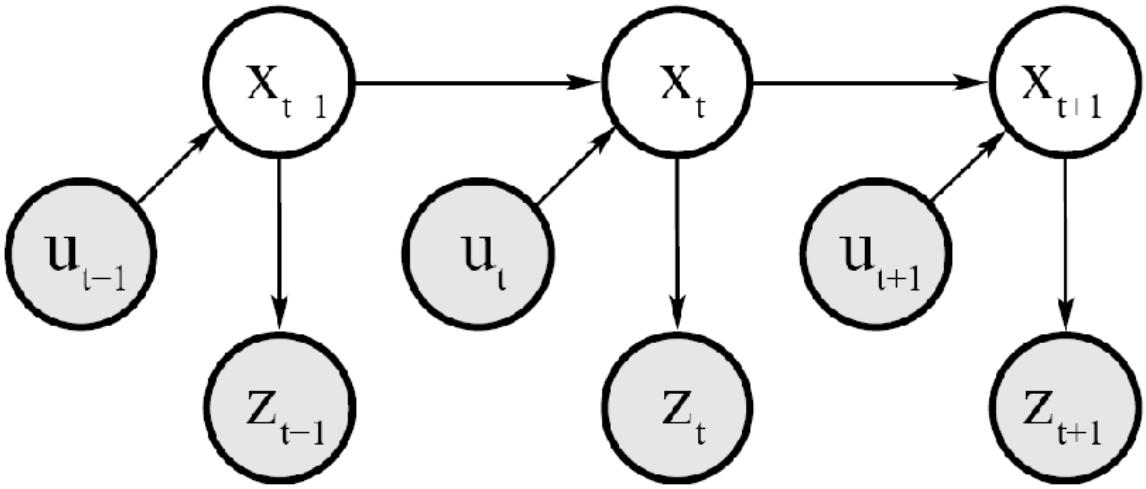


Figure 2.2: A temporal Bayesian model with hidden states x_t, observations z_t and controls u_t. Figure taken from [34]

Thrun [34] stated that the key idea behind particle filter is to represent the posterior bel(x_t) by a set of random state samples drawn from this posterior. Instead of representing the distribution by a parametric form particle filter represents a distribution by a set of samples drawn from this distribution. The representation is a approximation but it is nonparametric and therefore there are advantages of using particle filters as an alternative to Extended Kalman Filter and Unscented Kalman Filter. Particle Filters can represent a broader space of distributions for example non-Gaussian and can model non linear transformations of random variables.

The objective of particle filters is to estimate the state of the system given the observation variables. They are designed for Hidden Markov Models(Fig 2.2), where the system consists of hidden and observed variables. In this model the state x_t is the hidden random variable as it is not directly observed. The state at time t is only dependent upon the state at time $t-1$ and external influences such as control u_t. The measurement z_t depends on the state at time t. The knowledge about the influence of the control on the system can be used to calculate a new expected location and the measurement can be combined in a Bayesian way.

The algorithm for particle filters is described below-:

Input: X_{t-1}: particle set

u_t: most recent control

z_t: most recent measurement

Output: X_t:particle set

begin

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

    sample $x_t^m \sim p(x_t|u_t, x_{t-1}^m)$

    $w_t^m = p(z_t|x_t^m)$

    $X_t^- = X_t^- + (x_t^m, w_t^m)$

  end

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

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

    add $x_t^{[i]}$ to X_t

  end

  return X_t

end

Algorithm 1: Particle Filter Algorithm. x_t^m is instantiation of the state at time t. X_t^- is a temporary particle set. M is the number of particles. Algorithm taken from [34]

In Algorithm 1 each particle x_t^m is instantiation of the state at time t. The first step is to generate a hypothetical state x_t^m for time t based on previous state x_{t-1}^m and control u_t. The particles are samples from the state transition distribution $p(x_t|u_t, x_{t-1})$. The importance factor for each particle x_t^m is calculated and denoted by w_t^m. Importance factor is defined as the probability of measurement z_t under the particle x_t^m. This probability is defined by a sensor model $p(z_t|x_t)$, thus importance factor are used to incorporate the measurements into the particle set. In practice, the number of particles used are on the order of thousands.

The key part of the algorithm is the re-sampling step in particle filter algorithm. The algorithm draws M particles with replacement from a temporary particle set X_t^-. The probability of drawing the particles is given by the importance factor. By re-sampling, particle set is refocussed to regions in state space with higher probability.

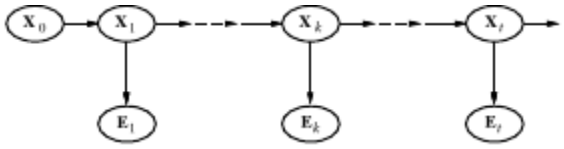


Figure 2.3: Smoothing computes $P(X_k|e_{1:t})$, the posterior distribution of the state at some past time k given a complete sequence of observations from 1 to t. Figure taken from [30]

.

2.3 Particle smoothing

The particle filter algorithm as described before is the first step in the Expectation process. The next algorithm that completes the Expectation Step is the particle smoothing. Doucet [9] in his paper stated that filtering based on observations received up to the current time is used to estimate the distribution of the current state of an Hidden Markov Model (HMM) whereas smoothing is used to estimate distribution of state at a particular time given all the observations up to some later time (Figure 2.3). Russell and Norvig [30] showed that the state of the system is better estimated by smoothing as it incorporates more information than just filtering. We use particle smoothing algorithm proposed by Yap and Shelton [39] which was based on the technique presented by Docuet et.ãl [8] and Godsill et.ãl [13].

Particle smoothing is carried out in order to generate samples from the entire joint smoothing density $p(x_{0:T}|u_{1:T}, z_{1:T})$. The equations described by Yap [39] are

$$p(x_{0:T}|u_{1:T}, z_{1:T}) = \prod_{t=0}^{T} p(x_t|x_{t+1:T}, u_{1:T}, z_{1:T}) \tag{2.9}$$

where,

$$\begin{aligned} p(x_t|x_{t+1:T}, u_{1:T}, z_{1:T}) &= p(x_t|x_{t+1}, u_{1:t+1}, z_{1:t}) && (2.10)\\ &= \frac{p(x_{t+1}|x_t, u_{1:t+1}, z_{1:t})p(x_t|u_{1:t+1}, z_{1:t})}{p(x_{t+1}|u_{1:t+1}, z_{1:t})} && (2.11)\\ &= \frac{p(x_{t+1}|x_t, u_{t+1})p(x_t|u_{1:t}, z_{1:t})}{p(x_{t+1}|u_{1:t+1}, z_{1:t})} && (2.12)\\ \alpha\ & p(x_{t+1}|x_t, u_{t+1})p(x_t|u_{1:t}, z_{1:t}) && (2.13) \end{aligned}$$

Equation 2.13 is used to generate states backwards in time given the future states.

$p(x_{t+1}|x_t, u_{t+1})$ is the state transition probability and $p(x_t|u_{1:t}, z_{1:t})$ is obtained by performing particle filtering.

Algorithm 2 shows the step involved to sample from the entire joint smoothing density.

Input: $X_t, t = 0, 1, ..., T$: particle approximations to the posterior pdfs $p(x_t|c_{1:t}, s_{1:t})$

$c_{1:T} = (c_1, c_2, ..., c_T)$: set of controls from time 1 to time T

Output: $x'_{0:T} = (x'_0, x'_1, ..., x'_T)$: a sample from the entire joint smoothing density $p(x_{0:T}|c_{1:T}, s_{1:T})$

begin

  draw i with probability $\propto w_T^{[i]}$ $x'_T \leftarrow x_T^{[i]}$

  for $t \leftarrow T-1$ *down to 0 do* **do**

    for $i \leftarrow 1$ *to* N_s *do* **do**

      $w_{t|t+1}^{[i]} \leftarrow w_t^{[i]} p(x'_{t+1}|x_t^{[i]}, u_{t+1})$

    end

    draw i with probability $\propto w_{t|t+1}^{[i]}$

    $x' \leftarrow x_t^{[i]}$

  end

end

Algorithm 2: Sample the entire joint smoothing density $p(x_{0:T}|c_{1:T}, s_{1:T})$

In the first step of the algorithm a particle is drawn with probability proportional to the filtered weight of the particles. The next step is to move a time step back and modify the weights of the particles by calculating the new smoothed weights. The new smoothed weights are the product of state transition probability $p(x'_{t+1}|x_t^{[i]}, u_{t+1})$ and the weight of the particle $w_t^{[i]}$. The next step in the algorithm is to draw particles with probability proportional to new smoothed weights $w_{t|t+1}^{[i]}$. The sequence of particles x' drawn from joint smoothing density $p(x_{0:T}|c_{1:T}, s_{1:T})$ from time 0 to T, form a sampled trajectory $x'_{0:T} \triangleq (x'_0, x'_1, ..., x'_T)$.

Chapter 3

Learning the motion model

3.1 Previous Work

There is an extensive literature on different methods to calibrate a robot (eg-: [5] [38]). All the methods discussed before Roy's work [29] required human intervention and assumed the world to be static. These methods required a human to have experience and a device to measure the ground truth. The most important assumption that these methods made was that the configuration of a robot never changed and operated in a static environment.

Roy and Thrun first proposed an online self-calibration method [29] in 1999 that adapted to changes that occurred during the lifetime of a robot. The algorithm was designed for land robots where the final pose was given by

$$x^{'} = x + Dcos(\theta + T) \tag{3.1}$$

$$y^{'} = y + Dsin(\theta + T) \tag{3.2}$$

$$\theta^{'} = (\theta + T) mod 2\pi \tag{3.3}$$

D and T are the true translational and rotation of a robot. The measured translational and rotational is d and t and if robot's odometry is accurate then $D = d$ and $T = t$. In practice there is a difference and Roy represents D and T by equations 3.4 and 3.5 respectively.

$$D = d + \sigma_{trans} d + \epsilon_{trans} \tag{3.4}$$

$$T = t + \sigma_{rot} d + \epsilon_{rot} \tag{3.5}$$

ϵ_{trans} and ϵ_{rot} are the random variables with zero mean. σ_{trans} and σ_{rot} denote the systematic errors in the system. An example of such an error is drift. The systematic errors referred here stay almost constant over a prolonged period of time.

The algorithm proposed by Roy and Thrun aimed at estimating σ_{trans} and σ_{rot} using sensor data collected throughout robot's motion. They treat the problem as maximum likelihood estimation problem where the parameters are estimated under a dataset z as described in equation 3.6.

$$(\sigma^*_{trans}, \sigma^*_{rot}) = argmaxP(\sigma_{trans}, \sigma_{rot}|z) \tag{3.6}$$

Austin and Eliazar [10] proposed a different method to achieve the same goals proposed by Roy and Thrun. Their algorithm was different for two reasons. Firstly, Austin ad Eliazar used a more general model which incorporated independence of motion terms. Secondly, the method was able to estimate parameters for non systematic errors as well. The motion model proposed by them to account for the discretization error is -:

$$\begin{aligned} x^{'} &= x + Dcos(\theta + T/2) \\ y^{'} &= y + Dsin(\theta + T/2) \\ \theta^{'} &= (\theta + T/2) mod 2\pi \end{aligned} \tag{3.7}$$

As the turn and drive commands are performed independently therefore to not violate this assumption this model makes T reasonably small and it is absorbed as part of noise. To estimate non-systematic errors the true translational(D) and rotation(T) are represented by normal distribution with mean d and t and the variance will scale with d^2 and t^2 as shown in equation 3.8

$$\begin{aligned} D &\sim \mathcal{N}(d\mu_{D_d} + t\mu_{D_t}, d^2\sigma^2_{D_d} + t^2\sigma^2_{D_t}) \\ T &\sim \mathcal{N}(d\mu_{T_d} + t\mu_{T_t}, d^2\sigma^2_{T_d} + t^2\sigma^2_{T_t}) \end{aligned} \tag{3.8}$$

where μ_{A_b} is the coefficient for the contribution of odometry term b to the mean of the distribution over A. The algorithm is used to learn these set of mean and variances.

The method by Austin an Eliazar uses the EM framework to learn the parameters of the motion model for land robots. In the E step, particle filtering and smoothing were performed to get a set of trajectories. In M step, the maximum likelihood values of parameters given the trajectories was calculated.

Teddy Yap [39] used the same framework to learn parameters for motion model and sensor model (Figure 3.1). They adopted the same motion model but with slightly different noise model as shown in equation 3.9.

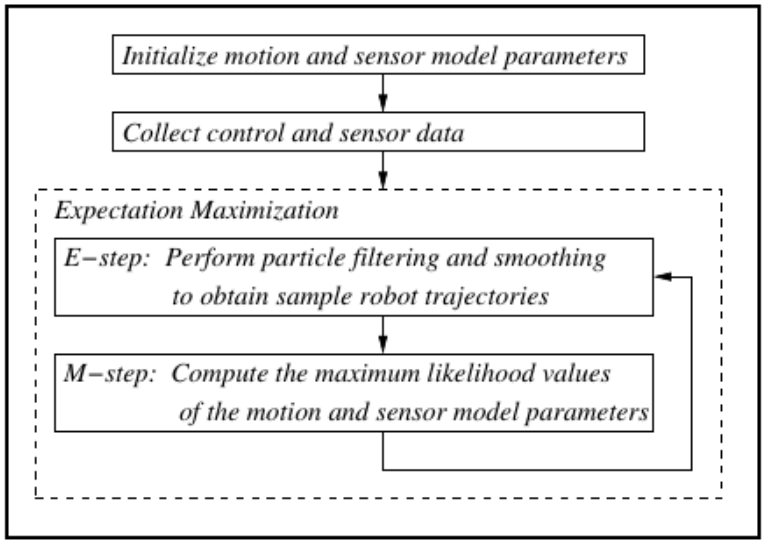


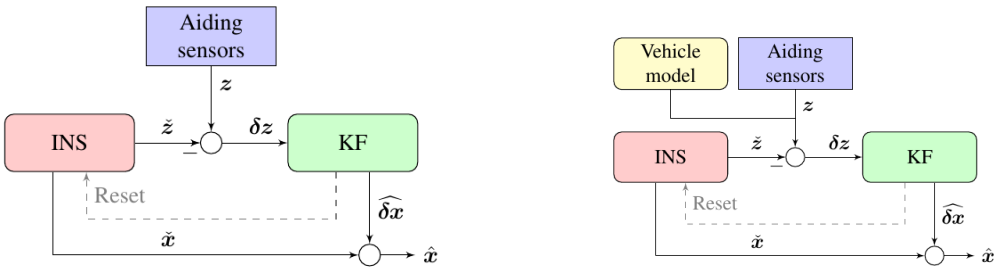
Figure 3.1: Block Diagram for the parameter estimation framework. Figure taken from [39].

$$\begin{aligned} D \sim &\quad \mathcal{N}(d, d^2\sigma_{D_d}^2 + t^2\sigma_{D_t}^2 + \sigma_{D_1}^2) \\ T \sim &\quad \mathcal{N}(t, t^2\sigma_{T_d}^2 + t^2\sigma_{T_t}^2 + \sigma_{T_1}^2) \end{aligned} \tag{3.9}$$

The extra constant terms σ_{D_1} and σ_{T_1} are added to account for the errors that are not proportional to the translation or rotation of a robot.

Hegrenaes [16] in his work showed the importance of motion model for navigation in underwater vehicle. They proposed a novel approach for navigation systems in which knowledge about the vehicle dynamics was used to aid the Inertial Navigation System(INS). The new navigation system was tested on real dataset collected by an AUV.

For navigation in AUV's the velocity of the vehicle needs to be estimated for which sensors such as IMU, DVL are used. In a traditional INS system the key component is an IMU and a set of navigation equations. The readings from the accelerometers and gyroscopes are integrated to get an estimate of velocity, position and orientation. The reading from such sensors have inherent errors and leads to drift in the INS system. Generally sensors such as surface GPS, DVL etc. are an aiding system to the INS (Figure 3.2(a)). Combination of such systems leads to better estimate of velocity, position and orientation [24].



(a) Traditional aided INS

(b) Model-aided INS

Figure 3.2: High level system outline. The vehicle model can be used in parallel to external aiding sensors. Figure taken from [16]

Hergreanaes [16] points out alternative velocity information such as velocity estimate through vehicle dynamics is required because there are situations where it is not possible for the AUV to surface and get a GPS reading or DVL measurement needs to be discarded due to poor quality. The high level system outline for such a model is shown in Figure 3.2(b).

The system is very similar to traditional INS except that the vehicle model output in also integrated to the system. The vehicle model output doesn't require any extra instruments therefore can be easily applied to any vehicle. An alternative velocity estimate aids the INS where DVL readings are lacking as well as gives redundancy to the system.

All the above calibration methods are designed for odometric based motion models and for land robots. The use of vehicle model to aid the INS for navigation purposes shows us the importance of an adaptive motion model. The algorithm that I propose is for underwater vehicles and velocity based motion model. The process to adapt the motion model is similar to the previous work by Yap and Eliazar and our approach is described in rest of the chapters.

3.2 General Architecture of new system

In this section we give an overview of the system and point out the differences in my system as compared to the existing system proposed by Yap [39]. Figure 3.3 is an

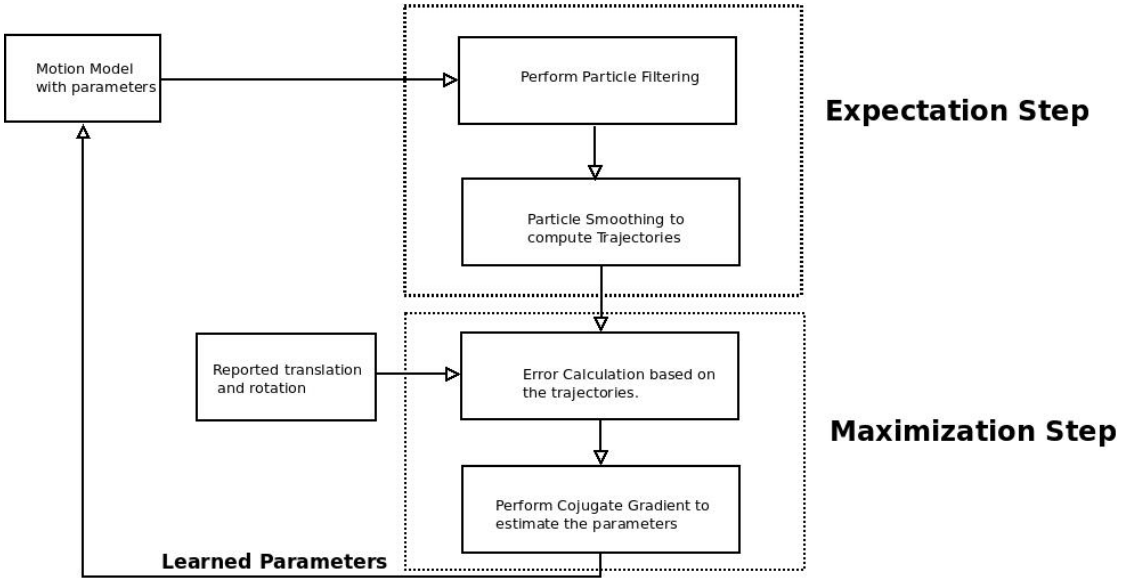


Figure 3.3: Block diagram for the algorithm proposed to adapt motion model.

outline of the proposed system. The motion and sensor models are initialized with a set of parameters. Given the motion model $p(x_t|x_{t-1}, u_t)$, sensor model $p(z_t|x_t)$ and the pose $p(x_{t-1})$ of a robot we can perform particle filtering using Algorithm 1. It is performed to estimate the pose $p(x_t)$ of a robot at the next time step.

The key ingredient to learn the motion model is to have an estimate of the movement of a robot. Particle smoothing (Algorithm 2) is performed on the particle set produced by particle filtering to get an idea of ground truth. The algorithm can be repeated several times to get a set of trajectories. Particle filtering and smoothing are the key algorithm for the Expectation Step.

The reported translational and rotational movement of a robot is recorded for every time step. Based on the trajectories and reported movements the errors are calculated at each time step. After we have a set of errors, we perform Newton Conjugate gradient [33] on the error function to estimate parameters. Conjugate gradient is an iterative method to solve large systems of linear equations [33]. This completes the Maximization Step.

The learned parameters are reassigned to the motion model and helps in adapting our model to changes in robot and the environment. The whole algorithm is repeated at every time step so that we can dynamically learn the right parameters.

The system proposed for parameter estimation is similar to the system by Yap [39].

I use the same EM framework and conjugate gradient to learn the right parameters for the model. The difference lies that the motion model learned is a velocity based motion model as compared to odometry based model. Secondly we use side sonar images to calculate landmarks on the fly for underwater environments. Therefore the algorithm doesn't rely on static maps for reference points.

3.3 Adapting Motion Model in proposed system

3.3.1 Expectation Maximization

EM is an iterative process of finding maximum likelihood of parameters of a model which depend upon hidden variables. It is used to estimate unknown parameters θ given the observed data X. Z is the non-observed (hidden, latent) variable. In practice a complete dataset is not given and only a set of observations or incomplete dataset X is found. The hidden variables are important to a problem but complicate the learning process [30]. In order to learn with hidden variables Dempster et.ãl [7] proposed a method to maximize the probability of the parameters θ given the dataset X with hidden variables Z that he called as EM.

$$\theta^* = argmax_\theta \int p(\theta, Z|X)dZ \tag{3.10}$$

The key idea behind the EM algorithm is to alternate between estimating parameters θ and hidden variable Z. The E step consists of finding the posterior distribution of hidden variables $p(Z|X, \theta^{old})$ given the current estimate of parameters θ^{old}. The posterior distribution is used to find expectation of the data as shown in equation 3.11.

$$\mathcal{Q}(\theta, \theta^{old}) = \sum_Z p(Z|X, \theta^{old}) \ln p(X, Z|\theta) \tag{3.11}$$

In the M step we maximize the function as shown in equation 3.12 to estimate the new parameters θ^{new}.

$$\theta^{new} = argmax_{theta}\mathcal{Q}(\theta, \theta^{old}) \tag{3.12}$$

The objective as described by Minka [26] is to maximize $\mathcal{Q}(\theta, \theta^{old})$ and we want an updated estimate of θ^{new} such that

$$\theta^{new} > \theta^{old} \tag{3.13}$$

or we want to maximize the difference,

$$\mathcal{Q}(\theta^{new}) - \mathcal{Q}(\theta^{old}) = \ln P(X|\theta^{new}) - \ln P(X|\theta^{old}) \tag{3.14}$$

The above equations with hidden variables can be written as

$$\mathcal{Q}(\theta^{new}) - \mathcal{Q}(\theta^{old}) = \ln(\sum P(X|z,\theta^{new})P(Z|\theta^{new})) - \ln P(X|\theta^{old}) \tag{3.15}$$

Using Jensen's Equality it can be show that,

$$\ln \sum_{i=1}^{n} \lambda_i x_i \geq \sum_{i=1}^{n} \lambda_i \ln(x_i) \tag{3.16}$$

for constants $\lambda_i \geq 0$ with $\sum_{i=1}^{n} \lambda_i = 1$.
This can be applied to equation 3.15 and $\lambda = P(Z|X,\theta^{old})$. As $P(Z|X,\theta^{old})$ is a probability measure we have $P(Z|X,\theta^{old}) \geq 0$ and $\sum_Z P(Z|X,\theta^{old}) = 1$.

$$Q(\theta^{new}) - Q(\theta^{old}) = \ln(\sum_{Z} P(X|Z,\theta^{n} \tag{3.17}$$

$$= \ln(\sum_{Z} P(X|Z,\theta^{new})P(Z|\theta).\frac{P(Z|X,\theta^{old})}{P(Z|X,\theta^{old})}) - \ln P(X|\theta^{old}) \tag{3.18}$$

$$= \ln(\sum_{Z} P(z|X,\theta^{old})\frac{P(X|Z,\theta^{new})P(Z|\theta^{new}}{P(Z|X,\theta^{old})}) - \ln P(X|\theta^{new}) \tag{3.19}$$

$$\geq \sum_{Z} P(Z|X,\theta^{old}) \ln(\frac{P(X|Z,\theta^{new})P(Z|\theta^{new})}{P(Z|X,\theta^{old})}) - \ln P(X|\theta^{old}) \tag{3.20}$$

$$= \sum_{Z} P(Z|X,\theta^{old}) \ln(\frac{P(X|Z,\theta^{new})P(Z|\theta^{new})}{P(Z|X,\theta^{old})P(X|\theta^{old})}) \tag{3.21}$$

$$\triangleq \triangle(\theta^{new}|\theta^{old}) \tag{3.22}$$

We continue by writing

$\mathcal{Q}(\theta^{new}) \geq \mathcal{Q}(\theta^{old}) + \triangle(\theta^{new}|\theta^{old})$

and for convenience define,

$q(\theta^{new}|\theta^{old}) \triangleq \mathcal{Q}(\theta^{old}) + \triangle(\theta^{new}|\theta^{old})$

The function $q(\theta^{new}|\theta^{old})$ is bounded by the likelihood functions $\mathcal{Q}(\theta^{new})$. We need to choose values of θ^{new} so that $\mathcal{Q}(\theta^{new})$ is maximized. The new updated value is denoted by θ_{n+1}.

$$\theta_{n+1} = argmax_\theta\{q(\theta^{new}|\theta^{old})\} \tag{3.23}$$

$$= argmax_\theta\{\mathcal{Q}(\theta^{old}) + \sum_Z P(Z|X,\theta^{old}) \ln(\frac{P(X|Z,\theta^{new})P(Z|\theta^{new})}{P(Z|X,\theta^{old})P(X|\theta^{old})}\} \tag{3.24}$$

$$= argmax_\theta\{\sum_Z P(Z|X,\theta^{old}) \ln(P(X|Z,\theta^{new})P(Z|\theta^{new})\} \tag{3.25}$$

$$= argmax_\theta\{\sum_Z P(Z|X,\theta^{old}) \ln P(X,Z|\theta^{new})\} \tag{3.26}$$

$$\tag{3.27}$$

We use the EM algorithm as described by Bishop[3] in his book. The steps taken in EM algorithm are described below -:

1. Have an initial estimate of the parameters θ^{old}
2. **E Step:** Evaluate $P(Z|X,\theta^{old})$
3. **M Step:** Evaluate $\theta^{new} = argmax_\theta L(\theta,\theta^{old})$

   where $L(\theta,\theta^{old}) = \Sigma_Z P(Z|X,\theta_{old}) \log P(X,Z|\theta)$
4. Check for the convergence of either the log likelihood or the parameter values. If the convergence criterion is not satisfied then let

   $\theta \leftarrow \theta^{new}$

   and return to step 2

In our algorithm we use Newton Conjugate Gradient to check for convergence and estimating the right parameters. Conjugate gradient is a method to determine the minimum or maximum of a function [33]. It is important to point that EM algorithm is local optimization technique and there are situations where it can get stuck in local optimum.

3.3.2 Parameter Estimation

In this section we give a detailed explanation of the method proposed to learn the motion model of a robot. The framework is similar to the one proposed by Yap [39]. As stated in Chapter 2.1 to calculate the position of an AUV we need to estimate

velocity and feed it to the navigation equation 2.8. As we are operating in a dynamic environment various factors can lead to changes in our motion model. To adapt the motion model we assume a Gaussian distribution to represent velocity. We assume it to be Gaussian as sum of several random noises leads to a such a distribution. Another assumption for the algorithm that we represent the pose of an AUV in two dimension as compared to six.

The velocity based motion model equations used are the following

$$x_t = x_{t-1} + V_{t-1}/W_{t-1}\sin(\theta_{t-1}) + V_{t-1}/W_{t-1}\cos(\theta_{t-1} + W_{t-1}\delta t) \tag{3.28}$$

$$y_t = y_{t-1} + V_{t-1}/W_{t-1}\cos(\theta_{t-1}) - V_{t-1}/W_{t-1}\sin(\theta_{t-1} + W_{t-1}\delta t) \tag{3.29}$$

$$\theta_t = \theta_{t-1} + W_{t-1}\delta t \tag{3.30}$$

where,

$$V \sim \mathcal{N}(v_t, v_t^2\sigma_{V_v}^2 + w_t^2\sigma_{V_w}^2 + \sigma_{V_1}^2) \tag{3.31}$$

$$W \sim \mathcal{N}(w_t, v_t^2\sigma_{W_v}^2 + w_t^2\sigma_{W_w}^2 + \sigma_{W_1}^2) \tag{3.32}$$

The translational(V) and rotational(W) velocity are represented by a Gaussian distribution. The mean of the distributions are the reported translational v and rotational w velocity respectively. σ_{V_1} and σ_{w_1} are added to the motion model to account for errors that are not directly proportional to the translation and rotation of a robot. The noise model is similar to the one proposed by Yap [39].

Putting the whole problem of estimating parameters in EM framework, we define the parameters of the motion model that we want to learn are $\theta = \sigma_{V_v}^2, \sigma_{V_w}^2, \sigma_{V_1}^2, \sigma_{W_v}^2, \sigma_{W_t}^2, \sigma_{W_1}^2$ The data X from which the parameters can be learned is defined as $X = u_{1:T}, z_{1:T}$ where, $u_{1:T}$ and $z_{1:T}$ are the history of control and sensor readings. The robot's trajectory is the hidden variable in the system as it not directly observable and can be defined as $Z = x_{0:T}$.

The first step in an EM algorithm is to initialize the set of parameters θ with some initial values. In the E step we calculate the expectation of $\log p(Z, X|\theta)$ with

respect to distribution $p(Z|X,\theta)$. The distribution can be represented by the entire join smoothing density $p(x_{0:T}|u_{1:T}, z_{1:T})$. To approximate the E step i.e. the joint smoothing density, particle filtering and smoothing is performed to calculate a set of robot trajectories as discussed in section 2.3. In the M step the maximum likelihood of parameters are computed by treating the set of trajectories as ground truth. The algorithm keeps on alternating between the E and M step until convergence.

For calculating the maximum likelihood values for parameters we need to calculate the motion errors $\epsilon_{V_t}, \epsilon_{W_t}$ based on robot trajectory and the contribution of translational v and rotational w velocity to the errors. The distribution that represents the errors are

$$\epsilon_{V_t}^{[j]} \sim \mathcal{N}(0, v_t^2\sigma_{V_v}^2 + w_t^2\sigma_{V_w}^2 + \sigma_{V_1}^2) \tag{3.33}$$

$$\epsilon_{W_t}^{[j]} \sim \mathcal{N}(0, v_t^2\sigma_{W_v}^2 + w_t^2\sigma_{W_w}^2 + \sigma_{W_1}^2) \tag{3.34}$$

where j is the index for each sampled trajectory.

The likelihood functions are

$$\mathcal{Q}_{\epsilon_D}(\sigma_{V_v}^2, \sigma_{V_r}^2, \sigma_{V_1}^2) = p(\epsilon_{V_v}^{[j]}|u_{1:T}, z_{0:T}^{[j]}) \tag{3.35}$$

$$= \prod_j \prod_{t=0}^{T-1} \frac{1}{\sqrt{2\pi(v_t^2\sigma_{V_v}^2 + r_t^2\sigma_{V_w}^2 + \sigma_{V_1}^2)}} * \exp(\frac{(\epsilon_{V_t}^{[j]})^2}{2(d_t^2\sigma_{V_v}^2 + w_t^2\sigma_{V_w}^2 + \sigma_{V_1}^2)} \tag{3.36}$$

$$\mathcal{Q}_{\epsilon_T}(\sigma_{W_d}^2, \sigma_{W_r}^2, \sigma_{W_1}^2) = p(\epsilon_{W_t}^{[j]}|u_{1:T}, z_{0:T}^{[j]})$$

$$= \prod_j \prod_{t=0}^{T-1} \frac{1}{\sqrt{2\pi(v_t^2\sigma_{W_v}^2 + w_t^2\sigma_{W_w}^2 + \sigma_{W_1}^2)}} * \exp(\frac{(\epsilon_{W_t}^{[j]})^2}{2(v_t^2\sigma_{W_v}^2 + w_t^2\sigma_{W_w}^2 + \sigma_{W_1}^2)})$$

The estimate the parameters we get the maximum likelihood estimates

$$\sigma_{V_v}^{2*}, \sigma_{V_r}^{2*}, \sigma_{V_1}^{2*} = argmax_{\sigma_{V_v}^2, \sigma_{V_r}^2, \sigma_{V_1}^2} \mathcal{Q}(\sigma_{V_v}^2, \sigma_{V_r}^2, \sigma_{V_1}^2) \tag{3.37}$$

$$\sigma_{W_d}^{2*}, \sigma_{W_r}^{2*}, \sigma_{W_1}^{2*} = argmax_{\sigma_{W_d}^2, \sigma_{W_r}^2, \sigma_{W_1}^2} \mathcal{Q}(\sigma_{W_d}^2, \sigma_{W_r}^2, \sigma_{W_1}^2) \tag{3.38}$$

We maximize the log likelihood function via Newton conjugate gradient method with respect to motion model parameters. In this method the gradient of the function

is taken as the first search direction while the next search direction are chosen in such a way that they are orthogonal to all previous search directions [33]. The gradient of log likelihood functions are

$$\mathcal{L}(\sigma^2_{V_v}, \sigma^2_{V_w}, \sigma^2_{V_1}) = -\frac{1}{2}\sum_j \sum_{t=0}^{T-1} [\log 2\pi + \log(v_t^2\sigma^2_{V_v} + w_t^2\sigma^2_{V_w} + \sigma^2_{V_1}) + \frac{(\epsilon^{[j]}_{V_t})^2}{v_t^2\sigma^2_{V_v} + w_t^2\sigma^2_{V_r} + \sigma^2_{V_1}}] \tag{3.39}$$

$$\mathcal{L}(\sigma^2_{W_v}, \sigma^2_{W_w}, \sigma^2_{W_1}) = -\frac{1}{2}\sum_j \sum_{t=0}^{T-1} [\log 2\pi + \log(v_t^2\sigma^2_{W_v} + w_t^2\sigma^2_{W_w} + \sigma^2_{W_1}) + \frac{(\epsilon^{[j]}_{W_t})^2}{v_t^2\sigma^2_{W_v} + w_t^2\sigma^2_{W_w} + \sigma^2_{W_1}}] \tag{3.40}$$

Chapter 4

Simulation setup and results

In this chapter we describe the experimental setup of the simulator to test our algorithm. To demonstrate the effectiveness of our approach the results from the simulated experiment are also shown.

4.1 Simulation setup

To test our algorithm for learning the motion model, we create a simulated world as shown in Figure 4.1. The world size of our simulation is 400X400 with four landmarks shown in blue dots. The red $*$ shows the location of the robot and blue triangle is the location estimate of the robot by the particle filter. The robot is moving with a constant forward and rotational velocity throughout the simulation. In the following simulation the robot can measure its distance from all four landmarks at all times. The sensor noise in the simulation was varied in different experiments. The particle size is 500 and is constant throughout the simulation. Table 4.1 shows the summary of variables for the simulation to test the effectiveness of our algorithm to learn a motion model.

	simulation
World Size	400 units X 400 units
Total timesteps	200
No. of sensor readings	200
Translational Velocity	3 units/timestep
Rotational Velocity	0.1 degrees/timestep

Table 4.1: Summary of the parameters of the simulation performed to adapt motion model.

At the start of the simulation the initial location of the robot and particle filters are randomly initialized. The robot is moved at a constant velocity and Figure 4.2 shows the updated location of the robot as well as an estimate by particle filters of

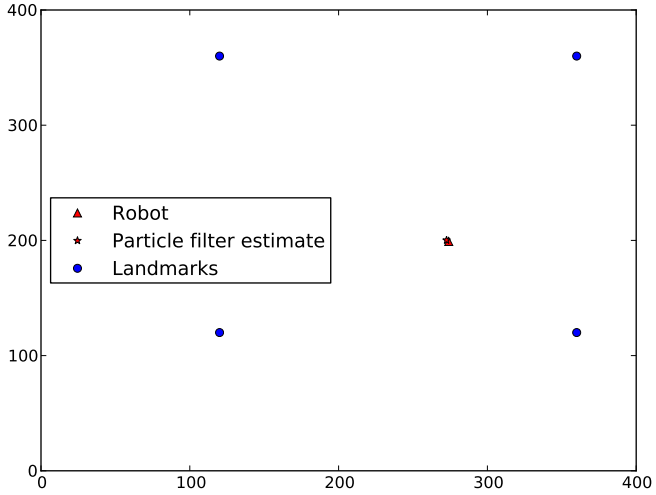


Figure 4.1: The simulation environment consisting of the robot(triangle) and the particle estimate of the location($*$). The blue dots represent the landmarks.

the robot's location.

4.2 Results

In this section I describe the various simulations that were performed in the simulated world. The external changes in the environment were simulated by changing the parameters to a robot's motion model. The parameters can be altered by artificially introducing a drift in the motion or by changing the variance of the motion model distribution. As shown in Table 4.1 the total timesteps is 200 and in all the simulations we simulate the changes at time step 60.

As described in Chapter 2.1, the noise model for our simulated experiment is

$$V_t \sim \mathcal{N}(v_t, v_t^2\sigma_{V_v}^2 + w_t^2\sigma_{V_w}^2 + \sigma_{V_1}^2) \tag{4.1}$$

$$W_t \sim \mathcal{N}(w_t, v_t^2\sigma_{w_v}^2 + w_t^2\sigma_{V_w}^2 + \sigma_{W_1}^2) \tag{4.2}$$

In the first simulation we change $\sigma_{V_v}^2$ at time step 60 as shown in Table 4.2. The changed values of the parameter for the first and second case are 0.5 and 1.0 respectively. The number of trajectories is 3 and is kept constant for both cases. In this simulation we keep the sensor noise constant as well. This simulation was used to mimic the change in the environment by changing the noise of the motion model.

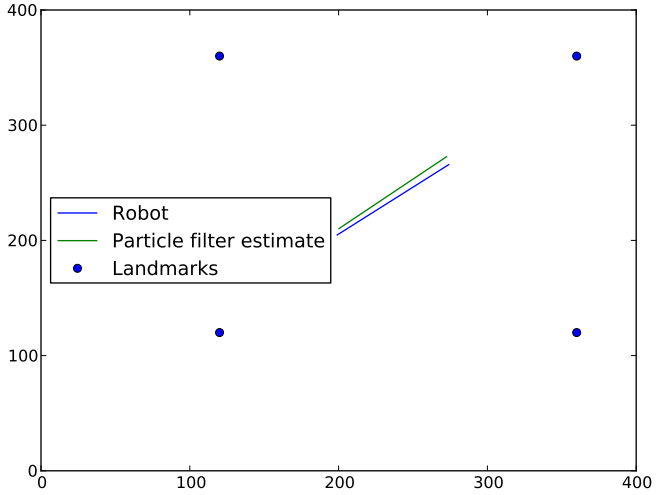


Figure 4.2: Particle Filters estimating the position of the robot. The robot is moved at a constant velocity of 3 units/timestep in each case. The particle filters estimate the location of the robot by integrating the motion and sensor models.

The change in environment doesn't affect the ability of the robot to sense the environment, therefore the sensor noise is kept constant throughout the simulation. I was looking for the performance of both the models when there is big change in the noise for example from 0.05 to 1.0.

Initial Parameter Values		Changed Parameter Values (after 60)		Estimated Parameter Values		Sensor Noise	Trajectories
		1	2	1	2		
$\sigma^2_{V_v}$	0.05	0.5	1.0	0.707	1.868	1.0	3
$\sigma^2_{V_w}$	0.05	0.05	0.05	0.05	0.05	1.0	3
$\sigma^2_{V_1}$	0.05	0.05	0.05	0.123	0.252	1.0	3
$\sigma^2_{W_v}$	0.05	0.05	0.05	1.476	1.496	1.0	3
$\sigma^2_{W_w}$	0.05	0.05	0.05	0.05	0.05	1.0	3
$\sigma^2_{W_1}$	0.05	0.05	0.05	0.208	0.210	1.0	3

Table 4.2: Initial and estimated values of parameters of a motion model with constant sensor noise and trajectories

As shown in Table 4.2 the estimated value of $\sigma^2_{V_v}$ are 0.707 and 1.868 which are greater than the actual values and could lead to better localization. In order to

demonstrate that, the localization error (euclidean error between the robot's actual position and location estimate of particle filters) is plotted for different times.

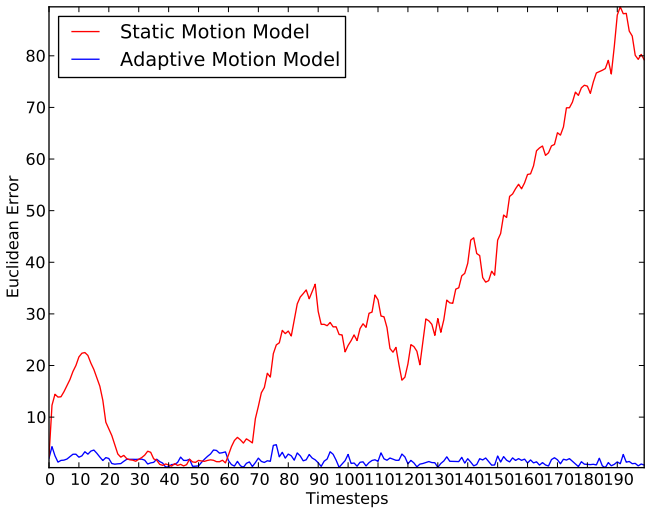


Figure 4.3: Plot showing the localization error between robot and the estimate of the robot's location by particle filters. The $\sigma^2_{V_v}$ is changed from 0.05 to 0.5 at timestep 60. The sensor noise is 1.0.

In Figure 4.3 the red and blue line shows the error with a static motion model and adaptive motion model respectively. At the start of the simulation the parameter values for the robot's motion model and static motion model for particle filters are initialized with the same value. The adaptive motion model starts estimating the parameters from time step 0. In Figure 4.3 in the first 20 timesteps there is no change and we can see that the adaptive motion model performs better than the static. In theory the static motion model has the best estimate of robot' motion as they are initialized with the same parameter value so the error should be less for static motion model. We don't see this because the location of particles are randomly initialized therefore it might be at different location to robot's start position. It needs the sensor model to decrease the error and as the sensor noise is high, it takes time to build the weight of particles. In the adaptive motion model to compensate for the sensor noise the distribution of motion model gets wider which helps in a better spread of the particle set.

In Figure 4.4 we plot the difference between the maximum weight and average weight of the particle set at each time step. This gives us an indication of the width

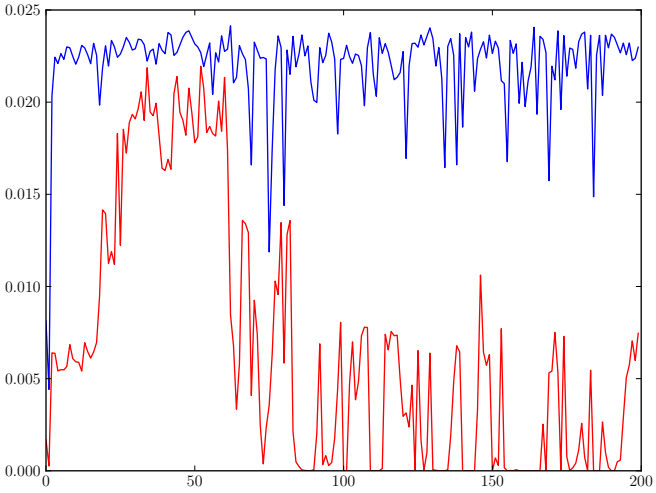
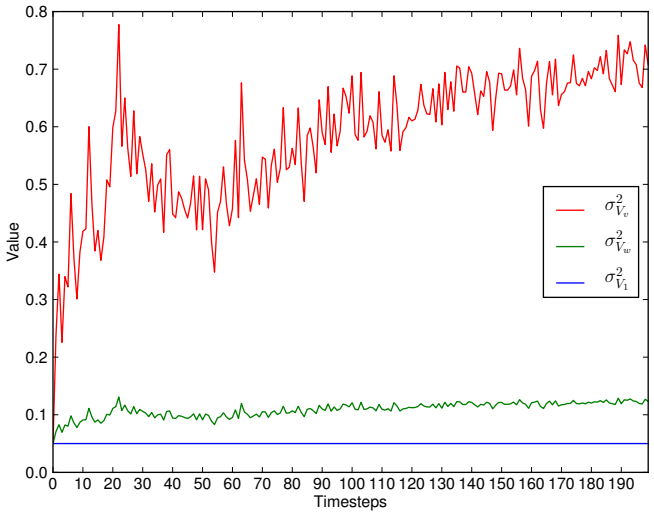


Figure 4.4: Difference between the maximum weight and average weight of the particle set. The motion model noise $\sigma^2_{V_v}$ is changed from 0.05 to 0.5 at timestep 60. The sensor noise is 1.0. The blue line and red line represents adaptive and static motion model respectively.

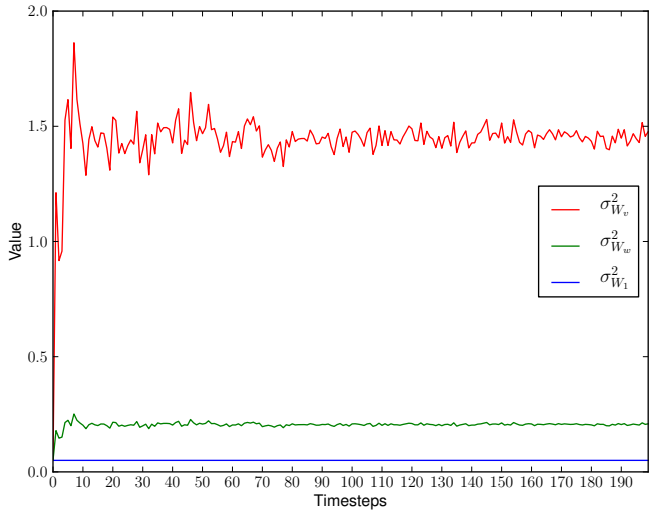
of the distribution of the particle set. If we have a higher difference it means that the particle set contains particle which are sure about robot's location or vice versa. When the weights of particle are higher we can see the static motion model performing better. This is due to the resampling step of particle filters where they choose the most probable particles based on motion and sensor models. As soon as the algorithm starts picking the most probable particles, the error goes down. In the case of adaptive motion model the distribution of motion model gets wider right from the start which leads to higher difference thereby resulting in a decrease of localization error. After time step 60 we change the motion model noise and we can see that adaptive motion model performing much better as compared to the static motion model. The difference in the weights of particles in Figure 4.4 also goes down.

In Figure 4.5 we don't see a jump in the red line or green line at time step 60 and this due to the fact that the estimate of parameters before the change was greater than the change after time step 60.

In the next simulation $\sigma^2_{V_v}$ is changed from 0.05 to 1.0 and the errors, estimation of parameters and average weight are shown in Figure 4.6, Figure 4.7 and Figure 4.8 respectively.



(a)



(b)

Figure 4.5: Estimate of parameter values a) $\sigma^2_{V_v}$, $\sigma^2_{V_w}$, $\sigma^2_{V_1}$ b) $\sigma^2_{W_v}$, $\sigma^2_{W_w}$, $\sigma^2_{W_1}$ at every time step. The motion model noise $\sigma^2_{V_v}$ is changed from 0.05 to 0.5 at timestep 60. The sensor noise is 1.0.

In both the simulations the robot's $\sigma^2_{V_1}$ is 0.05 and is not changed throughout the simulation. Table 4.2 shows that in the adaptive motion model the estimated value

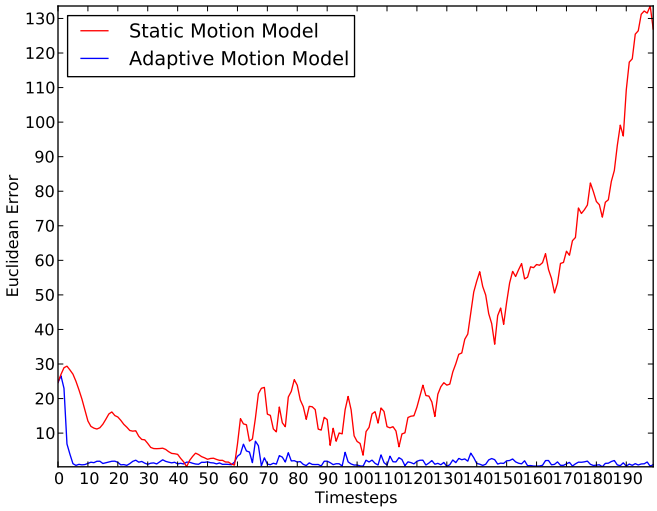


Figure 4.6: Plot showing the localization error between robot and the estimate of the robot's location by particle filters. The $\sigma^2_{V_v}$ is changed from 0.05 to 1.0 at timestep 60. The sensor noise is 1.0.

of $\sigma^2_{V_1}$ changes from the initialization value. Similarly other parameters such as $\sigma^2_{W_V}$ and $\sigma^2_{W_1}$ are also changed. This error in estimation is not of much concern as the main goal of the algorithm proposed was to learn the motion model to decrease the localization error.

In the first two simulations to simulate changes in the environment we changed $\sigma^2_{V_v}$. In the next two simulations we induce drift in the system and compare both motion models in terms of localization error. Drift is a common problem found in all robots. In AUV we get drift from INS systems. On land based robots there is drift present in the wheel encoders. In this simulation we subtract a constant number from the reported movement. This was also done to mimic the affects of water currents on AUV.

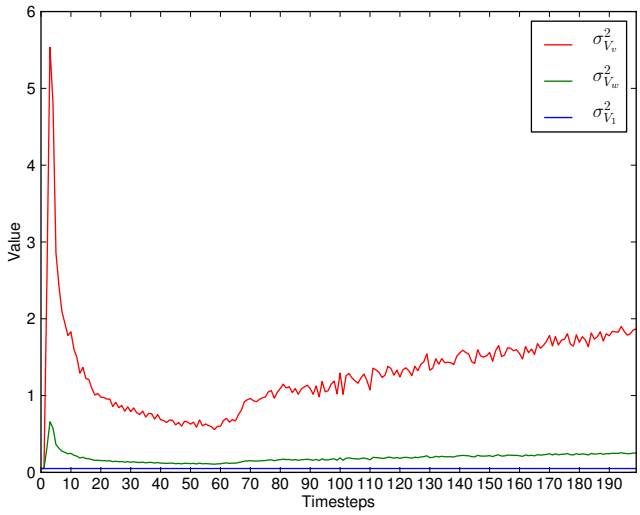
In order to simulate drift we describe the translational and rotational movement as

$$V_t \sim \mathcal{N}(v_t - a, v_t^2\sigma^2_{V_v} + w_t^2\sigma^2_{V_w} + \sigma^2_{V_1}) \tag{4.3}$$

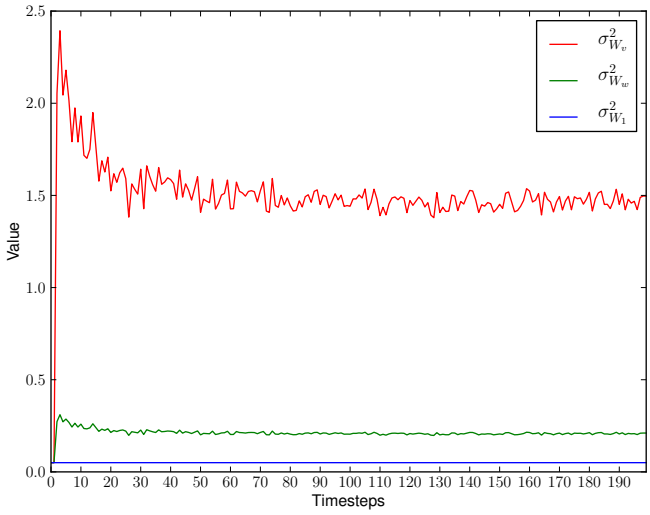
$$W_t \sim \mathcal{N}(w_t - b, v_t^2\sigma^2_{w_v} + w_t^2\sigma^2_{V_w} + \sigma^2_{W_1}) \tag{4.4}$$

where a and b are constants representing drifts in the system.

Table 4.3 gives a summary of the simulations performed to demonstrate the



(a)



(b)

Figure 4.7: Estimate of parameter values a) $\sigma^2_{V_v}$, $\sigma^2_{V_w}$, $\sigma^2_{V_1}$ b) $\sigma^2_{W_v}$, $\sigma^2_{W_w}$, $\sigma^2_{W_1}$ at every time step. The motion model noise $\sigma^2_{V_v}$ is changed from 0.05 to 1.0 at timestep 60. The sensor noise is 1.0.

effectiveness of the algorithm to account for drift. To account for the drift we don't include an extra parameter in the motion model as the variances $\sigma_{V_v^2}$, $\sigma_{V_v^1}$ should

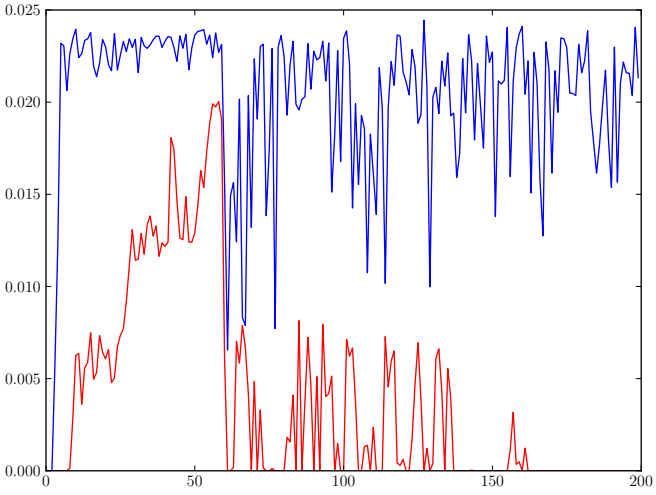


Figure 4.8: Difference between the maximum weight and average weight of the particles. The motion model noise $\sigma^2_{V_v}$ is changed from 0.05 to 1.0 at timestep 60. The sensor noise is 1.0. The blue line and red line represents adaptive and static motion model respectively.

account for drift and we see that in Table 4.3. The rest of the parameters remain unchanged.

The drift is present throughout the simulation. The performance of the algorithm is shown in Figure 4.9. The red line i.e. static motion model performs worse as compared to the adaptive motion model. The model adjusts its parameters as shown in Figure 4.10 to account for the drift in the system. The sensor model works to contain the drift in the system. Over time as it builds sufficient amount of particles

Initial Parameter Values		Changed Parameter Values	Estimated Parameter Values	Drift	Trajectories	Sensor noise
$\sigma^2_{V_v}$	0.05	0.05	0.461	2.0	3	1.0
$\sigma^2_{V_w}$	0.05	0.05	0.05	2.0	3	1.0
$\sigma^2_{V_1}$	0.05	0.05	0.095	2.0	3	1.0
$\sigma^2_{W_v}$	0.05	0.05	1.460	2.0	3	1.0
$\sigma^2_{W_w}$	0.05	0.05	0.05	2.0	3	1.0
$\sigma^2_{W_1}$	0.05	0.05	0.206	2.0	3	1.0

Table 4.3: Initial and estimated values of parameters with drift

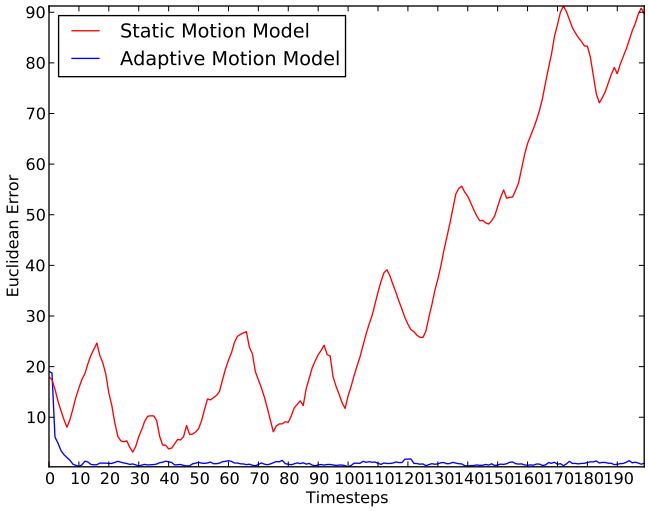


Figure 4.9: Plot showing the localization error between robot and the estimate of the robot's location by particle filters. The drift is present throughout the simulation described by equation 4.3 with a=2 . The rest of the parameters remain the same and the sensor noise is 1.0.

to be confident of robot's location it then relocates the robot. This characteristic of sensor model leads to a sinusoidal pattern in red line.

Another way to include drift in the system is

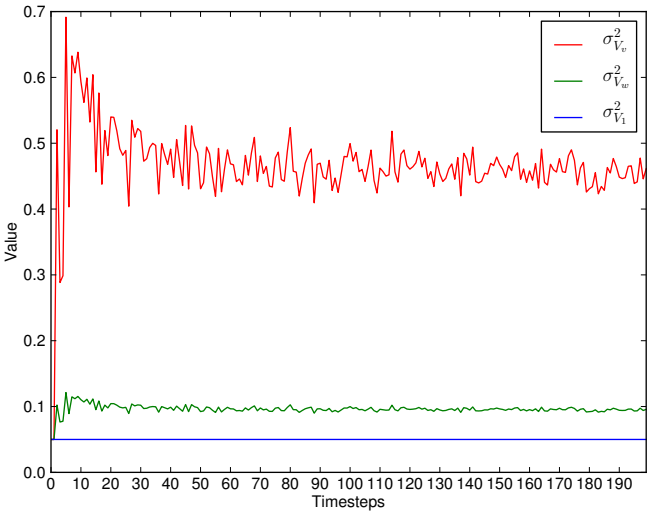
$$V_t \sim \mathcal{N}(v_t * a, v_t^2\sigma_{V_v}^2 + w_t^2\sigma_{V_w}^2 + \sigma_{V_1}^2) \tag{4.5}$$

$$W_t \sim \mathcal{N}(w_t * b, v_t^2\sigma_{w_v}^2 + w_t^2\sigma_{V_w}^2 + \sigma_{W_1}^2) \tag{4.6}$$

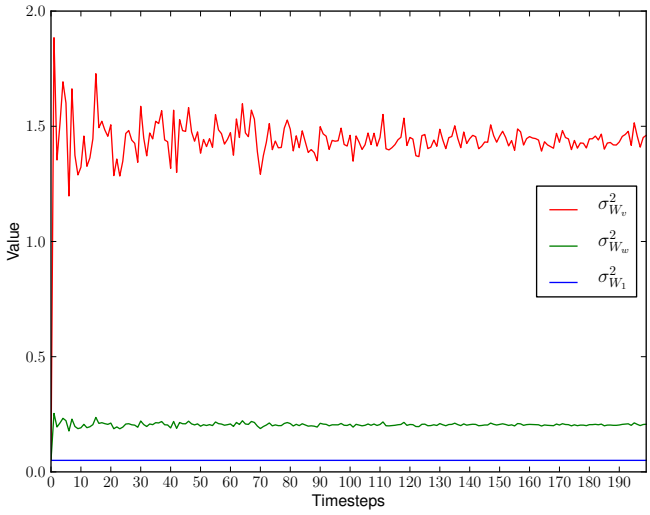
The results are described in Figure 4.11 and in this case as well the drift is present throughout the simulation. The static motion model performs worse as compared to adaptive motion model. In the figure we can see the red line coming down for a short time as the robot gets relocated based the estimate of the sensor model.

In all the simulations above we assume that the sensor noise is constant through out the simulation. In real world simulations we find that the quality of sensor readings varies with environment. For example, in an AUV we won't get sonar readings throughout the mission. This could be because sometimes its difficult to find the bottom of the sea floor or sonar sensor could be switched off for some time periods to save power on the battery.

In the next simulation, shown in Figure 4.12 there are two changes at time steps



(a)



(b)

Figure 4.10: Estimate of parameter values a) $\sigma^2_{V_v}$, $\sigma^2_{V_w}$, $\sigma^2_{V_1}$ b) $\sigma^2_{V_v}$, $\sigma^2_{V_w}$, $\sigma^2_{V_1}$ at every time step. The drift is present throughout the simulation described by equation 4.3 with a=2 . The rest of the parameters remain same and the sensor noise is 1.0.

60 and 100. At time step 60 we change the motion model noise $\sigma^2_{V_v}$ from 0.05 to 0.5. We change the sensor noise from 1.0 to 10.0 at time step 100 and compare

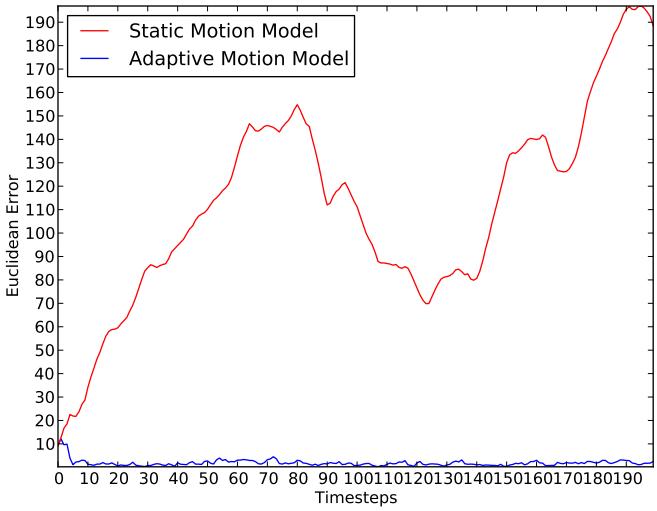


Figure 4.11: Plot showing the localization error between robot and the estimate of the robot's location by particle filters. The drift is present throughout the simulation described by equation 4.5 with a=2. The rest of the parameters remain same and the sensor noise is 1.0.

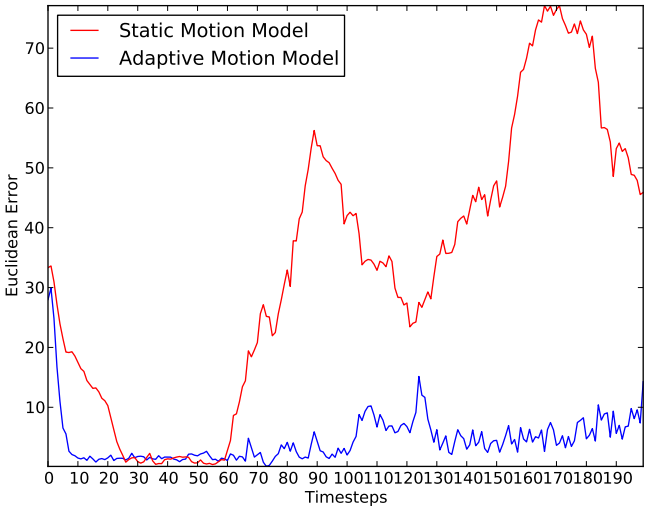
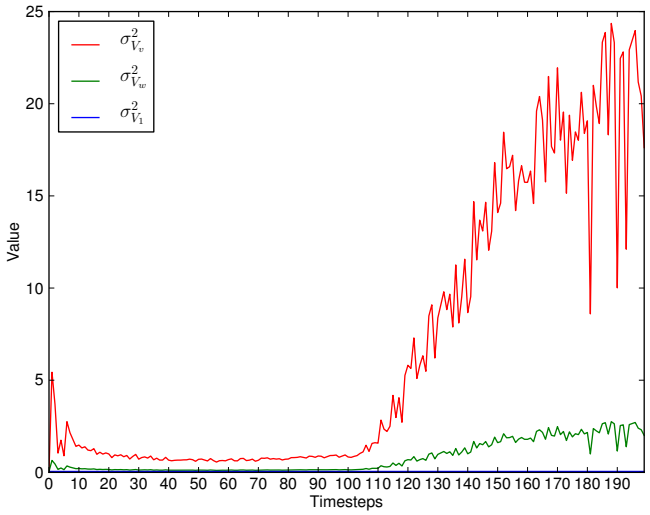
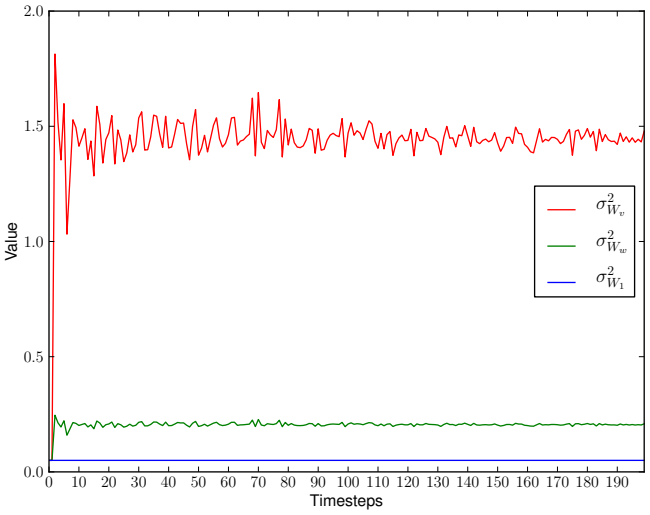


Figure 4.12: Plot showing the localization error between robot and the estimate of the robot's location by particle filters. The $\sigma^2_{V_v}$ is changed from 0.05 to 0.5 at timestep 60. The sensor noise is changed from 1.0 to 10.0 at timestep 100.

the behavior of static and adaptive motion model. After time step 100 we can see the error growing in adaptive and static motion model. The adaptive motion model quickly learns the sensor noise is high and starts relying on its motion model. This



(a)



(b)

Figure 4.13: Estimate of parameter values a) $\sigma^2_{V_v}$, $\sigma^2_{V_w}$, $\sigma^2_{V_1}$ b) $\sigma^2_{W_v}$, $\sigma^2_{W_w}$, $\sigma^2_{W_1}$ at every time step. The $\sigma^2_{V_v}$ is changed from 0.05 to 0.5 at timestep 60. The sensor noise is changed from 1.0 to 10.0 at timestep 100.

helps in decreasing the localization error and can be seen in Figure 4.12. In learning with a high sensor noise we are adjusting our motion model to compensate for the

noise in the sensor model therefore we see an increasing estimate of the parameters (Figure 4.13).

In this chapter we have shown results for both static and adaptive motion model. In all the cases tested in my thesis the adaptive motion has performed better. The localization error was less in adaptive as compared to static motion model. In the first two simulations where we changed the noise in the motion model adaptive motion quickly adapted to those changes. Even with low sensor noise, static motion model couldn't decrease the localization error. This shows that even with low sensor noise and to cope with changes in the motion model, adaptive motion model is a better choice.

Similarly with low sensor noise when drift was present in the system, we saw adaptive motion model easily adapting to the drift. In static motion model occasionally we saw the sensor model pulling back localization error. Although the parameter estimates are not that accurate, the result is good in terms of localization performance. The parameters estimates take into account the noise in sensor model. Therefore we don't see the estimated parameters being highly accurate. With high sensor noise we over estimate the parameters. This helps in distributing the mass in the particle filters and gives the algorithm more chance to pick particles which are near to the robot. The over estimation of parameters helps in decreasing localization error even with high sensor noise.

In this section we studied the adaptive motion model in more detail and showed that the adaptive motion model is a better choice. To our knowledge we haven't seen any detailed study of adaptive motion model as well as comparison of estimated parameters to real parameters.

Chapter 5

Landmarks extraction using Side Sonar Images

5.1 Introduction

The sensor model $p(z_t|x_t)$ is the probability of a measurement z given the robot is at position x. Thrun [34] divides the sensors for mobile robots in five classes which are contact sensors, internal sensors, proximity sensors, visual sensors and satellite-based sensors. Examples for various classes are shown in Table 5.1.

Classes of Sensors	**Examples of each class**
Contact Sensors	Bumpers
Internal Sensors	Accelerometers, Gyroscopes, Compasses
Proximity Sensors	Sonar, Radar, Laser range-finders, Infra-red
Visual Sensors	Cameras
Satellite Based Sensors	GPS

Table 5.1: Sensors for Mobile robots. Table taken from [34]

The measurements from these sensors in a particle filter algorithm are used to assign weights aka importance factor to particles. The most common way of sensing the environment is through landmarks. The sensors measure the distance, bearing or both from the landmarks to estimate their position in the environment.

In our simulated experiment the sensor model assumes to have four static landmarks and at all times we can measure the distance from them. For our algorithm to work on AUV we need some sort of reference points to measure the movement of the vehicle against. These reference points need to be computed on the fly as we don't have the luxury of having static maps for underwater environments. As my algorithm targets cases in which AUV doesn't loop back and we have limited field of view we need dynamic landmarks for our sensor model. To extract dynamic landmarks we used side sonar images and run feature extraction techniques such as SIFT on the images. These extracted landmarks can be used as reference points for our algorithm to adapt motion model for AUV.

The dynamic landmarks algorithm for side sonar images is not integrated in the simulated experiment described in Chapter 4. This is because of the unavailability of the motion data such as recording of IMU, DVL etc for AUV. To validate our algorithm for dynamic landmarks we extract motion information from real side sonar data and compare it to the total distance moved by the AUV. In the next two sections I describe side scan sonar and SIFT. In section 5.2 the algorithm to compute landmarks is presented. Chapter 6 contains the results of the algorithm.

5.1.1 Side Scan Sonar

Figure 5.1: Side scan sonar sensor using dual frequency made by JW Fishers . Image taken from [11]

Side scan sonar is used to create an acoustic image of the sea floor. Sound waves travel very effectively in water as compared to light therefore at lower depths sonar is used to image the sea floor instead of cameras. A side scan sonar sensor is shown in Figure 5.1. It measures how "loud" the return echo is and assembles an image as shown in Figure 5.2. In the sea floor there are hard areas such as rock and soft areas such as sand. The hard areas are represented by darker areas in the image as they return a stronger signal as compared to a soft area. Side scan sonar is the only imaging tool that works at low depth therefore we use it to extract landmarks for adapting the motion model.

Side-scan transmits sound energy and analyses the echo that is bounced off from the sea floor or other objects. It typically consists of three basic components: towfish, transmission cable and topside processing unit. It emits pulses in the shape of cone or fan to either sides of the towfish, typically to a distance of 100 meters. The

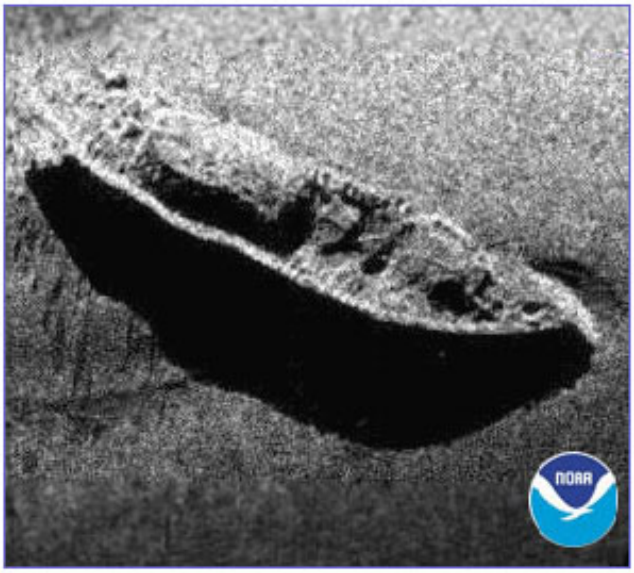


Figure 5.2: Side scan sonar image of the wreck. Image taken from [1]

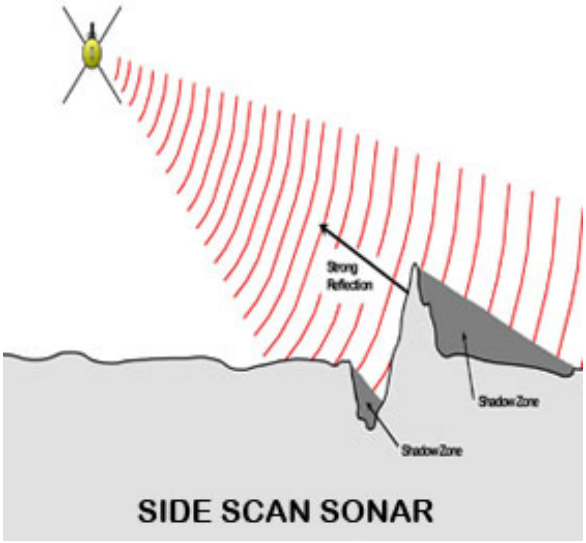


Figure 5.3: Working of a side scan sonar. Image taken from [1]

angular dimensions of these beams are designed to be narrow along-track and wide across track to cover as much seabed range as possible. The echoed sound waves are received by transducers and are continuously recorded. Each pulse shows a narrow strip below and to the sides of the transducers. The recorded echoes are put together along the direction of motion to form images of the seafloor. The sound frequencies typically range from 100 to 900 Khz in side scan sonar; higher frequencies yield

Sidescan Sonar Type	Frequency	Wavelength	Range
"Low"	5 kHz	30 cm	$>$ 50 km
"Low"	10 kHz	15 cm	10 km
"Low"	25 kHz	6 cm	3 km
"Medium"	50 kHz	3 cm	1 km
"Medium"	100 kHz	1.5 cm	600 m
"Medium"	200 kHz	0.75 cm	300 m
"High"	500 kHz	3 mm	150 m
"High"	1 mHz	1.5 mm	50 m

Table 5.2: Characterization of Sidescan system according to their operating frequency. Table taken form [35]

better resolution but less range. Currently on the market there are systems with dual frequency which allow the operator to use high frequency to produce sharper images or lower frequencies to cover greater depths. Side scan systems can be characterized according to their operating frequency [35].

Sonar are useful instruments in fisheries research, environmental studies and military applications such as mine detection.

5.1.2 SIFT

SIFT is a popular algorithm in computer vision to detect and describe local features of an image that are not affected by scaling and rotation. It was proposed by David Lowe in 1999 [25] and has been used for object recognition, robotic mapping and navigation, image stitching, video tracking and others. The various steps for extracting SIFT feature as described by David Lowe are explained in this section.

The first step in SIFT algorithm is to construct scale space i.e. create internal representations of the original image to ensure scale variance. In SIFT, progressively blurred out images are generated using the original image. In next step you the original image is resized to half of its size and again blurred out images are generated.

Images of same size are grouped together and belong to an octave. In Figure 5.4 there are four octave and each octave has five images. The number of octaves depends upon the original size of the image. The blurring of images in each octave can be represented mathematically by

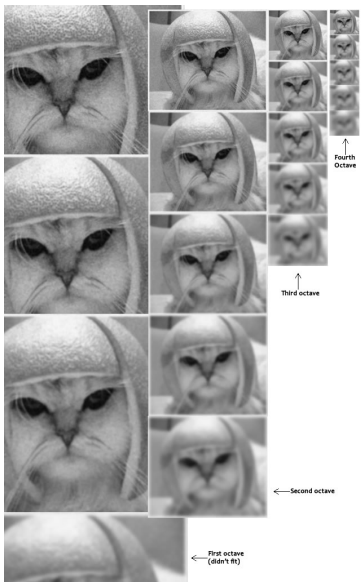


Figure 5.4: Representation of what Octaves look like in SIFT. Image taken from [36]

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{5.1}$$

where L is the blurred image, G is the Gaussian blur operator, I is an image, x,y are the location coordinates and σ is the scale parameter. The symbol $*$ represents the convolution operator. The next step is to find another set of images by Difference of Gaussian (DOG) which finds out interesting keypoints in the image. In this step the difference between two consecutive scales is calculated as shown in Figure 5.5. This process is done at every octave. As an example the DOG is applied to cat images and the output is shown in Figure 5.6.

After calculating the DOG images we can find the keypoints in two steps. We first locate the maxima/minima in DOG images. Secondly find subpixel maxima/minima. In the first step the algorithm iterates through each pixel and all the neighbors are checked. This process is explained in Figure 5.7

In Figure 5.7, X is the current pixel and the green circles represent the neighbors. X is marked as a keypoint if it is the greatest or least of all neighbours. These are approximate as mostly the maxima/minima never lies on a pixel. To access the data "between" pixels we need to calculate the subpixel location. This can be done by Taylor expansion of the image around the approximate keypoint.

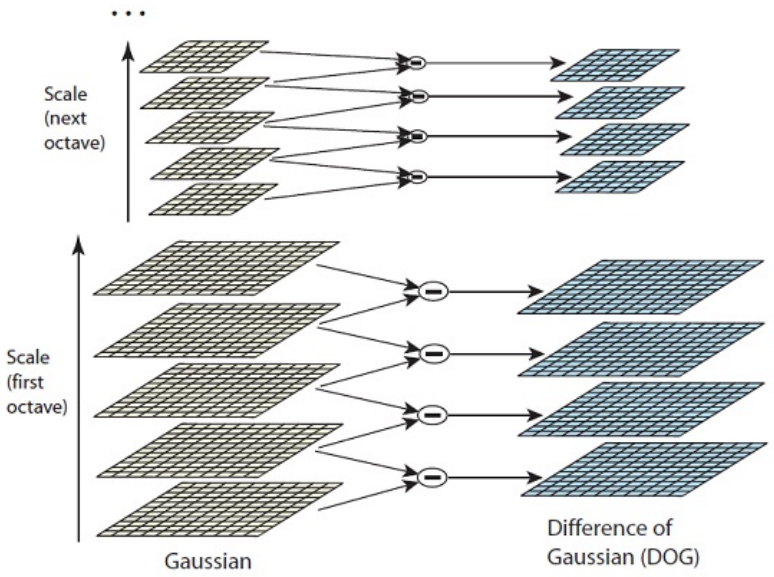


Figure 5.5: Difference of Gaussian done to calculate keypoints in the image. Image taken from [36]

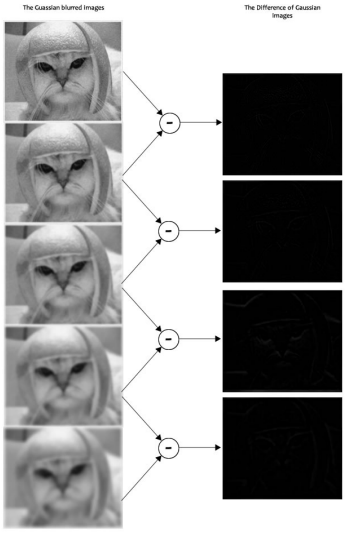


Figure 5.6: Applying DOG on a set of images present in a single octave. Image taken from [36]

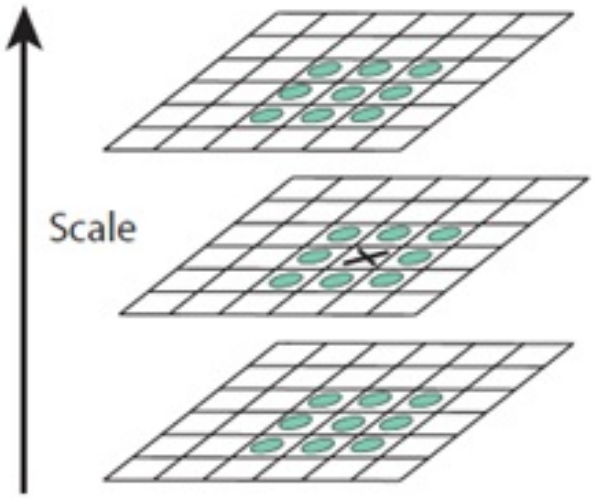


Figure 5.7: Locate maxima/minima in DOG images. X marks the current pixel and it is compared with its 26 neighbors. Image taken from [36]

After generating a set keypoints we need to eliminate points which have low contrast features and are present on an edge. The algorithm checks for intensity value at the pixel location of the keypoint and if it is less than a certain value if is rejected. To remove edges the perpendicular gradients are calculated at the keypoint. If both the gradients are small it is a flat region. If one gradient is big and other is small it is an edge. If both the gradients are big then it is a corner. The keypoints with both big gradients are considered to be a keypoint otherwise they are rejected. The SIFT algorithm can check if a point is a corner or not by using a Hessian matrix. After completing this step we have a set of legitimate keypoints which are scale invariant. The next step is assign orientation to each keypoint so that they are rotation invariant.

In this step, SIFT calculates gradient magnitudes and directions around each keypoint. To perform this, a histogram is created with 36 bins (each 10 degrees) representing 360 degrees of orientation. Suppose the gradient direction at a certain point is 20.56 degrees then it will go in 20-29 degree bin. The amount that is added to the bin is proportional to the magnitude of the gradient of the point. This process is repeated for all the pixels around the keypoint and the maximum bin is used to describe the keypoint. In Figure 5.8 it peaks at 20-29 degrees therefore the keypoint

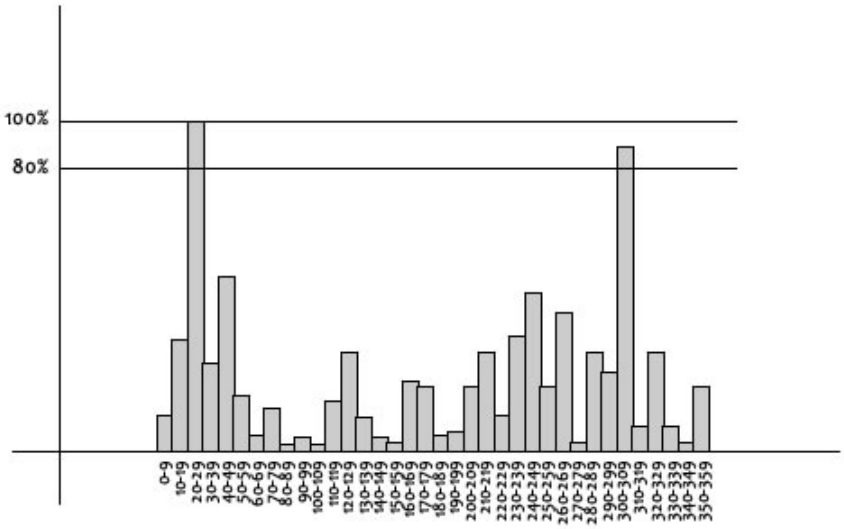


Figure 5.8: Histogram describing the bins for assigning orientation to the keypoint. Image taken from [36]

is assigned to third bin. In the Figure we also see that there is another peak which is above 80% of the highest peak. In this case the peak converted into a new keypoint. This new keypoint has the same location and scale as the original but has a different orientation.

In the final step of the SIFT algorithm, a unique fingerprint for a keypoint aka descriptor is calculated. To calculate the gradient we take 16×16 window around the each keypoint. This 16×16 window is broken into sixteen 4×4 windows (Figure 5.9).

In each window gradient magnitudes and orientations are calculated and are put in a 8 bin histogram (Figure 5.10). For examples a gradient orientation in the range of 0-44 degrees is put in the first bin. The amount added to the bin depends upon the magnitude of the gradient and the distance from the keypoint. The gradients that are far away from the keypoint will add smaller values to the bin. This can be performed using "Gaussian weighting function".

This is done for all 16 pixels, therefore we end up with $4 \times 4 \times 8 = 129$ numbers. The numbers are normalized and form the feature vector. This feature vector gives a unique identity to the keypoint. To achieve rotation independence, the keypoint rotation is subtracted from each orientation. Therefore each gradient orientation is

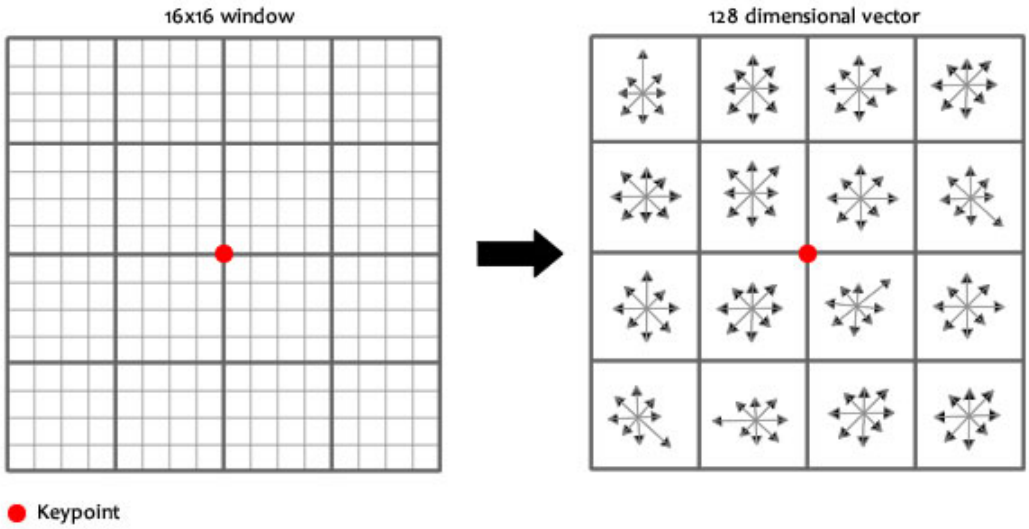


Figure 5.9: A 16×16 window is taken around a keypoint. This window is broken into sixteen 4×4 windows. Image taken from [36]

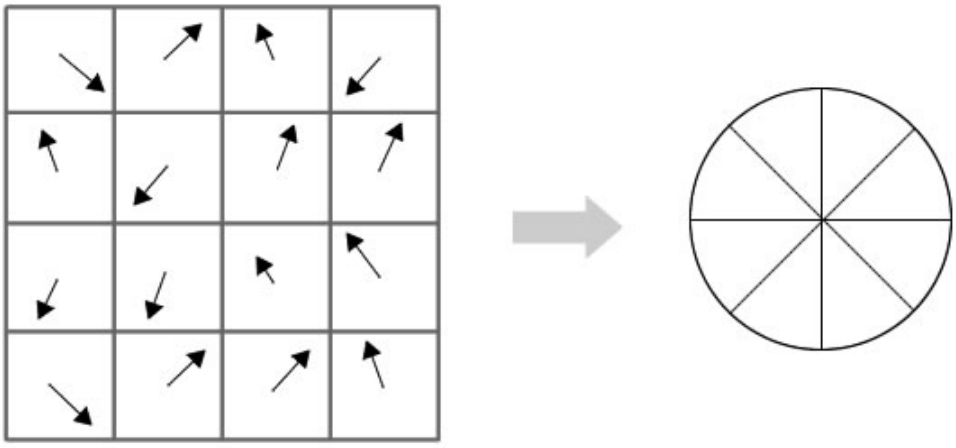


Figure 5.10: The gradient orientation is assigned to 8 bin histogram. The value depends upon the magnitude of the orientation and distance from the keypoint. Image taken from [36]

relative to the keypoints orientation. For illumination independence we threshold any numbers that are big and the resultant feature vector is normalized again.

After we calculate the descriptors for each keypoint, we have a set of features that describes the image and these can be used for various image processing tasks such as image matching, stitching etc.

SIFT has been widely used in various robotic applications. Stephen Se et.ãl [32] proposed an vision based algorithm to localize a robot and map the environment using SIFT features. Various algorithm have been proposed to estimate motion from camera images using SIFT features [2] [31]. Similar algorithms have been proposed to estimate motion underwater using camera images [6].

SIFT has been compared to self-organized features from restricted Boltzmann machines(RBM) in Hollensen [18]. Vardy et.ãl [37] compared various image registration techniques for side sonar images such as maximization of mutual information, log-polar cross-correlation, SIFT and phase correlation. He presented results and concluded that SIFT and phase-correlation provide the best performance among all the techniques. Peter King [21] described an algorithm to generate images from side scan sonar pings in real time. There implementation can be used as a black box to my algorithm.

5.2 Dynamic Landmarks

As stated above we need reference points to adapt our motion model. These reference points need to be dynamic as the learning algorithm is online. Secondly, we also assume that the AUV can't loop back on its route. To generate landmarks on side sonar images, a preprocessing step is required to get rid of horizontal lines produced by spurious electrical noise in the transducers. The noisy side sonar images is shown in Figure 5.11.

The preprocessing step involves using a median filter on the image to get rid of the noise. The preprocessed image is shown in Figure 5.12.

The disadvantage of filtering is that we loose information because the image is blurred. The blurred image won't give us meaningful keypoints therefore in the preprocessed image we see some horizontal lines due to restricted use of the filter. In order to get rid of all the lines we can increase the size of filter but the image will be

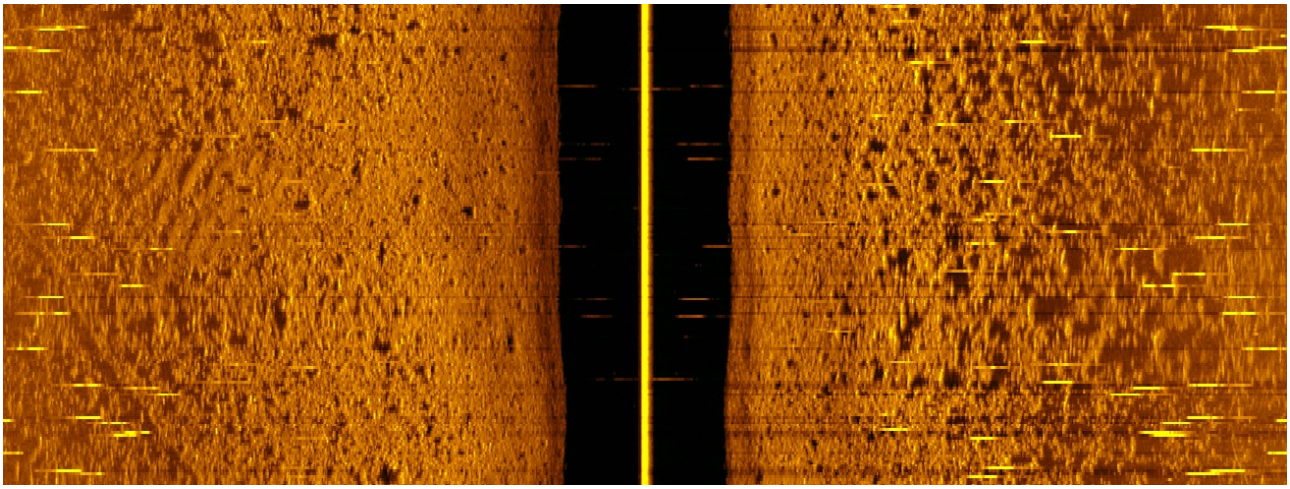


Figure 5.11: Image produced by Side scan sonar. Images produced from dataset provided by DRDC.

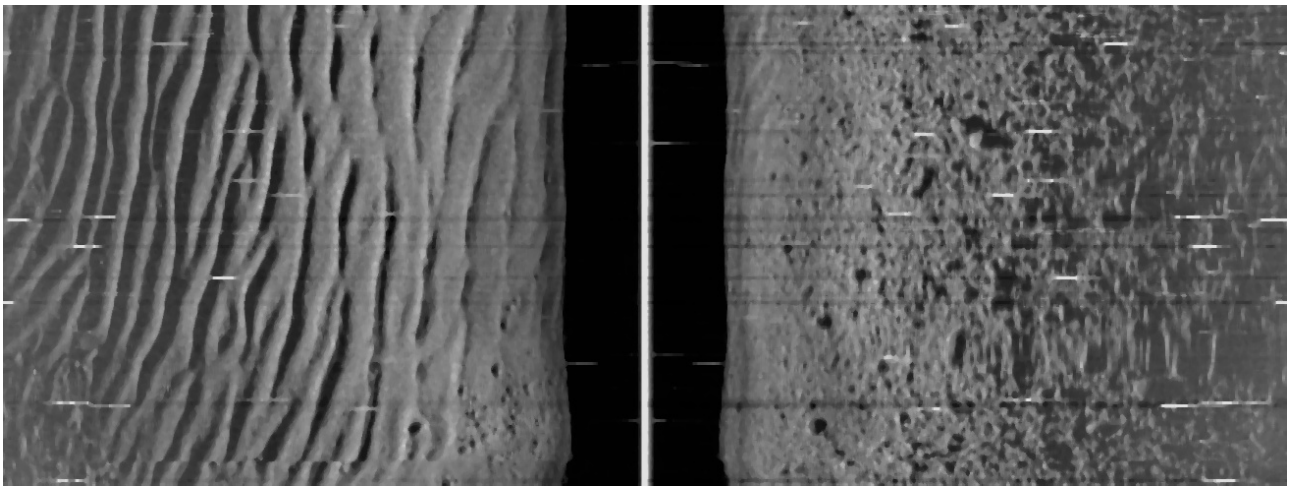


Figure 5.12: Median filter applied to side sonar image.

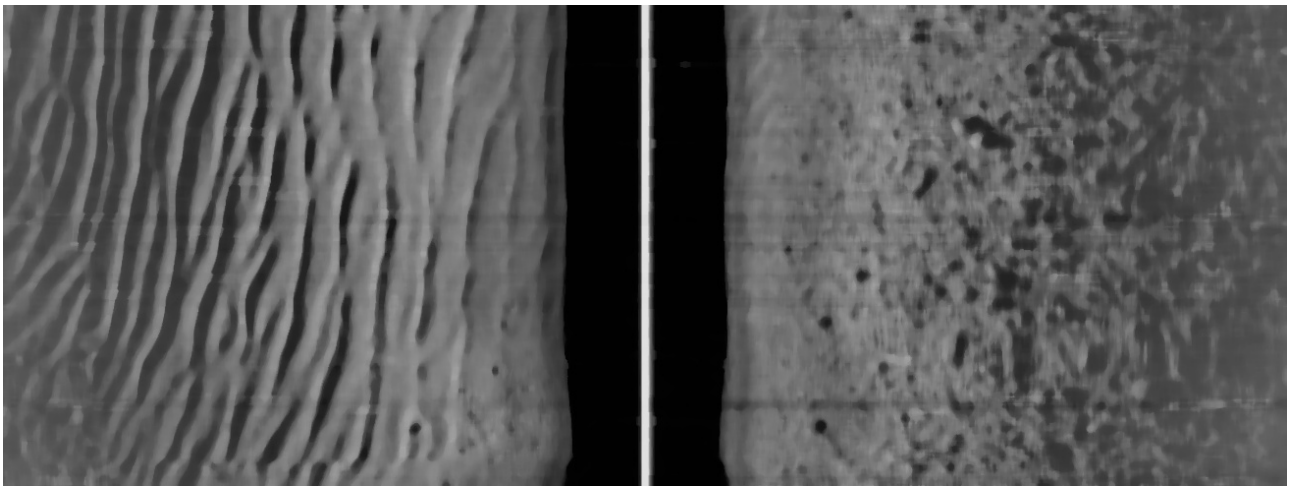


Figure 5.13: Median filter with high size applied to side sonar image.

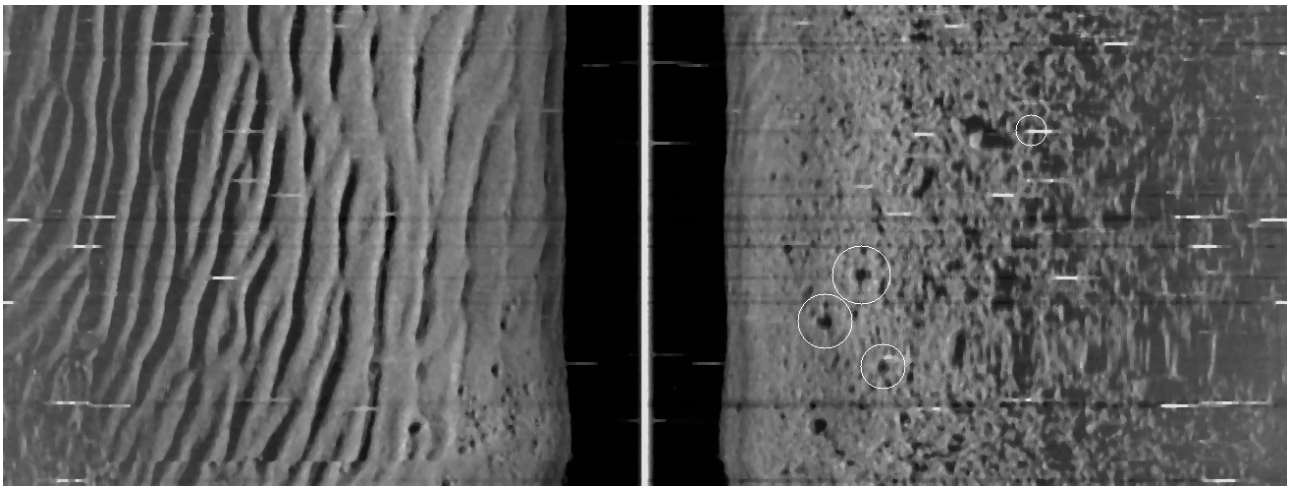


Figure 5.14: SIFT features on a side sonar image.

very blurred as shown in Figure 5.13.

The preprocessed image is used to extract landmarks for the AUV. To generate landmarks we use feature extraction techniques such as SIFT. Figure 5.14 shows the keypoints generated by SIFT using a high Hessian threshold. These keypoints can be used as landmarks to adapt our motion model. We need to keep in mind that we need a high Hessian threshold or else the algorithm will generate hundreds of landmarks as shown in Figure 5.15.

The distance of the AUV to landmarks in x-y plane can be measured. Similarly the distance of the particle filter estimate to the landmarks can be measured. Both the measured distances can be compared and used to assign weights to the particles.

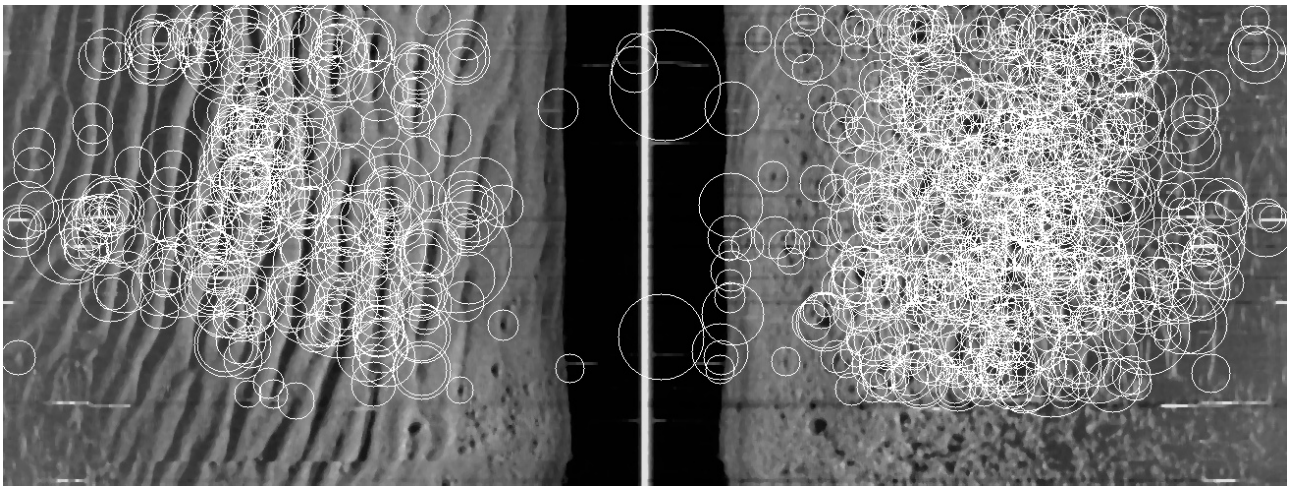


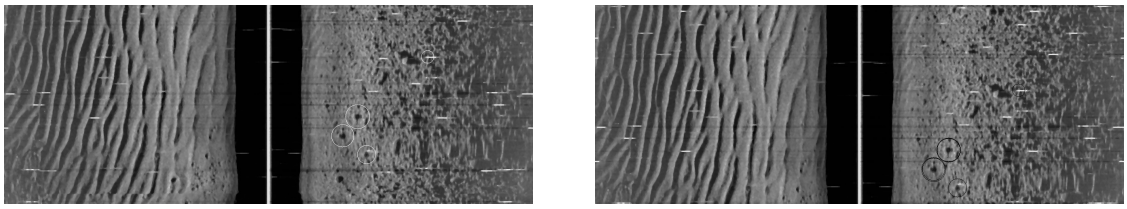
Figure 5.15: SIFT features on a side sonar image with low Hessian threshold.

Using dynamic landmarks allows our adaptive motion model algorithm independence from a static map and helps us in learning on the fly.

5.3 Motion Estimation using side sonar images

In many advanced land-based robots the position and orientation is determined through wheel encoders and velocity estimates. These techniques do not generalize well for marine environments [6]. Motion estimation using visual sensors such as camera are not restricted to particular locomotion and doesn't suffer from drift. There has been a lot of work done in estimating motion using camera images on land robots [2] [31]. Silvia [6] proposed an algorithm for AUV which used camera images and SIFT to estimate motion.

Hegreneas [16] combined the knowledge of vehicle dynamics to aid INS systems. In a similar manner visual motion information can be used to aid INS systems. The visual input to the dead reckoning algorithm has its pros and cons. The main advantage of using a visual estimate is that it doesn't suffer from drift which is prime concern for underwater vehicles. The disadvantage lies in the fact that we don't have side sonar images available every time. The second disadvantage is the computational power available on AUV. To specifically deal with the problems we use a high Hessian threshold to extract maximum of 4 landmarks so that feature matching is not computationally expensive.



(a) a (b) b

Figure 5.16: Two consecutive side sonar images. The white circles represent the landmarks and the black circle in next image shows the matched keypoints.

We verify our dynamic landmark approach by estimating motion information from side sonar images and compare it to reported movement by DVL. To generate keypoints we run SIFT on the side sonar images. We use a high Hessian threshold so that we can restrict the amount of keypoints. These keypoints are matched with keypoints of the next consecutive image using a KNN based matcher. This matcher is based on k-nearest neighbour based algorithm. As the name suggest the object is classified according to its k-nearest neighbours. Figure 5.16 shows two consecutive images that are used to estimate motion of the AUV. In the first image the keypoints are marked in white circles. These keypoints are matched with the second image and the matched keypoints are marked in black circles. The x and y position of the matched keypoints is compared to the original keypoints. This gives us motion estimate of the AUV.

The method to estimate motion is a very simple one with assumptions that AUV moves in a straight line at the same depth. The method proposed in the section was to verify the dynamic landmark approach instead of proposing an algorithm for estimating motion.

The performance of the algorithm is evaluated on real side sonar data. The results of the algorithm are compared to real motion information of the AUV and are discussed in the next chapter.

Chapter 6

Results

6.1 Motion estimation using side sonar images

We validate our algorithm on datasets consisting of side sonar images and the total distance the AUV moves. The datasets are provided by Defence Research and Development Canada(DRDC). The SIFT features that are applied to side sonar images are implemented using popular vision library(Opencv) in python. The motion estimation algorithm and feature matching is also implemented in python. The video datasets provided could only be read by a specific software called sonar data reader. To test our algorithm we extracted a set of images from the video and used it as an input to our motion estimation algorithm.

Datasets	Distance estimated by DVL(m)	Distance estimated through Side sonar images(m)
1	234.06	228.88
2	232.17	237.53
3	226.45	229.35
4	231.17	233.70
5	235.98	231.72
6	232.17	225.65
7	218.84	229.15

Table 6.1: Results of motion estimation using side sonar images. The results are compared to estimated distance by DVL, which is treated as ground truth.

Table 6.1 shows the results of motion estimation using side sonar images. The distance is compared to DVL. It is used to estimate the speed of the vehicle. This can be combined with a position fix, compass heading, and data from the acceleration sensors the position of the vehicle can be determined. These estimates can be noisy but at the same time there is no practical noise free way to measure how much the

Frequency	Long term accuracy	Range
150 kHz	±0.5% o.s. ± 2 mm/s	425-500 m
300 kHz	±0.4% o.s. ± 2 mm/s	200 m
600 kHz	±0.2% o.s. ± 1mm/s	90 m
1200 kHz	±0.2% o.s. ± 1 mm/s	30 m

Table 6.2: RDI Workhorse Navigator Doppler Velocity Log and range specifications. o.s. - of speed. Table taken from [19]. This DVL is used in Hugin 1000.

AUV actually moved.

The accuracy of DVL is dependent upon the frequency at which it operates. Higher frequency leads to better accuracy as shown in Table 6.1.

The algorithm gives us a decent estimate of the movement. The results are within 5% of the ground truth estimated by DVL. The movement from side sonar images can be coupled with DVL through a Kalman filter to give a better estimation of the movement. Hegeranes [16] in his paper pointed out the use of external sensors to aid INS and estimation of movement from side sonar images can be used to contain the drift inherent in INS systems. The results presented shows the accuracy of our motion estimation algorithm.

Chapter 7

Conclusion

The work presented here is a step towards building adaptive systems for AUV. As these systems take into account changes in the robot and the environment they are particularly useful for robotic missions over long duration. Adaptive systems can get rid of the need for labor intensive calibration process. They can also reduce the frequency of recalibration of a robot while on a mission.

Probabilistic robotics has been the way to approach robotics for a long time. The motion and sensor model are key ingredients to any algorithms for navigation, localization and map building. These models are treated as Gaussian distribution with static parameters. The questions that arises is how these models react to changes in the environment and robots. Higher level task such as navigation and path planning are dependent upon the accuracy of these models. To adapt to these changes we need to change our parameters to a model with time.

The goal of the work presented in this thesis was to present an algorithm that can adapt the parameters of an AUV's motion model. The work was primarily focused on developing and comparing an adaptive motion model to a static motion model. The use of side sonar images as a feedback to the algorithm enabled the algorithm to learn on the fly. Expectation maximization was used to learn the parameters in an unsupervised manner. The second part of the thesis focused on validating our dynamic landmark approach. This was done by estimating motion from side sonar images and comparing it to DVL. The results of the first section is illustrated in an simulation to produce pose estimates by integrating the motion and sensor models. The pose estimates of static and adaptive motion model are compared and the localization error is plotted. The estimated distances by the algorithm on real side sonar datasets are presented in the results of the second section.

Overall the work illustrated that adapting motion model can be extremely advantageous in decreasing the localization error which can in turn help a robot in better

navigation and path planning. The estimated parameters accuracy depends upon the sensor noise. It can be low but the goal of the thesis was to decrease the localization error over time with changes in environment. The method doesn't require any pre learning phase and helps in getting rid of the manual labour associated with the calibration process. The algorithm doesn't require the AUV to loop back in its path to learn the right parameters for the model.

Adaptive systems suffer from stability-plasticity dilemma. Plasticity helps in learning new information whereas stability prevents forgetting previous knowledge. To perfectly adapt the systems there needs to equilibrium between the two. My algorithm gives equal importance to every dataset acquired at any timestep. In the future, I would like to have a time decay function applied on the side sonar dataset. This would help up in adapting our model to most current changes. The algorithm as per now is limited to two dimensions and work needs to be done to get it working for six degrees of freedom. The work presented here is in a simulator and I feel there needs to be some extra work done in terms of testing and porting the algorithm. The compatibility of my algorithm with Andrew Vardy's [37] algorithm to generate side sonar images needs to be tested. The learning algorithm assumes the noise to be Gaussian. In the future, I would like to experiment with various noise distributions that are more suited for underwater noise. Overall the work presented here shows the importance of having an adaptive motion model for better localization of robots.

Bibliography

[1] Side Scan Sonar. `http://www.nauticalcharts.noaa.gov/hsd/SSS.html/`.

[2] Timothy D Barfoot. Online visual motion estimation using fastslam with sift features. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 579–585. IEEE, 2005.

[3] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 1. springer New York, 2006.

[4] Zhe Chen. Bayesian filtering: From Kalman filters to particle filters, and beyond. *Statistics*, 182(1):1–69, 2003.

[5] Ingemar J Cox and Gordon Thomas Wilfong. *Autonomous robot vehicles*, volume 447. Springer-Verlag New York, 1990.

[6] Silvia Silva da Costa Botelho, P Drews, Gabriel Leivas Oliveira, and da Silva Figueiredo. Visual odometry and mapping for underwater autonomous vehicles. In *Robotics Symposium (LARS), 2009 6th Latin American*, pages 1–6. IEEE, 2009.

[7] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

[8] Arnaud Doucet, Simon J Godsill, and Mike West. Monte Carlo filtering and smoothing with application to time-varying spectral estimation. In *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, volume 2, pages II701—-II704. IEEE, 2000.

[9] Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of Nonlinear Filtering*, 12:656–704, 2009.

[10] Austin I Eliazar and Ronald Parr. Learning probabilistic motion models for mobile robots. In *Proceedings of the twenty-first international conference on Machine learning*, page 32. ACM, 2004.

[11] J W Fishers. Side Scan Sonar. `http://www.jwfishers.com/sss.htm/`.

[12] Thor I Fossen. *Guidance and control of ocean vehicles*, volume 199. Wiley New York, 1994.

[13] Simon J Godsill, Arnaud Doucet, and Mike West. Monte Carlo Smoothing for Nonlinear Time Series. *Journal of the American Statistical Association*, 99(465):156–168, March 2004.

[14] Neil J Gordon, David J Salmond, and Adrian F M Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In *IEEE Proceedings F (Radar and Signal Processing)*, volume 140, pages 107–113. IET, 1993.

[15] G Grisettiyz, Cyrill Stachniss, and Wolfram Burgard. Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2432–2437. IEEE, 2005.

[16] Oyvind Hegrenaes, Einar Berglund, and Oddvar Hallingstad. Model-aided inertial navigation for underwater vehicles. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1069–1076. IEEE, 2008.

[17] O Hegrenses, Oddvar Hallingstad, and Bjørn Jalving. Comparison of mathematical models for the hugin 4500 auv based on experimental data. In *Underwater Technology and Workshop on Scientific Use of Submarine Cables and Related Technologies, 2007. Symposium on*, pages 558–567. IEEE, 2007.

[18] Paul Hollesen, Warren A Connors, and Thomas Trappenberg. Comparison of learned versus engineered features for classification of mine like objects from raw sonar images. In *Advances in Artificial Intelligence*, pages 174–185. Springer, 2011.

[19] Bjorn Jalving, Kenneth Gade, Kristian Svartveit, and Robert Sorhagen. DVL Velocity Aiding in the HUGIN 1000 Integrated Inertial Navigation System HUGIN 1000 Navigation System Philosophy. (2027).

[20] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[21] Peter King, Andrew Vardy, Peter Vandrish, and Benjamin Anstey. Real-time side scan image generation and registration framework for auv route following. In *Autonomous Underwater Vehicles (AUV), 2012 IEEE/OES*, pages 1–6. IEEE, 2012.

[22] Andrew Lammas, Karl Sammut, and Fangpo He. 6-DoF Navigation Systems for Autonomous Underwater Vehicles. 2004.

[23] Steven Michael LaValle. *Planning algorithms*. Cambridge university press, 2006.

[24] John J Leonard, Andrew A Bennett, Christopher M Smith, and H Feder. Autonomous underwater vehicle navigation. In *IEEE ICRA Workshop on Navigation of Outdoor Autonomous Vehicles*, 1998.

[25] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. IEEE, 1999.

[26] Thomas Minka. Expectation-maximization as lower bound maximization. *Tutorial published on the web at http://www-white.media.mit.edu/tpminka/papers/em.html*, 1998.

[27] Branko Ristic, Sanjeev Arulampalm, and Neil James Gordon. *Beyond the Kalman filter: Particle filters for tracking applications*. Artech House Publishers, 2004.

[28] Marshall N Rosenbluth and Arianna W Rosenbluth. Monte Carlo calculation of the average extension of molecular chains. *The Journal of Chemical Physics*, 23:356, 1955.

[29] Nicholas Roy and Sebastian Thrun. Online self-calibration for mobile robots. In *Robotics and Automation, 1999*, volume 3, pages 2292–2297. IEEE, 1999.

[30] Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Englewood Cliffs, 1995.

[31] Davide Scaramuzza and Roland Siegwart. Appearance-guided monocular omnidirectional visual odometry for outdoor ground vehicles. *Robotics, IEEE Transactions on*, 24(5):1015–1026, 2008.

[32] Stephen Se, David Lowe, and Jim Little. Vision-based mobile robot localization and mapping using scale-invariant features. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 2051–2058. IEEE, 2001.

[33] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[34] Sebastian Thrun, Wolfram Burgard, Dieter Fox, and Others. *Probabilistic robotics*, volume 1. MIT press Cambridge, MA, 2005.

[35] USGS. WHSC Sidescan Sonar Systems. `http://woodshole.er.usgs.gov/operations/sfmapping/sonar.htm/`.

[36] Utkarsh. SIFT: Scale Invariant Feature Transform. `http://aishack.in/2010/05/sift-scale-invariant-feature-transform/`, 2010.

[37] Peter Vandrish, Andrew Vardy, Dan Walker, and O A Dobre. Side-scan sonar image registration for AUV navigation. In *Underwater Technology (UT), 2011 IEEE Symposium on and 2011 Workshop on Scientific Use of Submarine Cables and Related Technologies (SSC)*, pages 1–7. IEEE, 2011.

[38] Miomir Vukobratovic. *Introduction to robotics*. Springer-Verlag New York, Inc., 1988.

[39] Teddy N. Yap and Christian R. Shelton. Simultaneous learning of motion and sensor model parameters for mobile robots. *2008 IEEE International Conference on Robotics and Automation*, pages 2091–2097, May 2008.