Planning an Interesting Path

by

©Abdullah Ali Faruq

A thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computer Science

Memorial University of Newfoundland

February 2017

St. John's Newfoundland

To my beloved mother who inspired me in every step of my life.
She always wished for my success, but
couldn't be with us to share the joy.

Abstract

Terrain aided navigation (TAN) is a well-studied method to localize an autonomous underwater vehicle in the absence of GPS. Researchers have been exploring new improvements; in particular Bachmayer and Claus have been incorporating terrain based navigation (glider TAN) into the Slocum gliders. To take full advantage of glider TAN, the glider path should favour areas of the ocean with uneven depth and unique features. This leads to a question of planning such an "interesting" path for the glider.

In this thesis, we present an offline path planning algorithm that optimizes the distance under the maximum uncertainty constraint. A major part of our contribution is developing a rating technique for evaluating the usefulness of an area of the ocean floor for reducing the uncertainty of the glider's position. We include experimental results showing how the generated path varies with the maximum allowable uncertainty, based on the ocean elevation data of the Conception Bay near Holyrood, Newfoundland.

Acknowledgements

I would like to thank all the people who contributed to this thesis with their help, support and inspiration. First and foremost, I thank my thesis supervisor Dr. Antonina Kolokolova for her excellent guidance and understanding. Her vast knowledge and experience helped me shape my work as a graduate student. I really appreciated the freedom she gave me to explore so many research directions. I am grateful to have the opportunity to work under her supervision.

I must thank my loving wife Shameema Anwar, who has always supported me and stood beside me during my struggles. She gave me the strength I needed to cope with the tragedies I faced in the past few years.

Last but not the least, I want to show gratitude to my parents for their unconditional love and support. I may have lost them, but their guidance will help me in every step of my life.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the advancement in robotics, Autonomous Underwater Vehicles (AUVs) have gained rapid popularity in the past few decades. AUVs are capable of completing a variety of tasks without any active human assistance. In recent years, their capabilities and applications have grown significantly; specifically the *Slocum glider* is drawing much attention in the research community. With a relatively slow speed, a glider can travel a long distance due to its low power consumption. Its long range capability along with economical value have encouraged its use in various underwater missions.

Like any other AUV, a glider can suffer from inaccurate localization. In the absence of any GPS signal, a glider has to localize using information from the surrounding environment. Significant studies have been done on this area and researchers have developed many techniques to address the problem. But in an unpredictable and dynamic environment like an ocean, these techniques are not always enough. Back in 2013 a glider research team at Memorial University, lost a Slocum glider during a field trial near Holyrood, Newfoundland [New13]. Localization techniques such as terrain-aided navigation can help to improve glider localization, but to be able to localize more precisely, a glider needs to follow an *"interesting"* path that favours

certain areas in the ocean which are suitable for the on-board localization. Here, we are focusing our work on the quest of computing such interesting path for the glider. Although the finding of our work can be applied to other AUVs, we are specially interested in Slocum gliders.

1.1 Slocum underwater glider

Slocum gliders are relatively small AUVs with long range capabilities. These gliders use variable buoyancy engine to glide in a saw-tooth pattern. In the absences of an active propulsion system, they are comparatively slower than other AUVs. They are usually equipped with a number of sensors to measure the surrounding environment. Here at the Autonomous Oceans Systems Lab of Memorial University, Dr. Ralf Bachmayer and Dr. Brian Claus have been working with Slocum gliders. They have performed multiple missions in the oceans near Newfoundland using these gliders. The primary motivation of our work came from the path planning requirements of those missions.

Despite their relatively slow speed, the gliders are well suited for a variety of missions including ocean data sampling and surveillance. Like any other autonomous vehicle, the mission success of a glider highly depends on its navigation capability, in particular on its localization technique.

1.2 Localization

In robotics, the term navigation refers to the task of safely and efficiently taking the robot from a given state to the desired state. In a simpler form, a state can be a point on a plane. In more complex cases, state includes location, orientation and other related information. The elements of a navigation system vary widely from one

Figure 1.1: Autonomous underwater vehicle: a Slocum glider [cF05]

design to another, but a fundamental part of most navigation is localization.

Localization is the process of acquiring knowledge about the current state of the robot in relation with its environment; in other words, knowing where the robot is actually located at a certain time. Usually it involves using a model of the environment or a *map*. The map can be constructed during the mission or a previously constructed map may be available during the localization. The environment is perceived through a single or multiple sensors like a GPS, camera or range finder, each of which produces some form of information. Most localization methods use these acquired information to determine the current state of the robot relative to the map. This may sound simple, but in reality, it can become very challenging to get the current state with proper accuracy.

One major challenge of localization arises from the noise in sensor measurement. Most sensors produce slightly deviated value from the actual value in the environment. The magnitude of the deviation may vary depending on the type and quality of the sensor, but such deviation can occur in all sensors. Even the most sophisticated

sensor can be erroneous to some extent. Some of these errors can be corrected by calibrating the sensing device, but we can not eliminate the error completely. When these measurements are used to generate a map of the environment, the map itself becomes erroneous and thus leads to further errors in localization. In many cases, the resulted state from localization is used as an input for the next iteration of localization. Thus the error becomes cumulative and if not corrected properly can cause total failure of the navigation system. Apart from the noise, the sensors are also limited by the type of information they can extract from the environment. A single beam sonar can measure the distance to an object, but to sense its color would require an additional camera sensor. And attaching all kind of sensors to every robot is not a feasible option.

Even if we had perfectly accurate sensors, the challenge of localization would not end there. In the real world, the surrounding environment is dynamic and highly unpredictable. The objects in the environment may not be static and other robots and humans may become a part of the environment. On top of that, the actuators which the robot uses to change its states introduce significant uncertainty. Considering all these arguments, it is not surprising that localization has received so much attention in the research community. A number of probabilistic localization methods [FHL$^+$03] have been fashioned using statistical estimators like Particle filter [DGA00, AMGC02, Sim06] to address these challenges.

In the probabilistic localization approach, the current state is not represented by an exact state; rather a probability distribution or *belief* is used instead. The belief is the likelihood of a state being the actual current state. These localization algorithms can be passive or active. In passive localization, the robot performs an action and changes its state. Based on that action along with information acquired through the sensor, the algorithm estimates the new state. Passive localization can be

implemented using statistical estimators such as *particle filters*. On the other hand, active localization algorithms aim to produce a plan that helps to localize better by reducing the uncertainty. In this work, we are only considering passive localization techniques, with all the path planning precomputed offline.

1.3 Glider localization

At Autonomous Oceans Systems Lab, Brian Claus and Ralf Bachmayer have developed the *gTAN* algorithm for glider localization based on the terrain-aided navigation [CB15]. With the help of on-board sensors and a static depth map of the ocean surface, a particle filter is applied to estimate the glider's position.

Like any other AUVs, the glider suffers from the unavailability of GPS as GPS signal cannot be received while underwater. As a result the glider has to rely on other localization methods such as localizing using the information obtained from the ocean terrain and surroundings. GPS is available when the AUV is at the surface, however this is not applicable for missions in ice covered areas like the Arctic. Apart from that, strong ocean current can drift the glider away from its desired path. For certain tasks, a sufficiently accurate localization algorithm is required to navigate safely in the ocean.

In gTAN, Claus and Bachmayer [CB15] have used a Digital Elevation Model (DEM) which contains the elevation information of the ocean floor with some margin of error (see Section 2.2.2.1 for details). A motion model based on Dead Reckoning (DR) system estimates the next position of the glider (see Section 2.2.1 for details). The DR keeps track of the glider location using glider velocity and traveling time. Due to the unpredictable ocean current and the error in velocity calculation, DR location estimation usually accumulates significant error and the uncertainty of the

glider's location increases with time. A particle filter can reduce that uncertainty by incorporating a depth measurement [CB15].

The glider is equipped with a single beam sonar altimeter and a pressure sensor. The altimeter estimates the distance between the glider and the ocean floor. The pressure sensor helps to calculate the distance between the glider and the ocean surface. Using these two values, the depth measurement model estimates the depth of the ocean floor at the gliders current location. The particle filter is applied to update the estimate from DR by matching the associated depth from the DEM. Both the gTAN algorithm and the particle filter are discussed in the next chapter.

Like any other terrain-aided navigation, the accuracy of gTAN largely depends on the area where the localization is taking place. A terrain rich with unique features can help to significantly reduce the uncertainty of the location estimation. On the contrary, a flat terrain in the ocean has very little to offer. A glider, travelling mostly on flat terrains, should have more uncertainty compared to the one travelling on terrains with *interesting* features. Hence arises the necessity of planning *an interesting path*; a path that favours interesting areas while optimizing the travel cost.

1.4 Problem statement

The goal of our work was to design an offline path planning method for AUVs that precomputes a path in such a way that the on-board terrain-aided navigation can localize better. We want to reduce localization uncertainty along the path, while at the same time minimizing the travel cost of that path. To address both, we are aiming to precompute the shortest path between two locations where the location uncertainty is bounded by a user-defined uncertainty constraint. The problem originally came from glider navigation using gTAN algorithm, but our methods are applicable to any

AUV that uses some form of terrain-aided navigation.

Given:

1. An elevation *map* of the ocean
2. A *start* and a *goal* location
3. A user-defined uncertainty *constraint*

Compute: A shortest path from *start* to *goal* such that the localization uncertainties in that path never exceeds the *constraint*

1.5 Related work

The necessity of a suitable path planner exists in many areas and researchers have designed a number of algorithms to meet those needs. However, the definition of the path planning problem varies from one field to another. In most cases, the final goal of these problems is to compute some "path" while optimizing some function for that path; but the definition and the requirements of the path can create huge difference among them. Though all of these problems are known as the path planning problem, the underlying problems can be very different from one another and may require different classes of algorithms to solve them. In some cases, an optimal solution is expected and a discrete state representation is acceptable; for such problems graph traversal algorithms are well-suited for efficient computation. But for some problems, the state configuration space can become exponentially large and even an efficient algorithm may not be a feasible option. In such cases, probabilistic or evolutionary algorithms can compute an acceptable approximation of the optimal solution. Undoubtedly, path planning is a broader term and the solution of a particular path planning problem will depend on the definition of that problem. In the next sections, we are going to

briefly describe different approaches to solve path planning followed by a discussion on how safety and uncertainty is addressed in these algorithms.

1.5.1 Path planning approaches in the literature

The typical solution for a path planning problem uses graph traversal algorithms. Dijkstra's algorithm [Dij59] can compute an optimal path by optimizing a cost function, where the cost can be modeled as distance, travel time or energy requirement. The computation time of Dijkstra can be improved significantly by using a heuristic algorithm such as A* [HNR68]. Although both of these algorithms compute optimal solutions, a large number of variations have been presented in the literature to address different aspect of the path planning problem. A Field D* algorithm proposed in [FS06b] uses linear interpolation to eliminate unnecessary changes of direction from the path computed by classic A*. In [NDKF07,DNKF10], the authors presented Any-angle Theta* algorithm which improves the shortest path on a grid by relaxing the angle restriction of the grid cells. Graph based path planning solutions serve well in many applications, but most of them suffer from a discrete representation of the environment. Specifically, in the presence of ocean current, path planning for AUVs requires more attention to the surroundings. Path planning using a variable ocean current model is presented in [FPCGHS$^+$10], which can compute path in continuous space and time. An iterative optimization technique is also used [IGHSFP$^+$11,FPHSIG$^+$11] in glider path planning considering the ocean currents. In [KSBB07], bidirectional flow in estuarine area is utilized in favour of the AUVs to minimize the energy expense.

The artificial potential field method has gained popularity in AUV path planning for addressing the ocean current and obstacles. A path planning technique using such a method is presented in [War90] to avoid paths getting too close to the obstacles. Artificial potential fields are created around the obstacles and the goal location such

a way that the path is attracted to the goal and repulsed by the obstacles. Numerical potential fields can also be used for that purpose [BLL92]. A two level path planning approach is proposed in [SR94] where the high level planner (HLP) uses a priori knowledge about the environment to optimize energy consumption and to produce some intermediate points. Then a low level planner (LLP) follows the intermediate points using potential field technique. A similar approach is taken in [YZF+13], which uses geometric methods for global path planning and continues local path planning with artificial potential fields. A modified version of the potential field technique is presented in [Sou11].

In recent years, the fast marching (FM) algorithm has been getting much attention in AUV path planning [PPPL05]. The fast marching algorithm is a special case of the level set method [OS88] that aims to solve the boundary value problem of the eikonal equation. Over the years, researchers have come up with new algorithms based on the fast marching method for path planning. Clément Pêtrès *et al.* has proposed an FM* algorithm [PPP+07] that produces continuous solution of AUV path planning based on discrete representation of the environment. In [VGGGGBM13], the authors have presented a comprehensive study on a number of variances of FM including FM^2 and FM^{2*}. The FM^2 algorithm follows the fast marching method while maintaining a safe distance from the obstacles present in the environment. In addition to that, the FM^{2*} algorithm can improve the computation time by introducing a heuristic function.

Path planning in a large configuration state space can become computationally expensive and computing an optimal solution may not be practical in those cases. A probabilistic approach for path planning is more suitable in such scenarios. Probabilistic roadmaps (PRM) is a sampling based path planning technique [KSLO96] aimed towards high dimensional configuration spaces specifically for robots with many degree of freedom. Usually path planning in PRM is attained in two phases; the learning

phase and query phase. In the learning phase, random free configurations are generated to construct a probabilistic roadmap and some form of local planner is utilized to connect these configuration. Later, in the query phase, a path is computed between two free configurations using that roadmap. Many PRM based algorithms are presented in the literature focusing on the computational efficiency at the expense of optimality [MB12]. The authors in [KF11] presented the complexity analysis of probabilistic path planning algorithms along with a optimal probabilistic roadmap (PRM*) algorithm. A new probabilistic path planning concept was introduced by Steven M. Lavalle [LaV98] as the rapidly-exploring randomly trees (RRT). Based on this concept, Kuffner and LaValle proposed a randomized algorithm [KL00] by constructing two rapidly-exploring random trees and connecting them using simple greedy heuristic. Variants of RRT approach can be found in [LK01, FS06a, ZKB07]. A solution for AUV path planning using RRT is presented in [TSC05]. Compared with other AUVs, the slow moving gliders are more impacted by ocean currents; Rao *et al.* [RW09] have addressed this issue by providing an RRT based path planning algorithm for gliders. In [KF11] the authors have presented similar algorithms such as rapidly exploring random graph (RRG) and optimal RRT (RRT*) to compute optimal path planning. Recently, another sampling based algorithm named fast marching tree (FMT*) algorithm is proposed in [JSCP15] which can produce asymptotically optimal path in less computational time with respect to probabilistic roadmaps and rapidly-exploring random trees.

Apart from probabilistic approaches, evolutionary algorithms have also been employed in AUV path planning problems with large configuration state space. Sugihara and Yuh have presented a genetic algorithm [SY97] for underwater path planning that can adapt with environmental changes such as dynamic obstacles. In their work, they have categorized the obstacles into solid and hazardous kind; where solid ob-

stacles are avoided all the time, but a path can go through hazardous obstacle for a higher path cost. Another genetic algorithm based path planning for AUV is proposed in [ACO04]. In [CFLC10], the authors fused genetic algorithm with dynamic programming technique to achieve AUV path planning. Apart from genetic algorithm, path planning using particle swarm optimization (PSO) and its variants are also present in the literature. A stochastic particle swarm optimization (S-PSO) algorithm is proposed in [CL06]. Recently, Zeng et al. presented a comparative study of popular AUV path planning approaches in [ZSL+16], along with a path planner based on Quantum-behaved particle swarm optimization (QPSO). Other evolutionary approaches include ant colony optimization (ACO) [LD09], hybrid ACO with PSO [SBL08] and imperialist competitive algorithm (ICA) [ZSL+15]. In [Agh12], the authors have presented underwater path planning solutions using five different evolutionary algorithms.

Among the other approaches for path planning problems, Li and Guo utilized neural network for planning paths in the estuary environment [LG12]. They have accounted for different oceanic conditions including static and dynamic currents. Recently, Yoo and Kim proposed an algorithm [YK15] using reinforcement learning to compute a near optimal path in reasonable time.

1.5.2 Addressing the safety and uncertainty in path planning

The study of AUV path planning requires special attention to the impact of dynamic ocean current present in the environment. Usually, AUV path planners ensure the safety of the vehicles by the means of avoiding the obstacles or keeping a safe distance from them [War90, CMN+92, ACO04, VGGGGBM13, AYK15]. But in a highly dynamic ocean environment, addressing only the obstacle avoidance is not enough. Specially, for the slow moving AUVs such as the gliders, the situation can become aggravated [RW09] and uncertain drift may happen from the actual path [YK15]. In

recent studies, the uncertainty of the ocean current is addressed while computing the path [ZSL+15, HDS15, WLMHK16].

Though uncertainty of the travel cost has been discussed in the path planning literature [DCZM12, NBK06, SS09], the uncertainty of the position seems to have received less attention. Yet a major issue in path planning for AUVs is the likelihood of them straying off the planned path. Pereira *et al.* have addressed the issue by proposing a planner [PBJ+11, PBHS13] that minimizes the risk of surfacing at the expense of a longer path. Most AUVs can not stay underwater forever and need to surface periodically for transmitting data and receiving instructions. Surfacing in an area of heavy marine traffic can be harmful for the AUV and such surfacing attempt can cause collision with surface vehicles. In their work, Pereira *et al.* have precomputed an optimized path for the AUV with low expected risk of collision while surfacing by trading additional path cost. Although, they have considered the risk of collision and the uncertain ocean current prediction in their planner, the chance of the AUV getting lost is not addressed properly. In [BTAH02] Bellingham *et al.* have accounted for the probability of getting lost in their path planner for unmanned aerial vehicles (UAVs). Nonetheless, in the absence of ocean current consideration, their approach may not be suitable for AUVs.

Though precomputing a safe path that favours the on-board localization technique does no seem to be sufficiently addressed, there has been research addressing online path planning. Dektor and Rock introduced an online localization method [DR12] that helps to localize better in areas which are comparatively less suitable for terrain based localization. Their method reduces the overconfidence and false fixes in uninformative terrain, thus addressing the uncertainty that results from the measurement error in map information. In their work, a modified particle filter is used which estimates the variance in terrain information and prioritises the measurement information

based on that variance. Notably, this approach focuses on improving localization in uninformative areas rather than avoiding such areas in path planning.

The focus of our work was designing an offline path planner for AUVs, specially for the gliders, that helps the online localization process by computing a path that favours areas more informative for localization. A similar approach is presented in [Ber93], which utilizes the relative measurement covariance matrices to identify useful areas for better localization using multi-beam sonar. Unlike our approach, the expected result of localization update is not addressed in this work and may not be applicable for the localization technique such as gTAN that uses a single beam sonar.

1.6 Thesis organization

The remaining portion of the thesis is organized as follows:

- In Chapter 2, we briefly describe some preliminary concepts, along with gTAN and related algorithms.
- In Chapter 3, we explain the rating technique to evaluate the usefulness of an area in the ocean. The chapter includes some experimental results to compare ratings performed on different areas in the map.
- In Chapter 4, we present an offline path planning algorithm that can produce interesting path using the rating technique from the previous chapter. The result of the algorithm is demonstrated with elevation data of the ocean near Holyrood, Newfoundland.
- In Chapter 5, we summarize our work and discuss future work

Chapter 2

Background

The idea of reducing uncertainty in AUV localization is far from new. Almost every localization algorithm is designed to accomplish this task. However, the story is a little different when it comes to path planning of AUVs. Usually, path planning refers to producing a list of intermediate states which will guide the AUV to reach the goal state from an initial state while optimizing some sort of travel cost. The cost can be distance, travel time or even the safety of the AUV. While many popular path planning algorithms can produce the optimal solution, in many cases the uncertainty associated with the path is ignored in the path planning. Before going into any further details, first we need to revisit a few preliminary concepts.

2.1 Probabilities and distribution

Working with uncertainty requires the understanding of probability. The probability of an event is the quantification of how likely that event can occur. To define probability in a formal fashion, we need to revisit some elementary concepts like experiments, events and sample spaces [Dek05]. An experiment is a process that produces some output and can be repeated any number of times. In most cases, the outcome of

an experiment is random, but the possible outcomes are well defined for an experiment. The set of all possible outcomes is known as the Sample space (Ω) of that experiment. In other words, the outcome of an experiment is an element of Ω. Any subset of sample space is known as an event. The occurrence of an event from a certain experiment is determined by whether the outcome of the experiment belongs to that event subset. Each event can be assigned a probability value that represents its likelihood of occurring. Two events are called mutually exclusive or disjoint when there is no common outcome between them.

Definition. *The probability of an event A in a finite sample space Ω is a non-negative number $P(A)$ in [0,1] such that the total probability $P(\Omega)$ is one and the probability of the union of any disjoint events is the same as the sum of the individual probabilities of those events.*

The probability of an event may change with additional knowledge about the occurrence of other events. Such probabilities are known as conditional probability. The conditional probability of an event A, given that event B has already occurred, can be defined as,

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

An important rule in probability theory is the *Bayes' Rule*. For disjoint events $A_1, A_2, \ldots, A_m$ with $A_1 \cup A_2 \cup \cdots \cup A_m = \Omega$, Bayes' Rule can be generalized as,

$$P(A_i|B) = \frac{P(B|A_i) \cdot P(A_i)}{P(B|A_1)P(A_1) + P(B|A_2)P(A_2) + \cdots + P(B|A_m)P(A_m)}$$

The rule can also be represented in its traditional form as follows,

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

In probability, a random variable can be either a discrete random variable or a continuous random variable. Each of these has significant usage and needs to be treated separately. A discrete random variable can be defined as a function $X : \Omega \to \mathbb{R}$, where Ω is the sample space and X can take on a finite number of values$(a_1, a_2, \ldots, a_n)$ or an infinite number of values$(a_1, a_2, \ldots, a_n, \ldots)$. The probability of a discrete random variable X can be represented with the *probability mass function* of X.

Definition. *For a discrete random variable X, the probability mass function is the function $p : \mathbb{R} \to [0, 1]$ where $p(a) = P(X = a)$ and $-\infty < a < \infty$.*

In many cases, storing and calculating all random variables is not practical, especially when the sample space Ω is quite large for computation. In such cases, we can summarize a random variable X by its Expectation $E[X]$. In some sense, expectation can be considered as the average of the distribution of that random variable. Thus expectation of a random variable X can also be called the mean, μ_X. For a discrete X, expectation is defined as

$$E[X] = \mu_X = \sum x_i p(x_i)$$

where $X = x_1, x_2, \ldots$ and p is the probability mass function.

A continuous random variable does not have a probability mass function; instead, we can make use of a *probability density function (pdf)* for that. Let us assume X is a continuous random variable, therefore X has a probability density function $f : \mathbb{R} \to \mathbb{R}$ such that f satisfies the following three conditions:

$$P(a \leq X \leq b) = \int_a^b f(x)dx \qquad \text{for any } -\infty < a \leq b < \infty$$

$$f(x) \geq 0$$

$$\int_{-\infty}^{\infty} f(x)dx = 1$$

The expectation of a continuous random variable X is calculated a little differently than its discrete counterpart. In continuous scenario, we can calculate expectation or mean using the *pdf* of X,

$$E[X] = \mu_X = \int_{-\infty}^{\infty} xf(x)\,dx$$

In addition to the expectation value, a *variance* of a distribution can help to describe the characteristics of that distribution. In simple terms, a variance is a measure of the spread of the random variable from its expectation. For a random variable X, the variance can be described as,

$$Var[X] = E[(X - E[([X])]$$

A standard deviation of a distribution can also be used instead of using the variance. A standard deviation is defined as $\sqrt{Var[X]}$ which makes it useful in practical application as standard deviation has the same dimension as the expectation $E[X]$.

2.1.1 Normal distribution

In this work, we are specially interested in the normal distribution of X. If X is a normally distributed single continuous variable, we represent that as $X \sim \mathcal{N}(\mu, \sigma^2)$ where, μ is the expectation, σ^2 is the variance and σ is the standard deviation. The probability density function of such distribution is defined as

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \quad for -\infty < x < \infty$$

In theory, X can be distributed from $-\infty$ to ∞, which is not suitable for many practical applications. In such cases, a rule of thumb is that approximately 99.7% values of X lie between 3 standard deviations (3σ) from the expectation of X. This rule can be really useful in calculations and in many applications we can safely ignore values outside of 3σ without having any significant difference in our result.

In our work, we want to estimate the probability distribution of the location of the glider. A location consists of a longitude and a latitude and we need two correlated variables to represent a location. The normal distribution we discussed above is an univariate distribution and can not be used to represent that location. We need to use a bivariate normal distribution for such configuration. It is worth mentioning that both univariate and bivariate versions hold the same properties for a normal distribution, but the form of representation is a bit different in the bivariate case. Let X and Y be two continuous random variables which represent longitude and latitude respectively. The variances of variables X and Y are defined as σ_x^2 and σ_y^2 respectively. Also, X and Y are correlated and the correlation is defined by the factor ρ. For simplicity, we are stating the random variables together as $\boldsymbol{X} = [XY]^T$. Now, the expectation $\boldsymbol{\mu_X}$, the variance matrix $\boldsymbol{\Sigma_X}$ and the probability density function $f(x, y)$ are defined as

$$\boldsymbol{\mu_X} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right] f(x, y)\, dxdy$$

$$\boldsymbol{\Sigma_X} = \begin{bmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{bmatrix}$$

$$f(x, y) = \frac{1}{2\pi\sqrt{|\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]-\boldsymbol{\mu})}$$

In a univariate normal distribution, we have used 3 standard deviations to repre-

sent 99.7% values of the random variable. Similarly, in the bivariate case we make use of a *99% confidence ellipse* of the random variable $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu_X}, \boldsymbol{\Sigma_X})$. The center of the confidence ellipse will be the expectation $\boldsymbol{\mu_X}$ and the semi-major and semi-minor axis can be calculated from the covariance matrix $\boldsymbol{\Sigma_X}$. To do such calculation, we can use the eigenvalues and eigenvectors of $\boldsymbol{\Sigma_X}$. The eigenvalue λ can be computed by the following equation

$$\begin{aligned}
\det(\boldsymbol{\Sigma_X} - \lambda \boldsymbol{I}) &= 0 \\
\det\left(\begin{bmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) &= 0 \\
\det\left(\begin{bmatrix} \sigma_x^2 - \lambda & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 - \lambda \end{bmatrix}\right) &= 0 \\
(\sigma_x^2 - \lambda)(\sigma_y^2 - \lambda) - (\rho\sigma_x\sigma_y)^2 &= 0
\end{aligned}$$

Solving the above quadratic equation will give us two eigenvalues λ_1 and λ_2. Using these eigenvalues in the following equation, we can determine the corresponding eigenvectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$.

$$(\boldsymbol{\Sigma} - \lambda_i \boldsymbol{I})\boldsymbol{v}_i = 0$$

Each eigenvalue λ_i and corresponding eigenvector $\boldsymbol{v}_i$ will define a semi-axis of the confidence ellipse. The length of a semi-axis is a function of a eigenvalue (equation 2.1) and the angle of that semi-axis is the same as the direction of the corresponding eigenvector.

$$a_i = \sqrt{c\lambda_i} \tag{2.1}$$

where, c is a constant factor determined by the level of confidence. For instance, in

case of a 99% confidence ellipse $c = 9.210$. As c is a constant factor for a particular confidence level, the larger eigenvalue will represent the major-axis and the smaller one will represent the minor-axis of the ellipse.

2.2 Glider Navigation

An electric Slocum glider is equipped with a ballast system at the front of its structure. The system can change the buoyancy of the glider to provide the necessary propulsion force. Usually the glider is adjusted in such a way that its buoyancy is neutral in ocean water. The ballast system can change that to create continuous cycle of upward and downward vertical motion of the glider. The vertical motion produces enough lift for the attached wings to take the glider in the forward direction. This unique propulsion technique makes the glider move in a sawtooth pattern.

In a typical mission, the glider is given a sequence of locations of interest, also known as waypoints. The glider starts by heading towards the first waypoint with its upward and downward cycles. Whenever the glider reaches the surface it can obtain the correct position using GPS, however the GPS signal is not accessible while under water and in some part of a mission when surfacing might not be possible. A dead reckoning system is utilized to navigate in the absence of GPS, but dynamic ocean currents can make dead reckoning unreliable. Due to the low horizontal velocity of the glider, strong ocean current can displace it in the direction of the water velocity. In general, the average displacement is about 10% of the distance travelled by the glider. The navigation system tries to compensate for this displacement, but in reality the water velocity can be very unpredictable and an adequate localization technique is required to estimate the actual displacement of the glider. In addition to that, dead reckoning displacement error is cumulative and if not corrected, can grow significantly

over time. To correct the state estimation from dead reckoning, the water depth at the estimated location can be matched with a prior elevation map of the ocean floor. The glider TAN algorithm (section 2.2.3) addresses this issue by implementing a particle filter with a water depth measurement.

The depth measurement is computed primarily by combining the data obtained from the on-board altimeter and pressure sensor. The altimeter is a narrow beam sonar mounted in the nose of the glider. The mounting is done such a way that while diving downwards the sonar points straight to the bottom of the ocean. A limitation of such configuration is that the altimeter reading as well as the depth estimation is only available during the downward motion of the glider. The glider TAN algorithm relies on dead reckoning when depth estimation is not present.

2.2.1 Dead Reckoning

In real world application of robotics specifically in mobile robotics, dead reckoning is a well known technique and often used in robot navigation system. Dead Reckoning or DR is the process of estimating the robot's current state based on previous state and known change in state over time. In its simplest form, let x_k is the last known state or *prior* state and Δx is the change in state for time Δt. The current state or *posterior* state x_{k+1} can be calculated as

$$x_{k+1} = x_k + \Delta x \tag{2.2}$$

The above equation is a state update equation. The state change Δx can be calculated from a motion model based on the configuration of the robot. In some cases, calculating Δx accurately is very challenging due to the complexity of the robot motion. In their work, Claus and Bachmayer used the on-board pressure sensor

and attitude sensor of the glider to calculate Δx for time Δt and the details can be found in [CB15]. For our work, this calculation is not relevant as we are not focusing on online path planning.

State estimation from the glider DR process works well with additional access of GPS updates. But, in the absence of such GPS updates, DR estimation becomes unreliable as the glider does not have any direct knowledge about its speed with respect to the ground. In addition to that, variable water velocity can contribute more errors in the estimation. As the errors from DR are cumulative, without any corrective measure the estimation of horizontal location becomes unusable. Especially in the highly dynamic areas, prediction of the water velocity does not match well with the actual water velocity which results in additional error in DR estimation. A proven technique to reduce DR error in glider navigation is known as Terrain-Aided Navigation or TAN.

2.2.2 Terrain-Aided Navigation

In a broader sense, most TAN algorithms use an *a priori* terrain map along with some form of measurement which can be used to match the glider location with the map. The glider's motion model is used to predict the current location and the measurement is used as a corrective update to that prediction. A Digital Elevation Model (DEM) containing the depth of the ocean floor can be used as the map. Many TAN algorithms use statistical estimators; among which we are particularly interested in the sequential importance sampling method or more commonly known particle filter.

2.2.2.1 DEM: Digital Elevation Model

The Digital Elevation Model (DEM) is an *a priori* map of the ocean terrain containing the elevation of the region of interest. Claus and Bachmayer have prepared a DEM

using the ocean survey data from the Centre for Applied Ocean Technology at the Marine Institute of Memorial University. In our work, we obtained data for the same area for implementation and validation purpose.

The data is collected by "MV Atlantica" under the Conception Bay survey. The survey was conducted near the Holyrood, Newfoundland and Labrador. We received the data in ESRI ASCII format and produced a grid of water depth for the Holyrood area. We are using the grid as the DEM and figure 2.1 is showing a portion of that grid. The DEM can be accessed using the longitude and latitude of a location with the resolution of 2 meters in both direction. We use bi-linear interpolation to access any location that does not coincide with the grid points. The depth bias of the DEM is defined [CB15] by its variance σ^2_{DEM} and can be computed for an arbitrary depth z.

$$\sigma^2_{DEM} = \frac{1}{2}\sqrt{1 + (0.023z)^2} \tag{2.3}$$

2.2.2.2 Particle filter

Particle filter is a well studied Bayesian Estimator that can work with nonlinear systems. The application of particle filters can be found in different areas of study. An in depth explanation of the particle filter has been presented in [DGA00], [AMGC02] and [Sim06]. In [FHL+03], a particle filter is implemented to estimate current location from a initial uniform distribution. Due to its capability to represent nonlinear, non Gaussian systems, particle filters are preferable for many applications.

In a particle filter based TAN algorithm, the probability of the glider location is represented by a set of particles, also known as particle cloud. Each particle is a possible location of the glider and more particles in a location indicate a higher probability of the glider being at that location. During initialization, a fixed number of particles is drawn using an importance density function. The function may vary

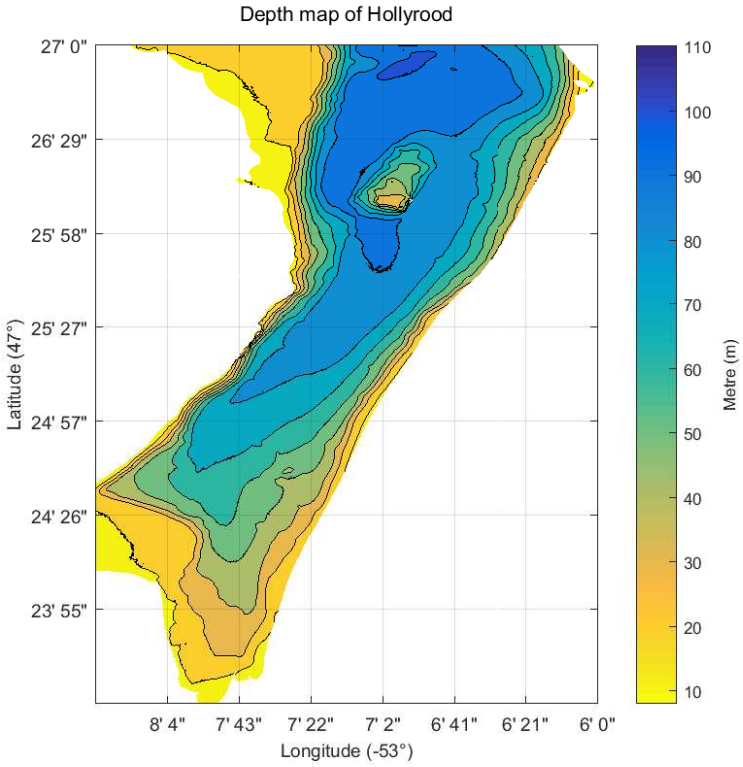


Figure 2.1: Elevation of the ocean floor in Conception Bay near Holyrood, NL

based on the particular application, but it must satisfy that more particles are drawn from the important part of the sample space using all the previous locations of the glider and all the previous measurements. Once the particles are drawn, the state update equation is applied to each particle to replicate the change of glider's state. Each particle is then assigned a measurement value from the DEM based on the particle's location. These values are compared against the actual measurement of the glider to evaluate a weight to the associated particle. The weighted mean of the particles, or in other words, the sum of the product of individual particle is the location estimate of the glider.

Choosing the right importance density function is particularly challenging while implementing particle filter. In the classic version, all the states and measurements

are needed to construct the function which is not suitable in all scenarios. Instead a suboptimal version of the importance density function can be used which requires the prior state of the particles. Using this approach can led to *particle degeneracy* where most of the probability mass is contained in few particles leaving the rest of the particles with a negligible mass. To circumvent this problem, the particles are *resampled* by disposing of low weighted particles and dividing high weighted ones into multiple particles. The resampling technique solves degeneracy, but it may cause *particle collapse* as the particle cloud becomes smaller over time and cannot correct itself anymore. A jittering can be introduced [GSS93] by adding some process noise with the particles and thus preventing the cloud to collapse.

In their work Claus and Bachmayer used a normally distributed jitter value $\boldsymbol{r}_k$ with 0 mean and σ_j standard deviation. At time k, $\{\boldsymbol{x}_{k-1}^i\}_{i=1}^N$ is the prior particle cloud and $\Delta\boldsymbol{x}_k$ is the change in state, where N is the number of particles and i is the index. The state update Equation 2.2 has been modified to include jitter value as

$$\boldsymbol{x}_k^i = \boldsymbol{x}_{k-1}^i + \Delta\boldsymbol{x}_k + \boldsymbol{r}_k \tag{2.4}$$

Using Equation 2.4, every particle's state is updated. Then the water depth measurement z_k is estimated for the time k. The probability of this depth estimation given the location of i^{th} particle can be considered as the weight $\widetilde{w}_k^i$ of that particle. The weights are then normalized by the sum of all the weights s_w. The normalized weight of the i^{th} particle is denoted by w_k^i.

$$\widetilde{w}_k^i = P(z_k|\boldsymbol{x}_k^i) \tag{2.5}$$

$$
\begin{aligned}
s_w &= \sum_{i=1}^{N} \widetilde{w}_k^i \\
w_k^i &= \frac{\widetilde{w}_k^i}{s_w}
\end{aligned}
\tag{2.6}
$$

Finally, the particles are resampled proportionally to their weight and the location estimate $\hat{\boldsymbol{x}}_k$ is calculated. The updated particle cloud $\{\boldsymbol{x}_k^i\}_{i=1}^N$ is stored to be used as the prior particle cloud in the next iteration.

$$\hat{\boldsymbol{x}}_k = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{x}_k^i$$

2.2.3 Glider TAN algorithm

A Slocum glider is equipped with all the required hardware modules to implement a particle filter based TAN algorithm. In [CB15], glider TAN algorithm is presented as an improvement of that particle filter based TAN. Although the glider has an on-board altimeter, the altitude value is only available during the downward dive cycle of the glider. This limitation demands some alteration of the base TAN algorithm. Glider TAN uses a combination of dead reckoning and jittered particle filter.

The inputs of the glider TAN algorithm are the prior particle cloud $\{\boldsymbol{x}_{k-1}^i\}_{i=1}^N$, the prior location estimate $\hat{\boldsymbol{x}}_{k-1}$ of the glider, the change in location $\Delta\boldsymbol{x}_k$ and depth measurement z_k. In the initialization stage of the algorithm, the longitude and latitude are taken from GPS reading at the initial location of the glider. A local reference frame named *Local Mission Coordinate* or LMC is created at this initial location. All the particles are set to location (0,0) in LMC, assuming the jitter value will spread the particles over time. After this initialization, the algorithm iterates for every time step k and produces an updated particle cloud $\{\boldsymbol{x}_k^i\}_{i=1}^N$ and glider's location estimate $\hat{\boldsymbol{x}}_k$ as the outputs of step k.

In step k, the algorithm first calculates the water depth z_k at the glider's current

location. The pressure sensor provides the glider's depth from the surface and the altimeter measures the glider's altitude from the ocean floor. Using these two values along with the tidal variation correction and the vertical separation of altimeter and pressure sensor, water depth is calculated.

The next part of the algorithm works with particles in a similar way to what TAN does with the exception of a dead-reckoning flag. The flag is set to true when any of the particles gets in a undesirable location such as getting outside of the DEM bound. In a way, the dead-reckoning flag governs how and when the particle filter is used by the algorithm. If the flag is set to true, the algorithm skips applying the filter and uses state update Equation 2.2. This way the location estimate $\hat{\boldsymbol{x}}_k$ only uses dead-reckoning without using the jitter $\boldsymbol{r}_k$ or the depth measurement z_k. On the other hand, when the flag is not set, the algorithm follows the usual steps of the TAN algorithm. Every particle is updated using state update Equation 2.4 with a normally distributed pseudo random jitter $\boldsymbol{r}_k$. The updated location of the particle $\boldsymbol{x}_k^i$ in LMC is then converted to longitude and latitude so that the associated water depth z_k^i can be interpolated from the DEM. This depth z_k^i is compared with the actual water depth estimation z_k to compute the probability $P(z_k|\boldsymbol{x}_k^i)$ which is the weight $\widetilde{w}_k^i$ of that particle. The comparison is done by the probability density function of a univariate normal distribution with variance σ_{DEM}^2 from equation 2.3. The weighting function can be defined as,

$$\widetilde{w}_k^i = P(z_k|\boldsymbol{x}_k^i) = \frac{1}{\sigma_{DEM,k}^2\sqrt{2\pi}} e^{-(z_k - z_{k,i})^2/2\sigma_{DEM,k}^2} \tag{2.7}$$

Once all the weights are computed, Equation 2.6 is used to normalize them appropriately. Next, the resampling is performed on the particles and the location estimate $\hat{\boldsymbol{x}}_k$ is computed using the newly resampled particle cloud. Finally, $\hat{\boldsymbol{x}}_k$ is converted to

longitude and latitude and the algorithm is ready for the next iteration.

Chapter 3

Rating

The glider TAN algorithm works well to navigate the glider to its destination, however the quality of the location estimation varies depending on which path the glider takes to its destination. For instance, in a path that goes over a flat terrain, the particle cloud can spread over a large area which can make it difficult to converge. Intuitively, a large particle cloud is more erroneous in location estimation than a smaller one. Therefore, in a path where the particle cloud remains appropriately small, the glider should localize better. The focus of this chapter is to design a method to *rank* locations in such a way that the glider TAN algorithm is expected to produce a smaller particle cloud in a well ranked location and hence the precision of location estimation will be proportional to the ranking. Subsequently, in Chapter 4 we will utilize the ranking to compute a safer path for the glider to reach its destination.

3.1 Inspecting the contribution of depth variation

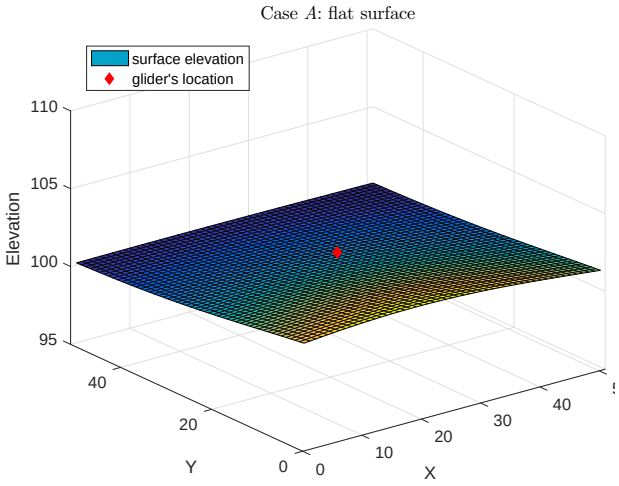
In the previous chapter, we discussed the depth measurement in Glider TAN algorithm. We are now interested in the relationship of depth measurement with the resulting particle cloud. In the resampling step of the particle filter, low-weighted

particles are discarded and high-weighted particles are replicated. In that process, when a small number of particles have higher weights, a large number of low-weighted particles are discarded. In this way, the probability mass accumulates on only those high-weighted particles which result in a smaller particle cloud. In contrast to that, when a large number of particles have similar weights, resampling cannot discard enough particles and the particle cloud becomes larger. Equation 2.7 shows that weight calculation of the particles directly relies on the variation of depth of those particles. To demonstrate the idea, let us consider the following case.

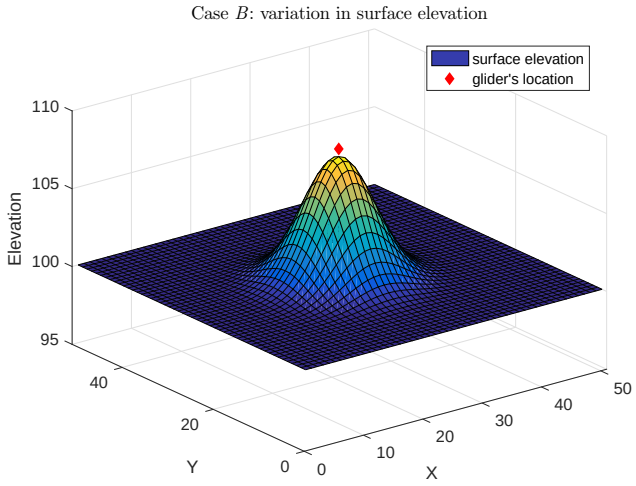
We want to compare the location estimation of the particle filter in two different locations A and B. Location A has very similar depth compared to its neighbours (Figure 3.1a) which simulates A has flat surface. On the other hand, location B has significant uniqueness in depth compared to its neighbours (Figure 3.1b). Now, we have assigned random particle clouds on both of these locations and their neighbours. Both of the initial particle clouds are identical and randomly taken from a uniform distribution (Figure 3.1c and 3.1d). Next, we have taken the depth measurements and weighted the particles in both cases. The figures clearly shows that the lack of depth variation in A has distributed the weights among a large number of particles (Figure 3.1e) whereas the weights are concentrated in case of B (Figure 3.1f). The similar effect of depth variation is reflected in the resulting particle clouds. As expected, after resampling step, particle filter produced a larger cloud in A (Figure 3.1g) and a significantly smaller cloud in B (Figure 3.1h). This suggests that the variation in depth should help the particle filter as well as the glider TAN algorithm to achieve a better location estimation.

We want to quantify the contribution of depth variation such that this concept can be utilized in path planning. We are naming such quantification *rating* and we are going to construct a function to calculate the rating without requiring to run particle

Surface elevation of the ocean with the location of the glider

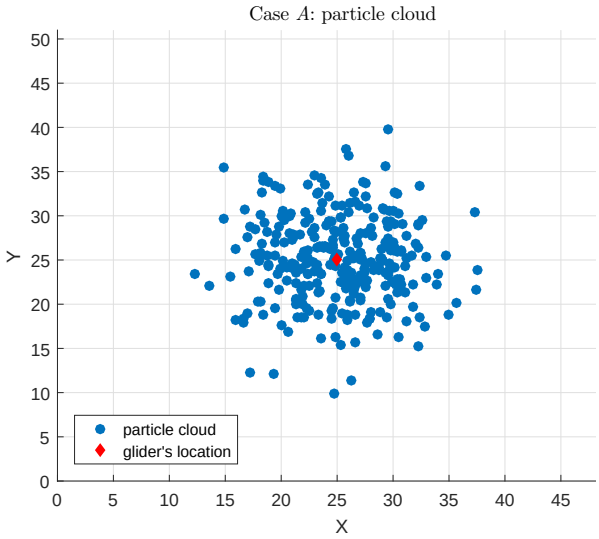


(a)

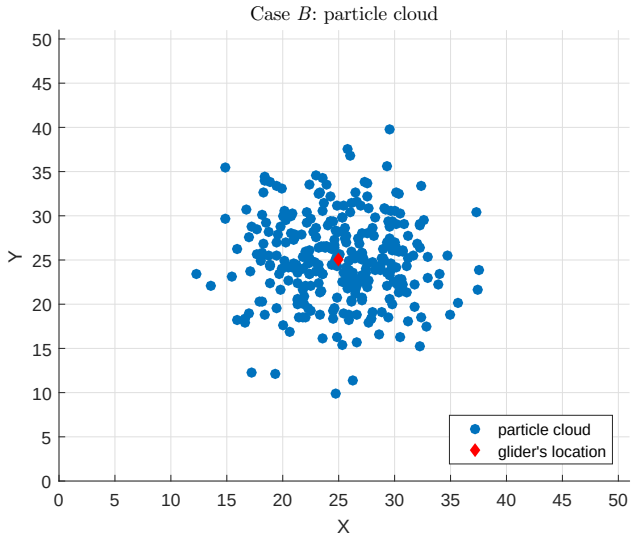


(b)

Initial particle cloud

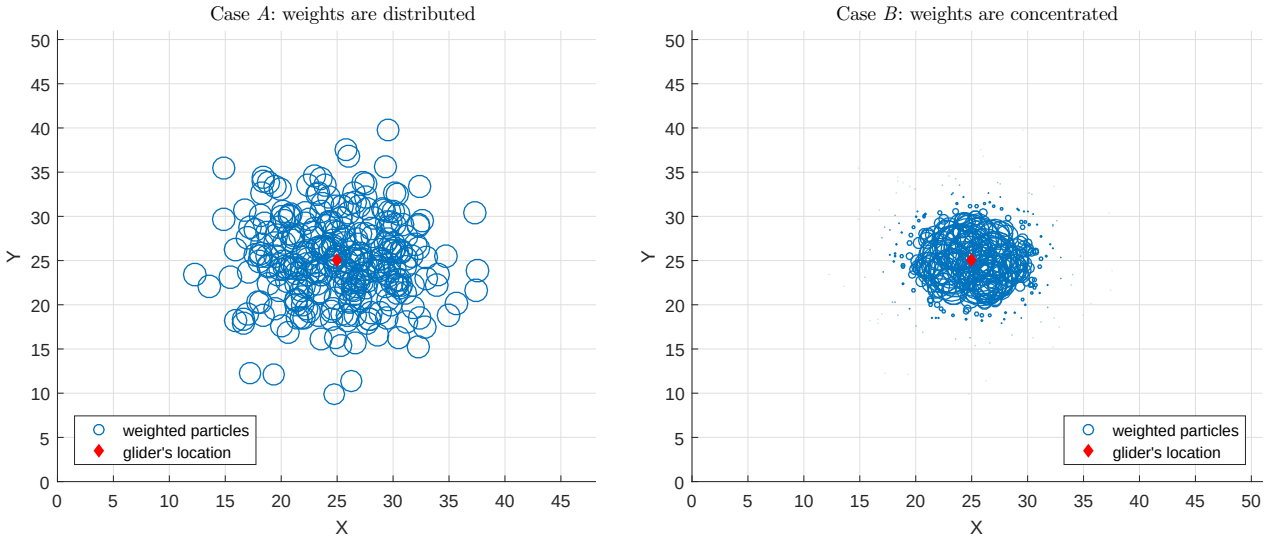


(c)



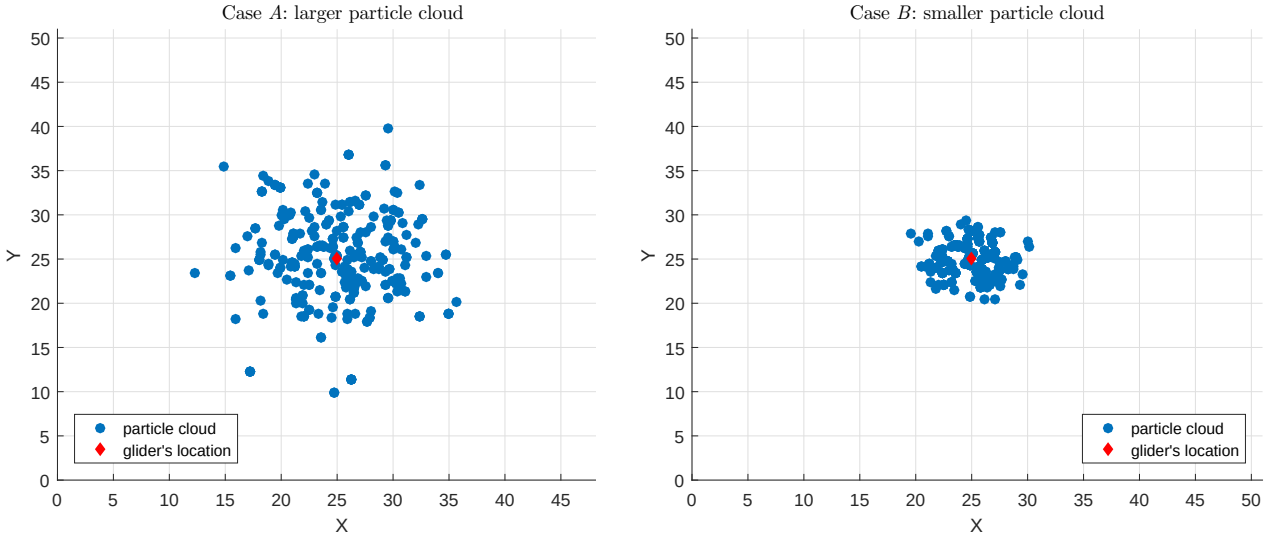
(d)

Weighted particles, larger circle indicates higher weight



(e) (f)

Updated particle cloud which represents the location estimation



(g) (h)

Figure 3.1: The impact of depth variation on the particle cloud size

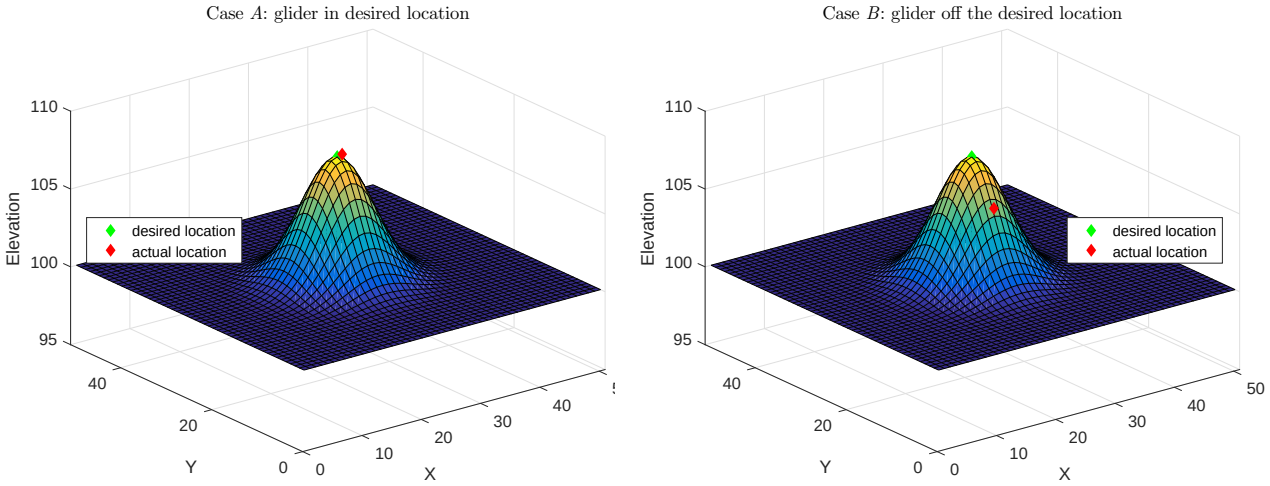
filter explicitly. To do so, first we need to decide what we should rate: a location or an area in the map.

3.1.1 Effectiveness of rating: a point vs an area

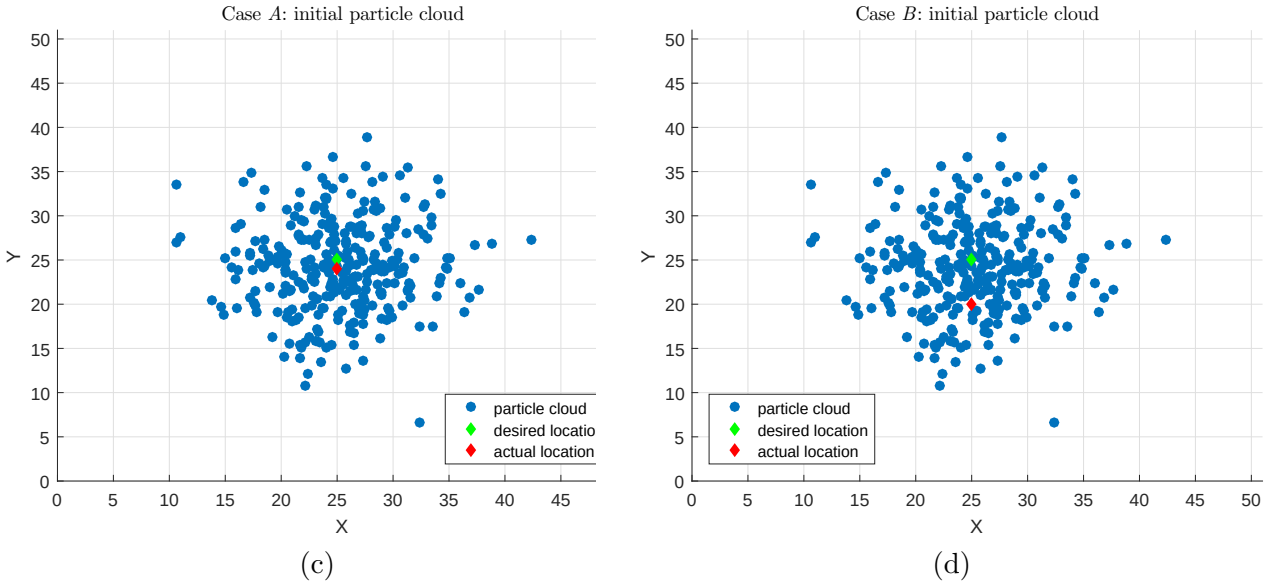
From the glider TAN algorithm, we have learned that only one depth measurement is taken in a given iteration and that depth is associated with the actual location of the glider in that particular iteration. By design, the rating must be pre-computable and the actual location of the glider will always be an unknown; therefore calculating the rating using the glider's actual location is not a feasible option. Although we can compute the rating of an arbitrary location assuming the glider will be on that location, this approach does not give a good guarantee to reduce uncertainty. For instance, we can rate a location as good and expect the glider to visit and take a measurement at that location, but in reality, there is a good possibility that the glider may fail to reach that exact location and end up being on one of its neighbours. Taking a measurement at that neighbour may not be as useful as the desired location and the purpose of the rating function will be nullified in such cases.

To illustrate the above mentioned problem, we have run a particle filter twice on the same area for a different location of the glider. In both cases, the glider was expected to reach a location (denoted by the green diamond in Figure 3.2a) in the area. In the first case, we assumed the glider was able to reach that location and the resulting particle cloud has reduced adequately (Figure 3.2g). However, in the second case we assumed the glider was slightly displaced by ocean current and took a measurement at a neighbouring location (denoted by red diamond in Figure 3.2b). In this case, the particle filter did not estimate as good as before and the resulting particle cloud is larger (Figure 3.2h) than the one in previous case. We can also see that estimated location closely matches the glider's actual location in the first case,

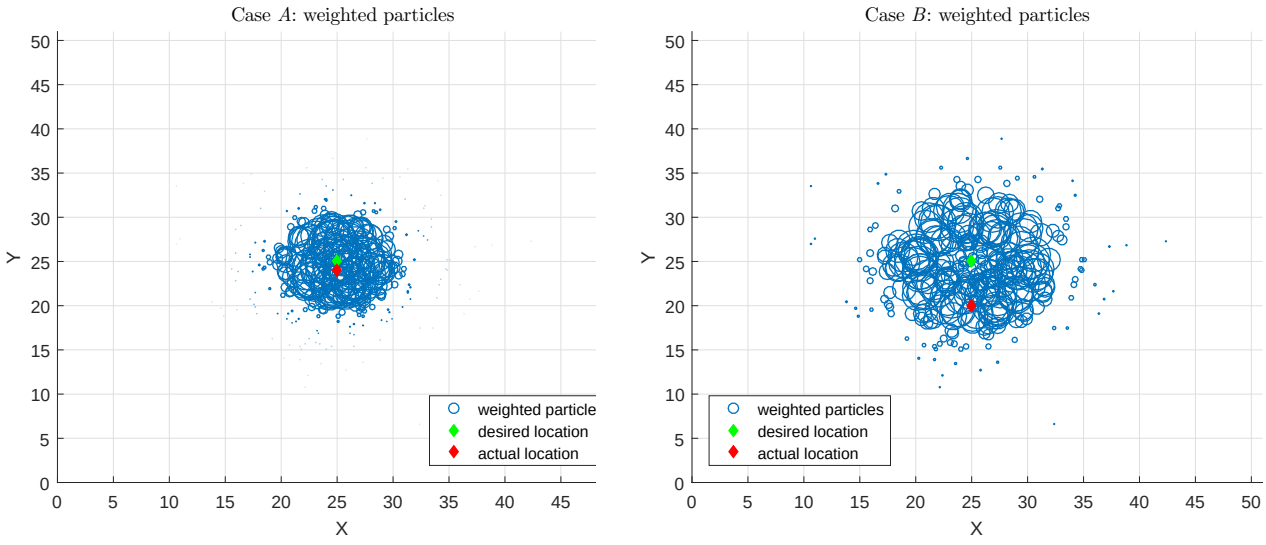
Surface elevation with the desired and the actual location of the glider



Initial particle cloud with desired and actual location

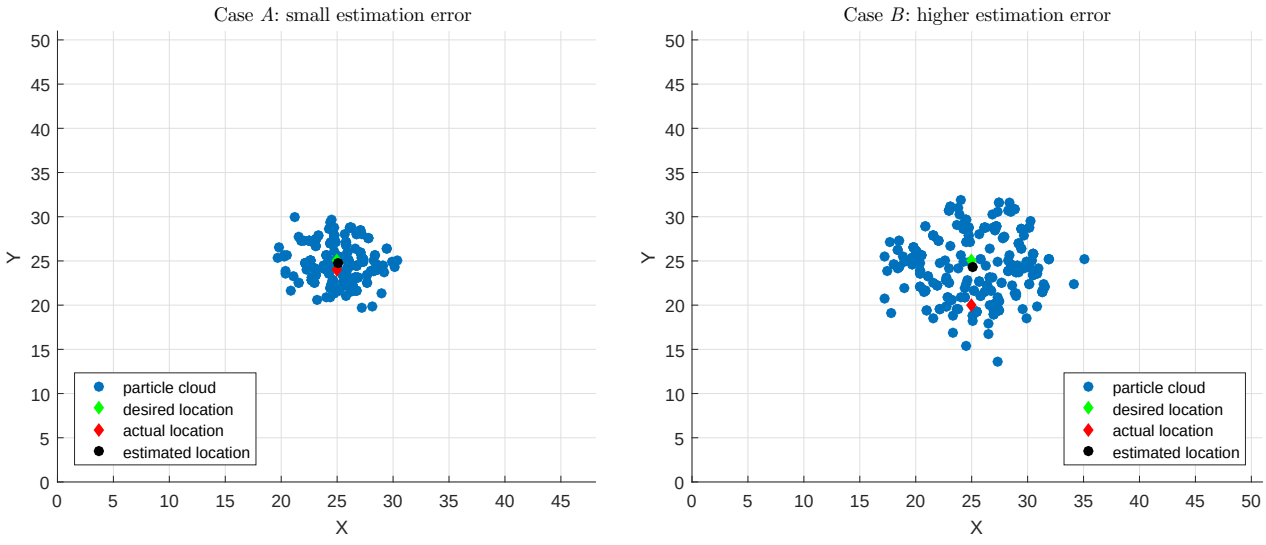


Weighted particles, larger circle indicates higher weight



(e) (f)

Updated particle cloud with actual and estimated location



(g) (h)

Figure 3.2: Comparing the effectiveness of rating of a location and an area

where as there is noticeable difference between them in the second one.

Considering the above example, we have reached the conclusion that computing the rating for a single location is not well suited for real world implementation. Rather, we have designed the rating function to rate a given area such that the rating value represents the overall quality of locations in that area. We are representing such area with a probability distribution and the following section describes more about that distribution.

3.2 Representing an area for rating

In the preceding section, we have shown that rating a single location is not appropriate for our real world application; a glider can attempt to reach a certain location and may end up at that location or any other nearby location. Assuming, the probability of the glider's actual location is highest in the attempted location and the probability decreases as we move further away from that location, we can use bivariate normal distribution to represent the location probability of the glider. In our work, we are using $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ to represent such distribution. $\boldsymbol{X}$ is a two dimensional random variable containing the longitude and latitude and can be defined as $\boldsymbol{X} = [XY]^T$, where X and Y are in global coordinate system and corresponds to the longitude and latitude respectively. The variable $\boldsymbol{\mu}$ denotes the mean of the distribution and represents the location which the glider is attempting to reach. $\boldsymbol{\Sigma}$ is the covariance matrix of the distribution, $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ corresponds to the area which the glider is expected to reach. We are assuming that the actual position of the glider will be anywhere within this area with higher probability of being at location $\boldsymbol{\mu}$ and lower probabilities at distant locations from $\boldsymbol{\mu}$.

3.3 Constructing the rating

The purpose of the rating is to take the glider's location distribution $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$[1] as input along with a digital elevation model and to produce the expected location distribution $\boldsymbol{X'} \sim \mathcal{N}(\boldsymbol{\mu'}, \boldsymbol{\Sigma'})$ after the glider takes an depth measurement. We are naming the digital elevation model as map_{depth}.

Now, from $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$, we can determine the probability of the glider being on an arbitrary location. Let us call these locations *cells* and represent them using $\boldsymbol{i}$, where $\boldsymbol{i}$ ranges over all relevant cells. In our work, a cell is an arbitrary location near $\boldsymbol{\mu}$ such that probability of $\boldsymbol{i}$ is not negligible.

$$\boldsymbol{i} = \begin{bmatrix} i_x \\ \\ i_y \end{bmatrix}$$

The coordinate $[i_x i_y]^T$ of a cell $\boldsymbol{i}$ determines the probability of glider reaching that cell when aiming for $\boldsymbol{\mu}$. As $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is normally distributed, we can show that the probability of cell $\boldsymbol{i}$ depends on the distance between $\boldsymbol{\mu}$ and the cell $\boldsymbol{i}$. If we move further away from $\boldsymbol{\mu}$ the probability decreases. Similarly, the probability increases if we move closer to $\boldsymbol{\mu}$ and the highest probability is contained in the cell located at $\boldsymbol{\mu}$. To calculate these probabilities, we need a probability density function. For the bivariate normal distribution, the probability density function can be defined as follows

$$P(i) = \frac{1}{2\pi\sqrt{|\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\boldsymbol{i}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{i}-\boldsymbol{\mu})} \tag{3.1}$$

where $P(i)$ is the probability of cell $\boldsymbol{i}$.

[1] It is important to have $\boldsymbol{\mu}$ in the global coordinate system, but the rest of the calculation can be done using local coordinates. To avoid unnecessary calculation, we work in a local coordinate system that has the origin set to $\boldsymbol{\mu}$. This is not required for the rating process, but using local coordinates eliminates unnecessary computation.

We can look up the depth measurement z_i of cell $\boldsymbol{i}$ using map_{depth}. This implies that if the glider is on cell $\boldsymbol{i}$ and takes a measurement, it should measure z_i. Using the z_i, we can formulate the probability of measuring an arbitrary z at cell $\boldsymbol{i}$. Ideally, this probability $P(z|i)$ should simply be defined as

$$P(z|i) = \begin{cases} 1, & \text{if } z = z_i \\ 0, & \text{otherwise} \end{cases}$$

But in a real application, the above definition of $P(z|i)$ is not appropriate. The depth information in map_{depth} may contain some error. In addition to that, the depth measurement of the glider can be contaminated with instrument noise. We cannot correct the error contained in map_{depth}, instead we can model this error and assign the probability $P(z|i)$ using that error model. We have used the error model (Equation 2.7) used in gTAN algorithm, defining $P(z|i)$ as

$$P(z|i) = \frac{1}{\sigma^2_{DEM}\sqrt{2\pi}} e^{-(z-z_i)^2/2\sigma^2_{DEM}} \tag{3.2}$$

where σ^2_{DEM} is the depth variance in map_{depth}, and can be calculated using Equation 2.3. The instrument noise can also be modeled and integrated here, but the model will vary depending on the type and quality of instrument used to take the measurements. For simplicity, we are assuming instrument noise to be zero.

Now, $P(i)$ in equation 3.1 provides the probability of being on a cell $\boldsymbol{i}$ and $P(z|i)$ in equation 3.2 provides the probability of measuring z in that cell $\boldsymbol{i}$. Combining these two probabilities, we can determine the overall probability of measuring z

$$P(z) = \sum_i P(i) \cdot P(z|i) \tag{3.3}$$

where $P(z)$ is the probability of measuring z. By applying Bayes Rule with Equations 3.1, 3.2 and 3.3, we can obtain the following

$$P(i|z) = P(z|i) \cdot P(i)/P(z) \tag{3.4}$$

where $P(i|z)$ is the probability of cell $\boldsymbol{i}$ after measuring z. Let us consider that at a certain time the glider takes a depth measurement z and we are interested in the impact of this measurement on the probability distribution of the location of the glider. In other words, how the probability $P(i)$ of cell $\boldsymbol{i}$ will change after measuring a depth z and $P(i|z)$ in equation 3.4 will give us that answer. Building on this concept, we can combine $P(i|z)$ for each cell near $\boldsymbol{\mu}$ and collectively they will give us the expected estimation of $\boldsymbol{X'}$ for an arbitrary measurement z,

$$\boldsymbol{\mu'}_z = E(\boldsymbol{X'}|z) = \sum_i i \cdot P(i|z) \tag{3.5}$$

Here, $E(\bar{\boldsymbol{X}}|z)$ is the expectation of the random variable $\bar{\boldsymbol{X}}$ denoting glider's location after measuring a depth z. Similarly, we can define the corresponding covariance matrix $\boldsymbol{\Sigma'}_z$ to represent the probability distribution after measuring a depth z. There are several ways to define a covariance matrix; we are defining $\boldsymbol{\Sigma'}_z$ as follows

$$\begin{aligned} \boldsymbol{\Sigma'}_z &= E((i - \boldsymbol{\mu'}_z)(i - \boldsymbol{\mu'}_z)^T) \\ &= \sum_i (\begin{bmatrix} i_x \\ i_y \end{bmatrix} - \begin{bmatrix} \bar{\mu}_{x|z} \\ \bar{\mu}_{y|z} \end{bmatrix}) \cdot P(i|z) \end{aligned} \tag{3.6}$$

It is worth mentioning that the computation of a covariance matrix in floating point arithmetic is not always numerically stable and can lead to catastrophic cancellations. Here, we are not concerned about the numerical instability and it will be addressed in the implementation.

It is clear that we can estimate the probability distribution after measuring a depth z by using equations 3.5 and 3.6. However, as we have stated earlier, there is no way of knowing the actual value of z beforehand with enough certainty. This does not concern the online algorithms like gTAN as they have the freedom to access and use the actual measurement and can act accordingly. On the contrary, we do not know on which cell the glider will be and what measurement it will get. We need to overcome this limitation such that the impacts of all possible measurements are covered. We have devised a solution by combining the $\boldsymbol{\mu'}_z$ and $\boldsymbol{\Sigma'}_z$ for all possible z values. In this process, taking expectation should be ideal. Because we can easily utilize $P(z)$ to weight the values and a weighted average can suppress irrelevant values of z. The calculation of the combining process is as follows

$$\boldsymbol{\mu'} = \int_0^\infty \boldsymbol{\mu'}_z \cdot P(z)\, dz \tag{3.7}$$

$$\boldsymbol{\Sigma'} = \int_0^\infty \boldsymbol{\Sigma'}_z \cdot P(z)\, dz \tag{3.8}$$

Here, $\boldsymbol{\mu'}$ is the expected mean that represents the possible location of the glider, and $\boldsymbol{\Sigma'}$ is the expected covariance matrix that represents the expected probability distribution. Collectively, $\boldsymbol{\mu'}$ and $\boldsymbol{\Sigma'}$ act as the expectation of the probability distribution after taking any measurement and we can denote it as

$$\boldsymbol{\mu'}, \boldsymbol{\Sigma'} = \mathit{Rating}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

An important remark about $\boldsymbol{\Sigma'}$ is that it should not be considered equivalent to the *Updated* probability distribution in the gTAN algorithm. In the gTAN algorithm, the glider takes a single measurement and updates the probability distribution based on that measurement. On the contrary, $\boldsymbol{\Sigma'}$ is a expectation of all the *Updated*

probabilities for all possible measurements.

3.4 Implementation of the rating

The rating process we constructed in the previous section can estimate the usefulness of taking a measurement in a certain area. But like many mathematical calculations, implementing the rating in a computer using floating point arithmetic requires some adjustment. In this section, we will present a few tweaking to eliminate unnecessary calculation as well as to improve the efficiency of those calculations.

We want to start with the calculation of the initial probability of a cell, $P(i)$. In Equation 3.1, we are using the probability density function of a bivariate normal distribution to calculate $P(i)$. In theory, every cell has some amount of the probability mass, but some of them have very low probability and may not have any significant contribution in the rating calculation. Some cells are so far away from $\boldsymbol{\mu}$ that their probability cannot even be stored using floating point arithmetic. In our implementation, we have found that excluding cells with negligible probability does not affect the rating and improves the efficiency of the calculation. A good way of doing such exclusion is using confidence ellipse [HR84] of the initial distribution. For a bivariate normal distribution, we can use Chi-Squared (χ^2) distribution with 2 degree of freedom [Sci17]. The semi-major axis a and semi-minor axis b of the confidence ellipse can be defined as

$$a = \sqrt{c\lambda_1}$$

$$b = \sqrt{c\lambda_2}$$

where c is the constant factor from χ^2 distribution, λ_1 and λ_2 are the eigenvalues

of the covariance matrix $\mathbf{\Sigma}$. The respective eigenvectors will determine the angle of the semi-major and semi minor axes. The value of c corresponds to the level of confidence. For instance, $c = 9.210$ will give us 99% confidence ellipse [Sci17]. From this point onwards, we are going to represent the distributions with the associated 99% confidence ellipse. Let us name the area inside such ellipse Ar. Now, we can use Equation 3.1 only for those cells which are in Ar and ignore the other cells as negligible. In addition to that, we need to normalize $P(i)$ so that the sum of all probabilities becomes one. We can define a normalization factor, η as below and use η to normalize $P(i)$.

$$\eta = \frac{1}{\sum_{i \in Ar} P(i)}$$

$$P(i) = \eta \cdot P(i)$$

Another good candidate for implementation specific improvement is the integration used in Equations 3.7 and 3.8. In both cases, we are integrating with respect to z in the interval $[0, \infty]$. Ideally, we want to integrate in that interval to ensure every possible value of z is evaluated. Fortunately in our case, we can determine all possible z values in Ar from map_{depth} and we can utilize this information to reduce the interval. To do so, we can extract the minimum and maximum z values in Ar and expand them using the standard deviation(σ_{DEM}) of map_{depth}. Now, our reduced interval for integration is $[z_{min}, z_{max}]$ where,

$$z_{min} = \min_{\forall i \in Ar} z_i - 3 \cdot \sigma_{DEM}$$

$$z_{max} = \max_{\forall i \in Ar} z_i + 3 \cdot \sigma_{DEM}$$

Using the above mentioned improvement techniques, we have implemented our

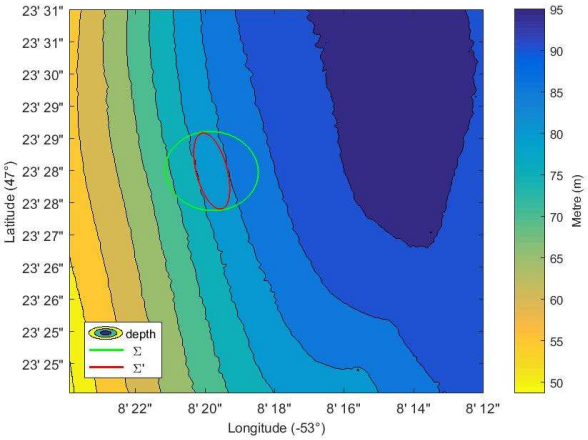
rating function. We have chosen Matlab as our programming tool due to the well known efficiency of Matlab in mathematical calculations and for compatibility with existing software written by the Autonomous Oceans Systems Lab. In the following sections, we present results obtained by running the rating algorithm on the map of Conception Bay near Holyrood, Newfoundland.

3.4.1 Rating computed on the Holyrood data

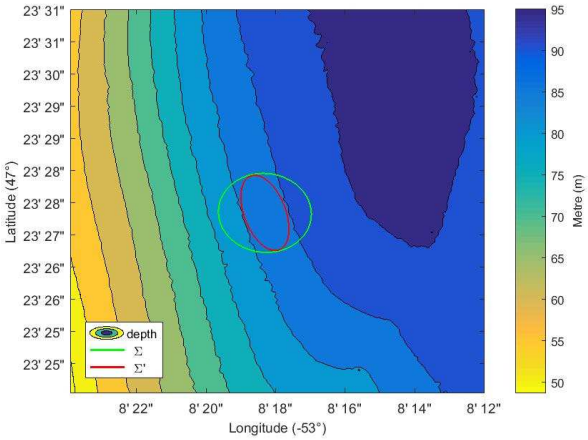
In our first test case, we computed the rating in different areas of the ocean floor of Conception Bay near Holyrood, NL. To compare the results, we have used the same initial distribution $\mathbf{\Sigma} = \left[\begin{smallmatrix} 29.85 & -0.86 \\ -0.86 & 25.15 \end{smallmatrix}\right]$ so that only the usefulness of those areas can contribute in the resulted distribution $\mathbf{\Sigma'}$. Figure 3.3 shows the result of rating calculation for four different areas which were chosen by decreasing variation of surface elevation.

To represent both the initial distribution $\mathbf{\Sigma}$ and expected distribution $\mathbf{\Sigma'}$, we calculated the 99% confidence ellipse and plotted them in green and blue color respectively. In Figure 3.3, area a has the highest variation in surface elevation where as area d has the lowest one and area b and c lie in between them. As we can see, the resulted expected distribution $\mathbf{\Sigma'}$ increases with the decreasing variation of elevation. We can conclude that with increasing variation in elevation, the expected distribution should become smaller after taking a measurement in the area which complies with our rating concept. Table 3.1 includes some details from running this test case.

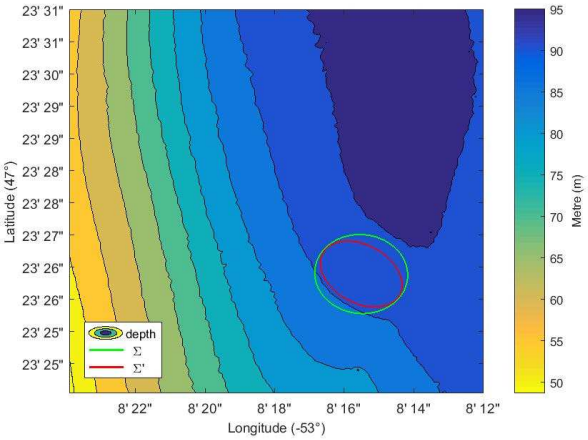
In our next test case, we wanted to show the contribution of different initial distribution $\mathbf{\Sigma}$ in the rating process. We calculated the expected distribution $\mathbf{\Sigma'}$ for the same area a from the previous test case, but in each calculation we used different arbitrary $\mathbf{\Sigma}$s. We chose four $\mathbf{\Sigma}$s such a way that each of them is different in size or shape from the others. Figure 3.4 shows the result from this test case. Like the



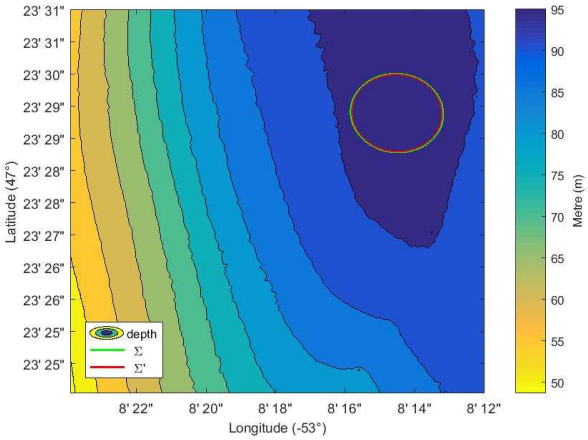
(a) Rating result at $(47°23'28", -53°8'19.5")$



(b) Rating result at $47°23'28", -53°8'19.5"$



(c) Rating result at $47°23'28", -53°8'19.5"$



(d) Rating result at $47°23'28", -53°8'19.5"$

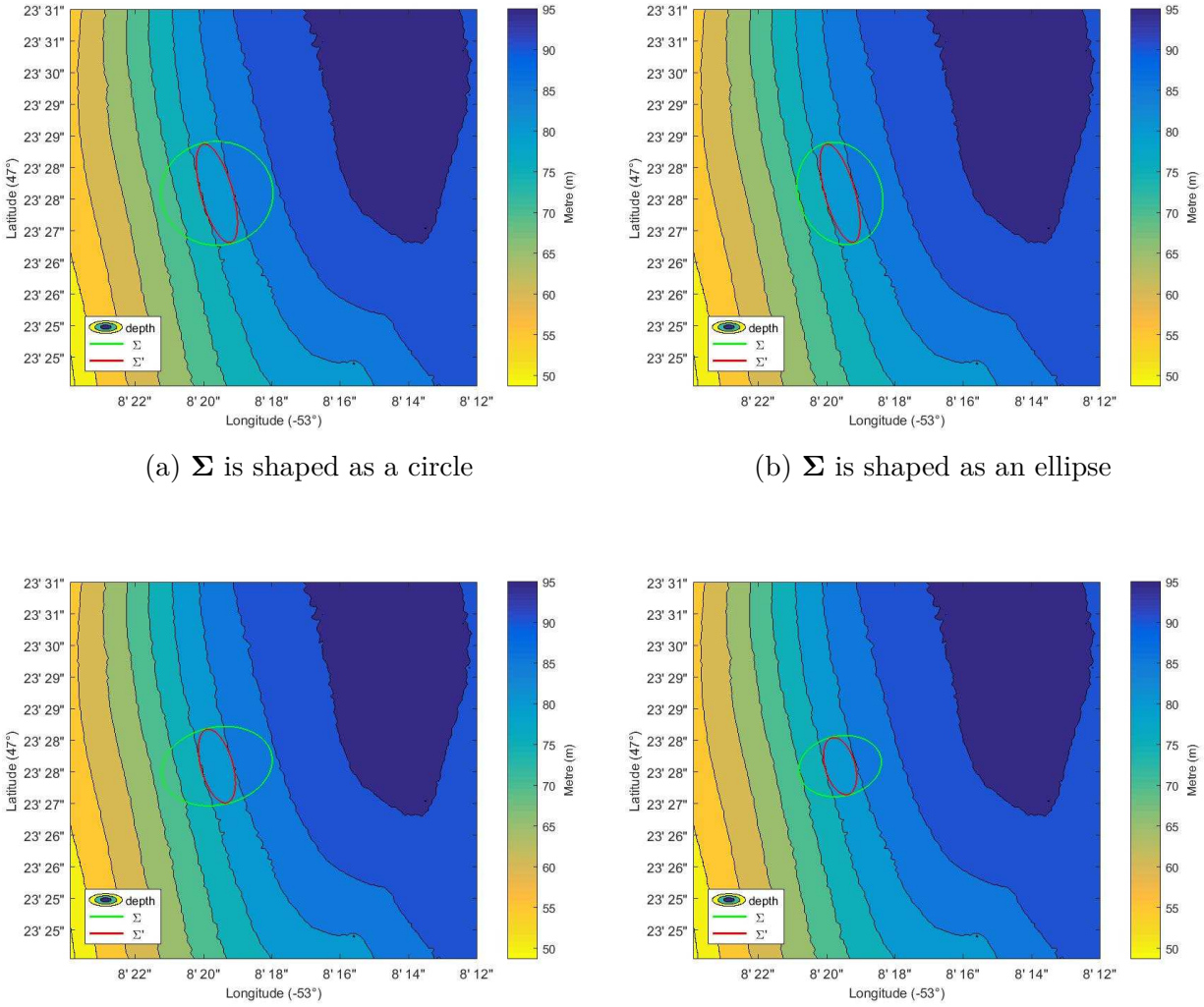
Figure 3.3: Comparing the rating result in different areas of the ocean near Holyrood, NL using 99% confidence ellipse

Area	$\boldsymbol{\Sigma}$	99% ellipse (m^2)	$\boldsymbol{\Sigma'}$	99% ellipse (m^2)
a	$\begin{bmatrix} 29.85 & -0.86 \\ -0.86 & 25.15 \end{bmatrix}$	868.02	$\begin{bmatrix} 24.28 & -4.82 \\ -4.82 & 4.25 \end{bmatrix}$	96.97
b	$\begin{bmatrix} 29.85 & -0.86 \\ -0.86 & 25.15 \end{bmatrix}$	868.02	$\begin{bmatrix} 23.58 & -5.59 \\ -5.59 & 7.37 \end{bmatrix}$	174.20
c	$\begin{bmatrix} 29.85 & -0.86 \\ -0.86 & 25.15 \end{bmatrix}$	868.02	$\begin{bmatrix} 17.63 & -5.51 \\ -5.51 & 18.38 \end{bmatrix}$	377.55
d	$\begin{bmatrix} 29.85 & -0.86 \\ -0.86 & 25.15 \end{bmatrix}$	868.02	$\begin{bmatrix} 25.18 & -0.69 \\ -0.69 & 21.05 \end{bmatrix}$	690.92

Table 3.1: Comparing the rating result in different areas of the ocean near Holyrood, NL using 99% confidence ellipse

previous case, green ellipses represent initial distribution $\boldsymbol{\Sigma}$ and blue ellipses represent expected distribution $\boldsymbol{\Sigma'}$.

In figure 3.4a, we chose a circular initial distribution with $\boldsymbol{\Sigma} = \left[\begin{smallmatrix} 45.00 & 0.00 \\ 0.00 & 45.00 \end{smallmatrix}\right]$. The resulted $\boldsymbol{\Sigma'}$ is an ellipse with its semi-major axis aligned with the contour line of the surface. In the second figure 3.4b, we chose an ellipse shaped initial distribution with $\boldsymbol{\Sigma} = \left[\begin{smallmatrix} 43.66 & -5.00 \\ -5.00 & 26.34 \end{smallmatrix}\right]$. Similar to the previous one, the resulted $\boldsymbol{\Sigma'}$ matches the contour line as well. To examine the impact of the semi-major axis of the initial distribution, in the next figure 3.4c we kept the size of the initial distribution same, but rotated that by 90° with $\boldsymbol{\Sigma} = \left[\begin{smallmatrix} 26.34 & 5.00 \\ 5.00 & 43.66 \end{smallmatrix}\right]$. As expected, the resulted $\boldsymbol{\Sigma'}$ matches the contour line, but it also decreased in size. In the last figure 3.4d, we maintained the same shape and rotation of the initial distribution while reducing the size with $\boldsymbol{\Sigma} = \left[\begin{smallmatrix} 15.67 & 2.50 \\ 2.50 & 24.33 \end{smallmatrix}\right]$. Just like the preceding three figures, the resulted $\boldsymbol{\Sigma}$ matched the contour line. All four of the figures are consistent in aligning with the contour lines of the surface, which



(a) $\mathbf{\Sigma}$ is shaped as a circle

(b) $\mathbf{\Sigma}$ is shaped as an ellipse

(c) $\mathbf{\Sigma}$ is rotated by 90°

(d) A relatively smaller $\mathbf{\Sigma}$

Figure 3.4: Comparing the result of rating calculation for the same area with different initial distribution varying in size, shape and rotation

is expected as the depth is effectively same along those lines. Table 3.2 shows some details from running this test case.

3.4.1.1 Rating map: a visual representation of depth variation

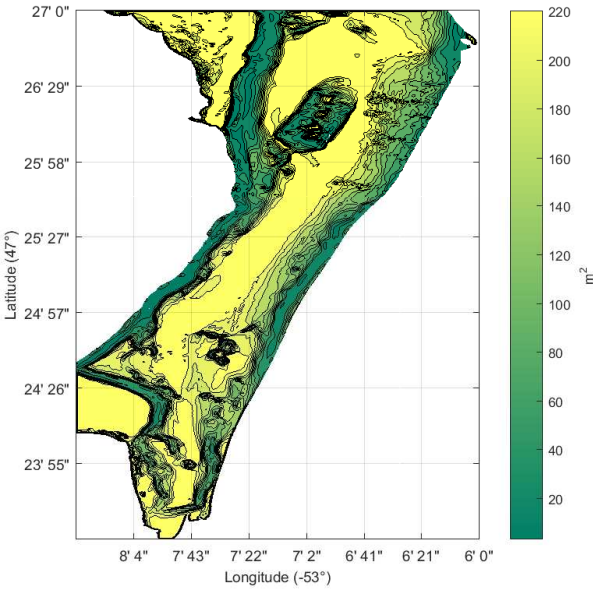
We designed the rating function to calculate the expected distribution after taking a measurement, which can later be used in path planning. But rating can also be

Figure	$\boldsymbol{\Sigma}$	99% ellipse(m^2)	$\boldsymbol{\Sigma'}$	99% ellipse(m^2)
3.4a	$\begin{bmatrix} 29.85 & -0.86 \\ -0.86 & 25.15 \end{bmatrix}$	1302.03	$\begin{bmatrix} 24.28 & -4.82 \\ -4.82 & 4.25 \end{bmatrix}$	106.23
3.4b	$\begin{bmatrix} 29.85 & -0.86 \\ -0.86 & 25.15 \end{bmatrix}$	1302.03	$\begin{bmatrix} 23.58 & -5.59 \\ -5.59 & 7.37 \end{bmatrix}$	95.42
3.4c	$\begin{bmatrix} 29.85 & -0.86 \\ -0.86 & 25.15 \end{bmatrix}$	1302.03	$\begin{bmatrix} 17.63 & -5.51 \\ -5.51 & 18.38 \end{bmatrix}$	105.62
3.4d	$\begin{bmatrix} 29.85 & -0.86 \\ -0.86 & 25.15 \end{bmatrix}$	723.35	$\begin{bmatrix} 25.18 & -0.69 \\ -0.69 & 21.05 \end{bmatrix}$	94.14

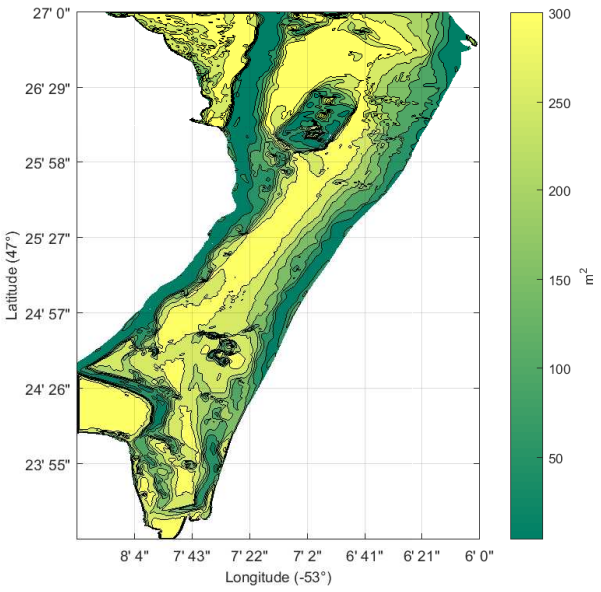
Table 3.2: Comparing the result of rating calculation for the same area with different initial distribution varying in size, shape and rotation

utilized to visualize the variation in usefulness for the localization of an entire area of the ocean. We can take an arbitrary initial distribution $\boldsymbol{\Sigma}$ and calculate rating for all the areas in the given elevation map with that $\boldsymbol{\Sigma}$. Thus we will have the corresponding expected distribution $\boldsymbol{\Sigma'}$s for those areas. Then we can plot the size of the $\boldsymbol{\Sigma'}$s in the associated position of a contour map and we named such map as a *rating map* of that particular $\boldsymbol{\Sigma}$. The current implementation of the rating process is efficient enough to compute a rating map without taking a significant time.

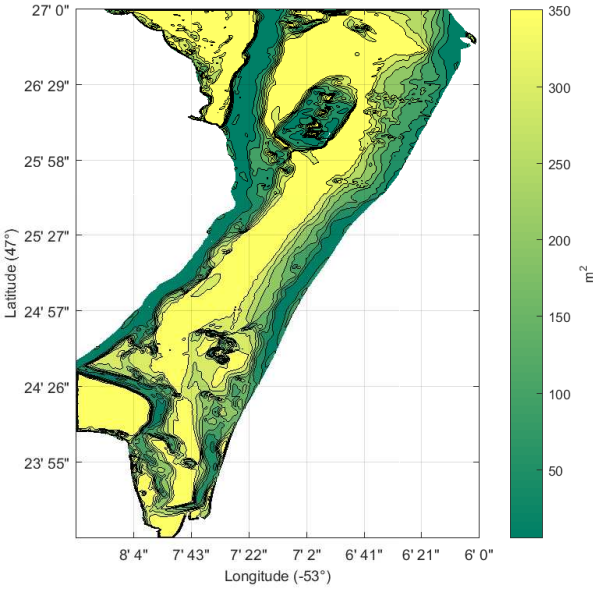
Precomputed rating maps can be useful as a reference for path planning, but creating rating maps for all possible $\boldsymbol{\Sigma}$ is neither an efficient nor a feasible solution. But usually the ocean floor is smooth enough that similar values of $\boldsymbol{\Sigma}$ give similar ratings. Thus, a rating map for some representative $\boldsymbol{\Sigma}$ is a good way to visually assess the usefulness of a region in the ocean. Figure 3.5 shows the rating maps for four different $\boldsymbol{\Sigma}$ values. The result of our rating process is a two dimensional distribution



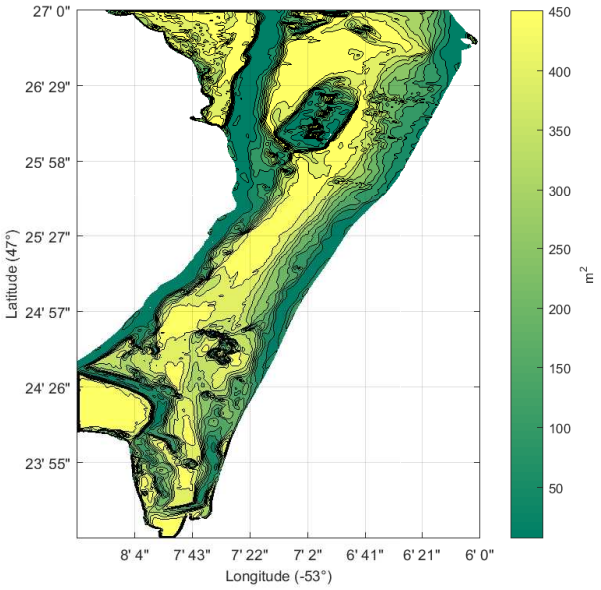
(a) Rating map for $\Sigma = \left[\begin{smallmatrix} 15 & 0 \\ 0 & 15 \end{smallmatrix}\right]$



(b) Rating map for $\Sigma = \left[\begin{smallmatrix} 20 & 0 \\ 0 & 20 \end{smallmatrix}\right]$



(c) Rating map for $\Sigma = \left[\begin{smallmatrix} 25 & 0 \\ 0 & 25 \end{smallmatrix}\right]$



(d) Rating map for $\Sigma = \left[\begin{smallmatrix} 30 & 0 \\ 0 & 30 \end{smallmatrix}\right]$

Figure 3.5: A visual representation of the rating of the entire region of the ocean shown in Figure 2.1 using rating maps

which is hard to plot visually for this purpose. So, we resorted to the 99% confidence ellipse and Figure 3.5 is showing the total area covered by the confidence ellipses. This representation of the rating maps is not useful for optimal path planning, however they are a good way to understand which regions are more useful for glider TAN algorithm to work better. Specially while manually planning a path by hand, the rating maps can be really helpful.

3.4.2 Comparing the result of rating using Holyrood data

We want to validate the expected distribution which results from the rating process. To do such validation, we have set up two experiments which will compare the expected distribution with the simulated distribution of the glider's location estimate. In the first experiment, we compare the expected distribution with the location estimation of a particle filter and the second experiment does a similar comparison with a simplified version of glider TAN algorithm.

3.4.2.1 Comparing rating with estimation from a particle filter

The goal of this experiment is to measure the accuracy of the expected distribution from rating function with respect to the estimation of particle filter. We took several arbitrary locations $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \ldots \boldsymbol{x}_k$ from the map_{depth} and each location $\boldsymbol{x}_i$, is assigned an initial random normal distribution $\boldsymbol{X}_i \sim \mathcal{N}(\boldsymbol{x}_i, \boldsymbol{\Sigma}_i)$. First we apply the rating function to compute $\boldsymbol{\Sigma'}_i$ for each location $\boldsymbol{x}_i$ and calculate the area (Ar_i^{rating}) of the 99% confidence ellipse of $\boldsymbol{\Sigma'}_i$.

In Matlab we have implemented a particle filter similar to the one used in glider TAN algorithm. We used this implementation to simulate the result of using the particle filter for each $\boldsymbol{x}_i$. In the simulation process, we created a particle cloud by taking particles from $\boldsymbol{X}_i$. Next, we randomly picked a particle in the particle cloud

to be the actual location of the glider. This step replicates the real world uncertainty of gliders actual location. Then we ran the particle filter as usual and compute the updated particle cloud. Lastly, we calculated the area (Ar_i^{pf}) of the smallest polygon containing the updated particle cloud. This process was repeated n number of times and the root mean square value $\hat{Ar}_i^{pf}$ was computed.

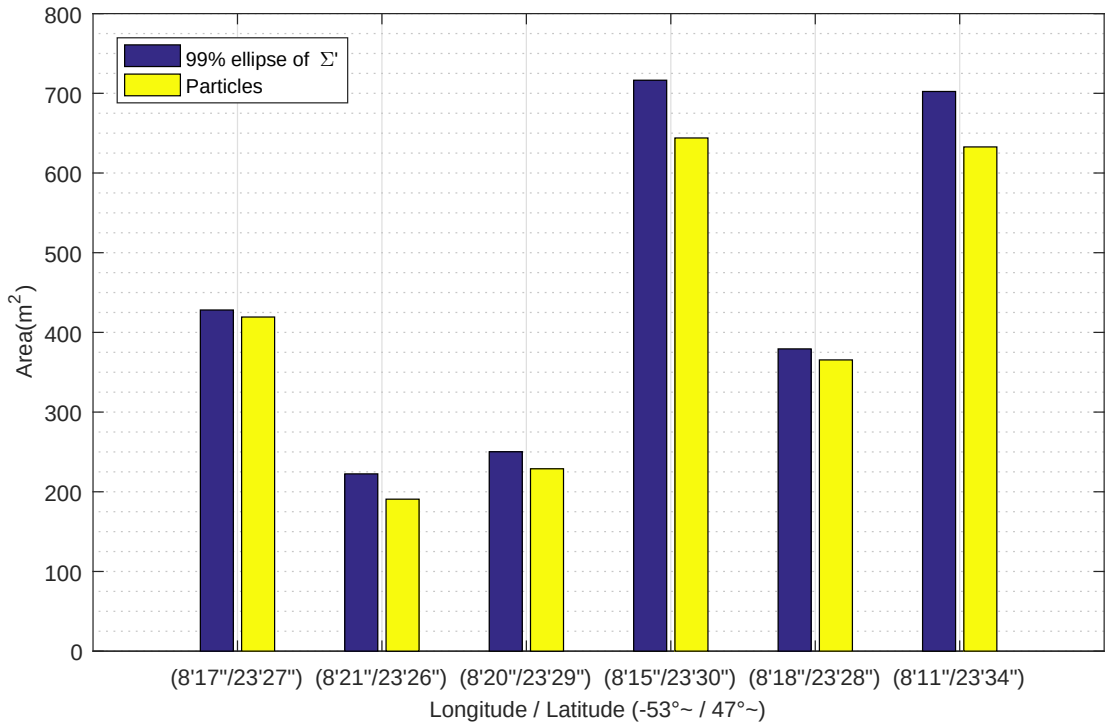


Figure 3.6: A comparison between the expected distribution from the rating process and location estimation from particle filter simulation

We have plotted our result from the experiment in Figure 3.6. The plot shows that the rating function complies with the RMS value from the particle filter simulation. An observation of the plot is that Ar_i^{rating} tends to be a slight overestimation of $\hat{Ar}_i^{pf}$. We conclude that the rating function can estimate the expected distribution which bounds the location estimate of particle filter, as intended. Some details of this experiment can be found in Table 3.3.

Location	**Σ′**	99% ellipse(m^2)	Particle cloud(m^2)
−53°8′17”, 47°23′27”	$\begin{bmatrix} 12.60 & -7.10 \\ -7.10 & 21.38 \end{bmatrix}$	428.1	419.3
−53°8′21”, 47°23′26”	$\begin{bmatrix} 3.19 & -4.01 \\ -4.01 & 23.60 \end{bmatrix}$	222.5	190.7
−53°8′20”, 47°23′29”	$\begin{bmatrix} 3.90 & -4.13 \\ -4.13 & 23.56 \end{bmatrix}$	250.2	228.9
−53°8′15”, 47°23′30”	$\begin{bmatrix} 25.57 & -0.55 \\ -0.55 & 23.99 \end{bmatrix}$	716.4	644.0
−53°8′18”, 47°23′28”	$\begin{bmatrix} 8.80 & -5.45 \\ -5.45 & 22.91 \end{bmatrix}$	379.3	365.5
−53°8′11”, 47°23′34”	$\begin{bmatrix} 24.84 & -1.62 \\ -1.62 & 23.82 \end{bmatrix}$	702.3	632.7

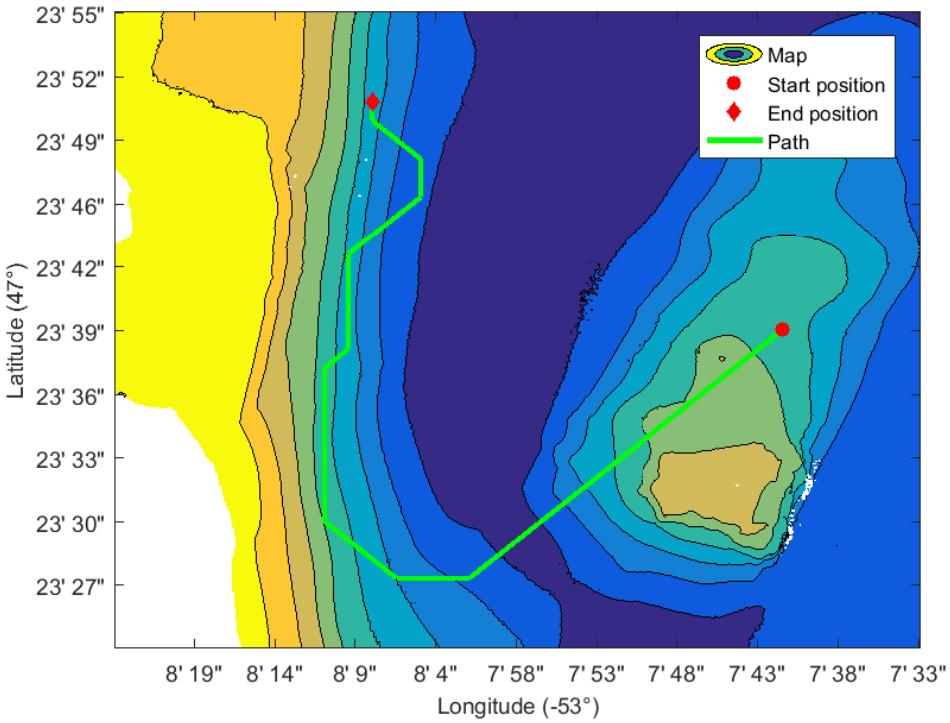
Table 3.3: A comparison between the expected distribution from the rating process and location estimation from particle filter simulation

3.4.2.2 Comparing rating with the effectiveness of glider TAN

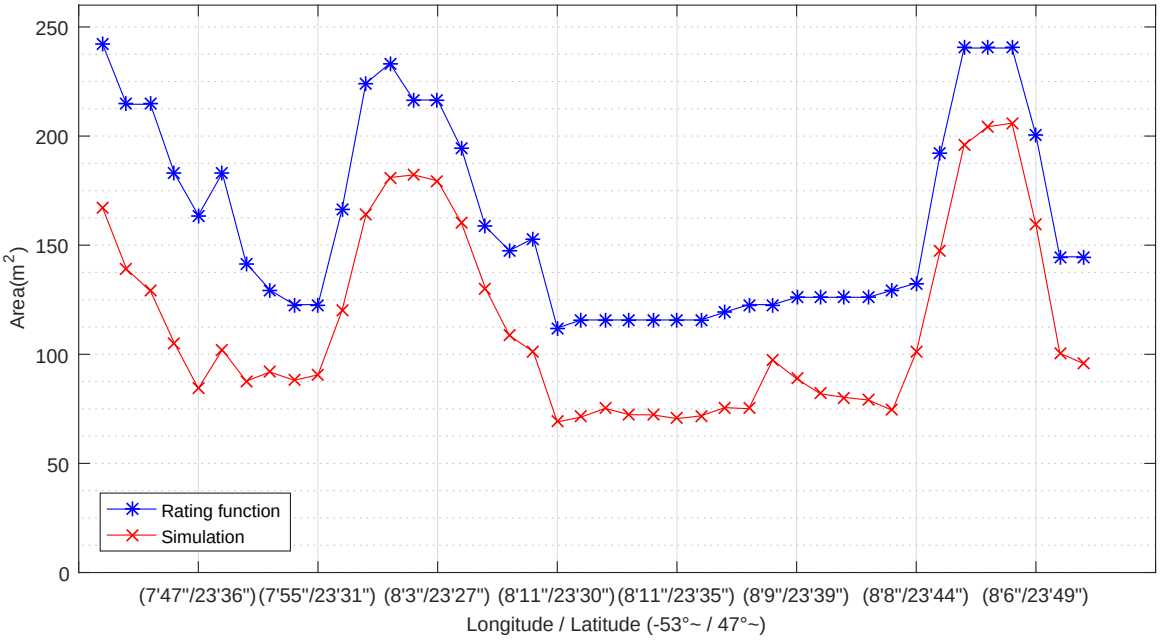
In our next experiment, we want to simulate a simplified version of the glider TAN algorithm and compare the estimation from the simulation with the rating estimation. We have simplified the depth calculation of glider TAN algorithm and extracted the depth value from map_{depth} with some noise. We also assumed that unlike the glider TAN algorithm, the depth measurement is not affected by the orientation of the glider.

With this setup, we selected two arbitrary locations in map_{depth} as the start and

goal location of the glider. Then we randomly selected some intermediate locations to create a path $P = \boldsymbol{x}_{start}, \boldsymbol{x}_1^p, \ldots \boldsymbol{x}_i^p \ldots, \boldsymbol{x}_{goal}$ that connects the start and goal locations. Figure 3.7a shows the path along with the elevation map. Next we ran a simulation of the simplified glider TAN algorithm on that path and recorded the size of particle clouds at each location where the glider took a depth measurement. Once the simulation completed, we took the path and computed the rating for those recorded locations. Figure 3.7b shows the result from both cases where red line represents the glider TAN simulation and blue line shows the associated rating estimation. The figure clearly shows that the rating estimation correctly follows the trend of particle cloud in glider TAN algorithm. One key observation regarding the result of this experiment is that the difference between the red and blue line has varied for different locations of the path. However, the red line is always smaller than the blue one; which is fine as we want the rating estimation to work as a bound for the particle cloud in glider TAN algorithm.



(a) Elevation map and a randomly selected path from start to goal locations



(b) A comparison between the particle cloud in glider TAN simulation and the estimation from the rating process

Figure 3.7: Comparing the rating estimation as a bound on the size of particle cloud in glider TAN algorithm

Chapter 4

Interesting Path Planning

In a typical mission, the glider visits a number of waypoints in which it can collect a wide range of data using the on-board sensors. A waypoint is a location of interest in the area of the ocean where the mission is executed. In most cases, these waypoints are set by the mission planner prior to the start of the mission. Usually two consecutive waypoints are connected by a straight path if there is no obstacles between them. During the mission, the glider tries to follow that straight path to reach the first waypoint. If a displacement is detected using the localization estimation, the glider can correct that by compensating the displacement from the path. The glider continues on that path until it reaches the first waypoint. Once the task at the first waypoint is accomplished, the glider heads towards the second waypoint. The process is repeated until all the waypoints are visited.

In the open ocean, the glider rarely faces a static obstacle like a land mass or an underwater mountain. This allows the glider to travel in a straight line between two consecutive waypoints which is also the shortest path between them. Although the ocean current and dynamic obstacles are addressed in some path planning algorithms for the glider, the likelihood of useful localization is not accounted in them. In our

path planning effort we are combining both aspects by computing the shortest path for a user defined constraint that limits the localization uncertainty in that path.

4.1 Shortest Path Problem

The shortest path problem is one of the classic problems in graph theory. A number of variations of this problem exist which can be classified using weighted, directed or undirected graphs. In its most common form, the single pair shortest path problem can be described as computing a path between two nodes in a graph in such a way that the sum of edge weights in that path is minimized. Edge weight can be the distance, travel time or the energy consumption during the travel between two nodes. Irrespective of what the weight represents, the solution to the shortest path problem aims to minimize that weight by producing an optimal path between the start node and goal node. Apart from the single pair shortest path problem, the other major variations include all pairs shortest path, single source shortest path and single goal shortest path. We are limiting our discussion to the single pair shortest path problem which is most relevant to our work.

4.1.1 Algorithm for shortest path problem

Dijkstra's algorithm [Dij59] is a well known method for solving the single pair shortest path problem. Although some variations of Dijkstra's algorithm can solve single source shortest path problem, the original Dijkstra's algorithm finds the shortest path between a source node and goal node in a directed weighted graph with no negative edge weight. Dijkstra's algorithm takes a greedy approach starting from the source node and does relaxation on every edge until an optimal path is computed to the goal node. Dijkstra's algorithm ensures the resulted path is the optimal with the

complexity of $\mathcal{O}(m \log n)$ on a graph with n vertices and m edges. Dijkstra's algorithm is an efficient algorithm, however many real world applications demand faster computation time, and hence heuristic-based algorithm like A* are more common in practice, especially when optimal paths are much shorter than the size of the graph.

4.1.1.1 A* Algorithm for path planning

The A* algorithm [HNR68] can be viewed as a Dijkstra's algorithm with an inclusion of a heuristic function. Whereas Dijkstra's algorithm greedily selects the best node based on the path cost, A* follows a similar approach with an addition of a heuristic value of the nodes, prioritising nodes with lesser combined path cost and heuristic. A common heuristic is an estimate of the path cost from any node to the goal node, allowing the algorithm to progress more quickly towards the goal.

To compute the desired path, A* begins by computing partial paths from the start node. Then, at each step, A* finds the best node to extend one of the previously computed partial paths. The step is repeated until one of the partial paths reaches the goal node, thus becoming a complete path from the start node to the goal node. To determine the best node, A* combines the current cost of a node with the heuristic cost of that node and selects the node with the minimum combined cost. Such selection criterion ensures that a node closer to the goal will have more priority than others with same partial path cost. This allows the algorithm to progress more quickly towards the goal, producing faster solution. Although heuristics can increase the efficiency of the algorithm in practice, poorly constructed heuristic function can affect the optimality of the solution.

4.1.1.2 Heuristic admissibility

A* can compute an optimal solution to the shortest path problem only with a admissible heuristic function. As we have mentioned before, a heuristic function estimates the true path cost from an arbitrary node to the goal node. And a heuristic function is considered admissible if and only if the estimation never exceeds the true path cost. It is necessary to note that heuristic can underestimate the true cost and that will not affect the optimality of the solution.

4.2 Interesting Path Problem

The problem of finding an interesting path is similar to the single pair shortest path problem with the exception of an uncertainty constraint. In the interesting path problem, we still want to minimize the path cost from the start location to the goal location but with an additional restriction that the maximum uncertainty in that path needs to be contained within a user defined constraint value. It is important to understand that the problem we are trying to solve here is not a multivariate optimization problem where two or more variables are needed to be optimized. On the contrary, an interesting path does not require optimization of uncertainty, rather we only consider paths which have uncertainty within a user defined threshold.

Given an elevation map of the ocean, an uncertainty constraint and a pair of start and goal locations, we want to compute the shortest path for the glider, with uncertainty bounded by a constraint. To define the problem, we construct a graph $G = (V, E)$, where V is the set of nodes representing locations (cells) and E is the set of directed weighted edges between those locations. There is an edge $(u, v) \in E$ if and only if the glider can directly travel from location u to location v and the weight of that edge is defined by the cost of that travel. To represent the uncertainty associated

with a node u, we use bivariate normal distribution $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{u}, \boldsymbol{\Sigma}_u)$. In our work, we considered an arbitrary $\boldsymbol{\Sigma}$ to be bounded by the given constraint if the major axis of the $\boldsymbol{\Sigma}$'s confidence ellipse is less than or equal to the constraint. We can formalize the interesting path problem as follows:

Given:

1. A graph $G = (V, E)$ representing the ocean area
2. An elevation map for the associated area of the ocean
3. A starting location $\boldsymbol{x}_{start}$ and an initial uncertainty $\boldsymbol{\Sigma}_{start}$
4. A goal location $\boldsymbol{x}_{goal}$
5. A user-defined uncertainty constraint t_{max}

Compute:

- The shortest path $P = \boldsymbol{x}_{start}, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \boldsymbol{x}_i, \ldots \boldsymbol{x}_{goal}$ such that every $\boldsymbol{\Sigma}_i$ in P is less than or equal to t_{max}.

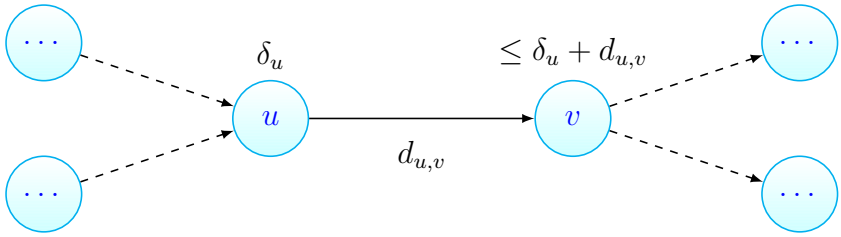
Naturally, the uncertainty associated with glider's location increases over time depending on the distance travelled and decreases after a measurement update. We are assuming the glider's on-board navigation system is based on Terrain-aided navigation and a depth measurement of the ocean is incorporated in the navigation. The growth of the location uncertainty may vary in different glider navigation systems, but uncertainty increases with the distance travelled by the glider between measurement updates. In our work, we are naming the increment of the uncertainty *Displacement Error*. We are assuming that a function *Displacement-Error()* computes the

displacement error of the glider for the associated navigation system. In general, the displacement error grows linearly with the distance in the direction of the ocean current [CB15]. Our work does not depend on the way the *Displacement-Error()* is defined and should work with any *Displacement-Error()* function that correctly reflects the growth of the uncertainty of the glider's location, such as dead reckoning.

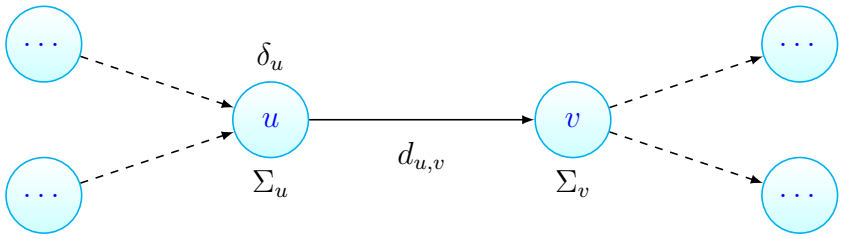
4.2.1 Algorithm for interesting path planning

Path planning algorithms like Dijkstra and A* are not suitable for solving the interesting path problem, although these algorithms can efficiently compute an optimal solution for the shortest path problem. In the shortest path problem, a traversal between two adjacent vertices depends only on the edge that connects those two vertices. In our case, a traversal in the interesting path problem also depends on the associated uncertainty distributions in both of those vertices. In addition to this, the uncertainty distribution can vary based on previous locations and distributions, which makes it very likely that different paths will reach the same vertex with different distribution and cost. The available path planning algorithms are not designed to address these challenges. Here, we are presenting some cases to elaborate them.

"Edge traversal in interesting path is not guaranteed for all edge $(u, v) \in E$*"*: Usually traversal on an edge in a weighted graph is predefined. If a path reaches an arbitrary vertex, that path can be extended to reach any of the adjacent vertices. Let us assume that a path reached a vertex u (Figure 4.1) with path cost δ_u and u has an adjacent vertex v connected by an edge (u, v) of weight $d_{u,v}$. In case of the shortest path problem (Figure 4.1a), it is certain that vertex v can be reached with a path cost of at most $\delta_u + d_{u,v}$. On the other hand, the same assertion can not be made for the interesting path problem. Figure 4.1b shows a similar scenario with an



(a) Edge traversal in shortest path problem



(b) Edge traversal in interesting path problem

Figure 4.1: The comparison of edge traversal on an arbitrary edge $(u, v) \in E$

inclusion of uncertainty $\mathbf{\Sigma}_u$ associated with the path ending at vertex u. Using the *Displacement-Error()* function, we can compute the uncertainty $\mathbf{\Sigma}_v$ at vertex v for the same path. Now, v can only be reached through that path if $\mathbf{\Sigma}_v \leq t_{max}$; thus making that edge traversal dependent on $\mathbf{\Sigma}_u$. It is necessary to understand that the uncertainty $\mathbf{\Sigma}_v$ depends on $\mathbf{\Sigma}_u$ as well as on every previous $\mathbf{\Sigma}$ in the current path. This may result in every path having different $\mathbf{\Sigma}$ values for the same vertex.

"Greedy selection can produce sub-optimal solutions for the interesting path problem": By design, both Dijkstra's algorithm and A* are greedy algorithms. A* uses a heuristic function to compute faster towards the goal, but underneath the algorithm follows a greedy approach. Greedy algorithms only keep track of the best path to reach a vertex, which works perfectly for the shortest path problem as edge traversal is guaranteed in such case, and we only need the best path to reach the goal. In the

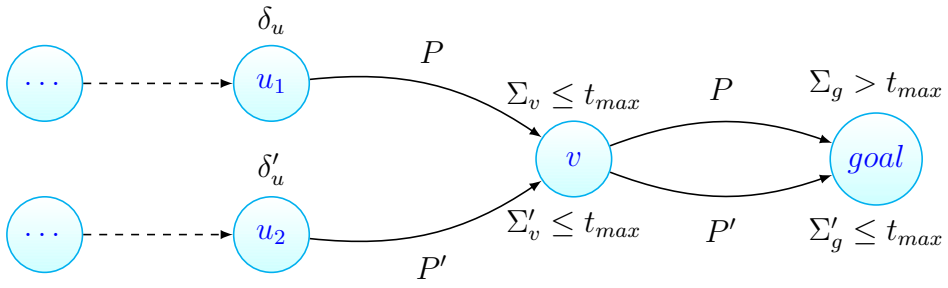


Figure 4.2: Greedy algorithms like Dijkstra's algorithm and A* can produce sub-optimal solution for interesting path problem

case of interesting path problem, an optimal path to an intermediate vertex may not reach the goal all the time, but a sub-optimal path to that intermediate vertex can lead to an optimal path to the goal. To illustrate the scenario, let us assume that two paths P and P' reached an intermediate vertex v through nodes u_1 and u_2 with uncertainties $\mathbf{\Sigma}_v$ and $\mathbf{\Sigma'}_v$, respectively (Figure 4.2). We also assume that $P < P'$ and $\Sigma'_v < \Sigma_v \leq t_{max}$. As both paths can reach the vertex v, naturally P is the optimal path for v and greedy approach will ignore P'. This should not pose any problem, unless the path P exceeds the uncertainty constraint and cannot reach the goal, but P' can. This can happen when P' reaches v with a longer path but with a lower uncertainty. Thus, P' can become a sub-optimal path for v, but an optimal path for the goal. This raises the question of keeping track of all previous sub-optimal paths which contradicts the concept of greedy approach.

"Traditional path planning algorithms rarely account for the actual travel path in real world": Path planning algorithms, specially the graph traversal ones can compute an optimal path and it is expected that following the optimal path should result in an optimal travel. In real world applications of path planning, such an expectation is hard to fulfill as environmental disturbances and faulty instruments can cause deviation from the optimal path. Thus, trying to follow an optimal path may result a sub-optimal travel. A similar challenge is present in the interesting path problem. For

instance, let us assume that we computed a path P for the glider and x is a location in P. We can also assume that x is very helpful for glider localization and the glider is expected to take a measurement at x. But, there is not enough guarantee that the glider will actually reach that location and the glider may end up at one of its neighbour x', which might not be as helpful for glider localization. Therefore, to compute a path with reduced uncertainty, we need to address the usefulness of a location as well as that of its neighbours.

4.3 Interesting path algorithm

We are proposing our *Interesting Path Algorithm* that can produce an optimal path while addressing the uncertainty involved in the interesting path problem. The algorithm builds upon A* algorithm and utilizes the rating technique we presented in Chapter 3. We combined the location of the glider and the associated uncertainty in the graph definition to allow the algorithm to compute the optimal shortest path within the user defined uncertainty constraint. As we have stated earlier, A* algorithm or its variants can not be applied directly to the given graph $G = (V, E)$. Instead, we define an intermediate graph $G' = (V', E')$ based on the given G in such a way that an optimal path in G' also serves as an optimal path in G.

4.3.1 Intermediate graph G'

To account for both the location in graph G and uncertainty of that location, we combined them in the intermediate graph $G' = (V', E')$, where V' is the set of vertices and E' is the set of edges between them. Each vertex $v \in V'$ consist of a location $\boldsymbol{x}_v \in V$ and a location uncertainty $\boldsymbol{\Sigma}_v$, thus the vertex v represents that the glider reached that location $\boldsymbol{x}_v$ with uncertainty $\boldsymbol{\Sigma}_v$. There is an edge $(u, v) \in E'$ whenever

it is possible to travel directly from vertex u to vertex v without a measurement, and with uncertainty not exceeding the constraint t_{max}. In other words, if the glider can start from location $\boldsymbol{x}_u$ with uncertainty $\boldsymbol{\Sigma}_u$ and reach location $\boldsymbol{x}_v$ with uncertainty $\boldsymbol{\Sigma}_v \leq t_{max}$, then there exists an edge between vertex u and v of cost $distance(\boldsymbol{x}_u, \boldsymbol{x}_v)$. We will only allow measurement update in the vertices and no depth measurement will take place in the edges. Such construction of G' ensures that if there is a path in G', then the glider can make that travel in G with the t_{max} bound.

For instance, let say we have a vertex $u = \left[\begin{smallmatrix} x_u \\ \Sigma_u \end{smallmatrix}\right] \in V'$ and we want to construct its neighbouring vertex v and the edge $(u, v) \in E'$ that connects them. First we compute the rating with distribution $\boldsymbol{X}_u \sim \mathcal{N}(\boldsymbol{x}_u, \boldsymbol{\Sigma}_u)$ to estimate the result of the measurement update at vertex u. Then we apply the *Displacement-Error()* function using the edge distance of $(\boldsymbol{x}_u, \boldsymbol{x}_v) \in E$ from the given graph G. By combining the rating estimation and the displacement error, we can compute $\boldsymbol{\Sigma}_v$ for any vertex v. If $\boldsymbol{\Sigma}_v \leq t_{max}$, then we can add edge (u, v) in E', otherwise there is no edge between these two vertices. The weight of an edge in G' will be the same as the corresponding edge in G, so that the path cost will remain same in both of the graphs.

Such construction of the intermediate graph G' implies that there will be multiple copies of each vertex of the given graph G along with a large number of edges between them. In theory, any combination of location and uncertainty can become a vertex in G' and therefore the number of vertices in G' can be infinite. Producing and storing the entire G' graph is not a feasible option. To address this issue, in our algorithm we are dynamically generating a finite subset of the vertices in G' which are only reachable from $\boldsymbol{x}_{start}$ with initial uncertainty $\boldsymbol{\Sigma}_{start}$ through some path bounded by t_{max}.

4.3.2 Computing path in G'

Once the intermediate graph G' is defined, we can apply our variant of A* algorithm to compute an optimal path. The algorithm is initialized with two empty sets of vertices Set_{open} and $Set_{visited}$. Then we create the first vertex $[\begin{smallmatrix} x_{start} \\ \Sigma_{start} \end{smallmatrix}]$ by combining the given starting location $\boldsymbol{x}_{start}$ and initial uncertainty $\boldsymbol{\Sigma}_{start}$. The distance of the vertex is set to 0 and the vertex is added to the Set_{open}. Next the algorithm enters its iterative phase. In each iteration, the best vertex is selected from the Set_{open} based on the current distance of the vertex combined with its heuristic value. It is worth mentioning that the heuristic function considers only the location part of the vertex and the associated uncertainty does not have any impact on the heuristic value. The selected vertex, or as we call it the current vertex, is removed the Set_{open} and added to the $Set_{visited}$.

Next, we apply the rating function to estimate the expected uncertainty, $\boldsymbol{\Sigma'}$ of the current vertex. Using $\boldsymbol{\Sigma'}$ we dynamically determine the neighbouring vertices of the current vertex. To be an eligible neighbour, a vertex needs to be reachable from the current vertex without exceeding the uncertainty constraint t_{max} and without any additional measurement update except for the one taken at current vertex. Once the neighbours are determined, we consider the following cases. First, the neighbours which exist in the $Set_{visited}$ are ignored. Second, we identify and ignore those neighbours for which at least one better vertex exists in the Set_{open}. A vertex is considered better if the vertex has the same location part as the neighbour, the distance of the vertex is smaller than that of the neighbour and the uncertainty of the vertex is smaller or equal to that of the neighbour[1]. Third, we find the vertices in the Set_{open} which are worse than any of the neighbours. A vertex is considered worse if the vertex

[1]We compare two uncertainties $\boldsymbol{\Sigma}_1$ and $\boldsymbol{\Sigma}_2$, by checking whether the confidence ellipse of one is entirely inside the confidence ellipse of other. If so, we consider $\boldsymbol{\Sigma}_1 \leq \boldsymbol{\Sigma}_2$. If not, we consider $\boldsymbol{\Sigma}_1$ and $\boldsymbol{\Sigma}_2$ to be incomparable

has the same location part as the neighbour, the distance of the vertex is equal or larger than that of the neighbour and the uncertainty of the vertex is larger than that of the neighbour. Finally, we add the remaining neighbours to the Set_{open}.

The process is repeated until a vertex with $\boldsymbol{x}_{goal}$ as a location is selected as the best vertex from the Set_{open} or no vertex is left in the Set_{open} to select from. If no vertex is left in the Set_{open}, then the algorithm terminates without a path; which indicates there is no path which is bounded by the user defined t_{max} value. Otherwise, the algorithm returns the path that reached the goal. A pseudo code of the interesting path algorithm is presented in Algorithm 1. Although the algorithm returns a path which is computed in G', it can be proven that the resulted path is also an optimal shortest path in G bounded by given t_{max}.

4.3.3 Proof of optimality

In the illustration of the algorithm in Section 4.3, we have stated that the interesting path algorithm utilizes an intermediate graph $G' = (V', E')$ from the given graph $G = (V, E)$, and subsequently applies a variant of A* algorithm on graph G' to compute an optimal solution for the given problem. In order to prove the optimality of our algorithm, first we need to establish that the optimality of A* holds for G'.

The intermediate graph G' can be considered as an ordinary graph where each vertex is a combination of a location and an uncertainty. We have defined the edges of G' using the uncertainties of the vertices and the user-defined constraint. Once the edges are defined, the uncertainty of a vertex does not contribute explicitly to the path cost and the edge weight as well as the heuristic function depend only on the location of the vertex. Although we are not producing all possible vertices and edges of graph G', all vertices and edges reachable from $\left[\begin{smallmatrix} x_{start} \\ \Sigma_{start} \end{smallmatrix}\right]$ can be generated when required. Therefore, if there exists an optimal path from location $\boldsymbol{x}_{start}$ to $\boldsymbol{x}_{goal}$ starting with

Algorithm 1 Interesting Path Algorithm

Require: $G = (V, E)$, x_{start}, Σ_{start}, x_{goal}, t_{Max}

Define the intermediate graph $G' = (V', E')$
$Set_{visited} \leftarrow \phi$
$Set_{open} \leftarrow \phi$
$v_{start} \leftarrow \left[\begin{smallmatrix} x_{start} \\ \Sigma_{start} \end{smallmatrix} \right]$
$dist[v_{start}] \leftarrow 0$
while $Set_{open} \neq \phi$ **do**
  find $u \in Set_{open}$ where $dist[u] + heuristic[u]$ is minimum
  $Set_{visited} \leftarrow Set_{visited} \cup \{u\}$
  if u is x_{goal} **then**
    return $RecoverPath()$
  end if
  $Set_{open} \leftarrow Set_{open} - \{u\}$
  Estimate $u.\Sigma' \leftarrow Rating(u.x, u.\Sigma)$
  $neighbours \leftarrow \{v : v \in V' \ \& \ (u, v) \in E'\}$
  for all $v \in neighbours$ **do**
    if $v \in Set_{visited}$ **then**
      ignore v and continue
    end if
    $dist[v] \leftarrow dist[u] + distance(u, v)$
    if $\exists v' \in Set_{open} : v'.x = v.x \ \& \ dist[v'] < dist[v] \ \& \ v'.\Sigma \leq v.\Sigma$ **then**
      ignore v and continue
    else if $\exists v' \in Set_{open} : v'.x = v.x \ \& \ dist[v'] \geq dist[v] \ \& \ v'.\Sigma > v.\Sigma$ **then**
      $Set_{open} \leftarrow (Set_{open} - \{v'\}) \cup \{v\}$
    else
      $Set_{open} \leftarrow Set_{open} \cup \{v\}$
    end if
  end for
end while

uncertainty $\mathbf{\Sigma}_{start}$ and bounded by t_{max}, that path will be included in the produced subset of the G'. Considering the above mentioned arguments, our variant of A* is similar to the original A* except for the fact that our variant ignores some vertices by comparing their uncertainties. We need to prove that no such vertex is a part of the optimal path and ignoring them will not cost the optimality of the algorithm.

Lemma 4.3.1. *For any two co-located vertices* $\left[\begin{smallmatrix} u \\ \Sigma \end{smallmatrix}\right], \left[\begin{smallmatrix} u \\ \Sigma' \end{smallmatrix}\right] \in V'$ *with shortest distance from start* $dist\left[\begin{smallmatrix} u \\ \Sigma \end{smallmatrix}\right]$ *and* $dist\left[\begin{smallmatrix} u \\ \Sigma' \end{smallmatrix}\right]$ *respectively, if* $\Sigma \leq \Sigma'$ *and* $dist\left[\begin{smallmatrix} u \\ \Sigma \end{smallmatrix}\right] < dist\left[\begin{smallmatrix} u \\ \Sigma' \end{smallmatrix}\right]$*, then* $\left[\begin{smallmatrix} u \\ \Sigma' \end{smallmatrix}\right]$ *cannot be a part of an optimal path.*

Proof. Let us assume that $\left[\begin{smallmatrix} u_0 \\ \Sigma_0 \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} u_0 \\ \Sigma'_0 \end{smallmatrix}\right]$ are two co-located vertices in intermediate graph G' where $\Sigma_0 \leq \Sigma'_0$. Both of the vertices are reachable from the starting location with distance $dist\left[\begin{smallmatrix} u_0 \\ \Sigma_0 \end{smallmatrix}\right]$ and $dist\left[\begin{smallmatrix} u_0 \\ \Sigma'_0 \end{smallmatrix}\right]$ respectively, where

$$dist\left[\begin{smallmatrix} u_0 \\ \Sigma_0 \end{smallmatrix}\right] < dist\left[\begin{smallmatrix} u_0 \\ \Sigma'_0 \end{smallmatrix}\right] \tag{4.1}$$

Let us also assume that vertex $\left[\begin{smallmatrix} u_0 \\ \Sigma'_0 \end{smallmatrix}\right]$ is a part of an optimal path P, where $P = \left[\begin{smallmatrix} x_{start} \\ \Sigma_{start} \end{smallmatrix}\right], \ldots, \left[\begin{smallmatrix} u_0 \\ \Sigma' \end{smallmatrix}\right], \left[\begin{smallmatrix} u_1 \\ \Sigma'_1 \end{smallmatrix}\right], \left[\begin{smallmatrix} u_2 \\ \Sigma'_2 \end{smallmatrix}\right], \left[\begin{smallmatrix} u_3 \\ \Sigma'_3 \end{smallmatrix}\right], \ldots, , \left[\begin{smallmatrix} x_{goal} \\ \Sigma'_{goal} \end{smallmatrix}\right]$. Now, the next vertex in the path is $\left[\begin{smallmatrix} u_1 \\ \Sigma'_1 \end{smallmatrix}\right]$ and there must exist a co-located vertex $\left[\begin{smallmatrix} u_1 \\ \Sigma_1 \end{smallmatrix}\right]$ which is adjacent to vertex $\left[\begin{smallmatrix} u_0 \\ \Sigma_0 \end{smallmatrix}\right]$. As we have mentioned earlier, the edge weight depends only on location; therefore, the distance between $\left[\begin{smallmatrix} u_0 \\ \Sigma_0 \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} u_1 \\ \Sigma_1 \end{smallmatrix}\right]$ will be the equal to the same between $\left[\begin{smallmatrix} u_0 \\ \Sigma'_0 \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} u_1 \\ \Sigma'_1 \end{smallmatrix}\right]$. Let that distance be d_{u_0,u_1} and using Equation 4.1 we can write

$$dist\left[\begin{smallmatrix} u_0 \\ \Sigma_0 \end{smallmatrix}\right] + d_{u_0,u_1} < dist\left[\begin{smallmatrix} u_0 \\ \Sigma'_0 \end{smallmatrix}\right] + d_{u_0,u_1}$$

$$dist\left[\begin{smallmatrix} u_1 \\ \Sigma_1 \end{smallmatrix}\right] < dist\left[\begin{smallmatrix} u_1 \\ \Sigma'_1 \end{smallmatrix}\right] \tag{4.2}$$

The uncertainty will also grow similarly with distance travelled in both cases and it is safe to say that $\Sigma_1 \leq \Sigma'_1 \leq t_{max}$. We can repeat the above step to show that

$$\begin{aligned} dist\begin{bmatrix} x_{goal} \\ \Sigma_{goal} \end{bmatrix} < dist\begin{bmatrix} x_{goal} \\ \Sigma'_{goal} \end{bmatrix} \\ \Sigma_{goal} \leq \Sigma'_{goal} \leq t_{max} \end{aligned} \tag{4.3}$$

Clearly, Equation 4.3 shows the existence of a shorter and valid path to location $\boldsymbol{x}_{goal}$ that does not go through vertex $\begin{bmatrix} u_0 \\ \Sigma'_0 \end{bmatrix}$, which contradicts the optimality of P. Therefore, such vertex $\begin{bmatrix} u_0 \\ \Sigma'_0 \end{bmatrix}$ cannot be a part of an optimal path. □

The above mentioned Lemma 4.3.1 shows that the exclusion of vertices we do in our algorithm does not affect the optimality and the optimality proof of A* [HNR68] still holds for computing a path in the intermediate graph G'. Using that proof of optimality, the following two corollaries can be proved for intermediate graph G'

Corollary 4.3.1. *At any time, if a vertex is selected from the Set_{open} as the minimum vertex with combined distance and heuristic value, then all the vertices in G' with smaller combined distance and heuristic values are already explored and added to the $Set_{visited}$.*

Corollary 4.3.2. *If a computed path P reaches a vertex $\begin{bmatrix} x_{goal} \\ \Sigma \end{bmatrix}$ in the intermediate graph G', then P will be the shortest path to reach $\boldsymbol{x}_{goal}$ with final uncertainty $\boldsymbol{\Sigma}$ in G'.*

We designed the Interesting Path Algorithm in such a way that when the goal is reachable, the algorithm terminates with the first path that reaches the goal location. The Corollary 4.3.2 tells us that the resulting path is optimal only for that particular $\boldsymbol{\Sigma}$ in graph G' and does not guarantee optimality in the given graph G as other shorter path ending with different $\boldsymbol{\Sigma}$ may exist. We need to prove that the first path that reaches the goal location in G', has equal or shorter distance compare to the other paths which reach the goal later.

Lemma 4.3.2. *The first path P reaching a vertex $\left[\begin{smallmatrix} x_{goal} \\ \Sigma \end{smallmatrix}\right]$ for an arbitrary $\boldsymbol{\Sigma}$, has less or equal path distance compared to that of any other path P' reaching vertex $\left[\begin{smallmatrix} x_{goal} \\ \Sigma' \end{smallmatrix}\right]$ for any $\boldsymbol{\Sigma'}$.*

Proof. Let P be the first path to reach vertex $\left[\begin{smallmatrix} x_{goal} \\ \Sigma \end{smallmatrix}\right]$ at time t in the intermediate graph G', where $\boldsymbol{x}_{goal}$ is the goal location. Let us assume that there exists a shorter path P' that will reach vertex $\left[\begin{smallmatrix} x_{goal} \\ \Sigma' \end{smallmatrix}\right]$ at a later time. As the heuristic function does not account for the uncertainty, the heuristic of both vertices $\left[\begin{smallmatrix} x_{goal} \\ \Sigma \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} x_{goal} \\ \Sigma' \end{smallmatrix}\right]$ should equal to 0 and we can represent the assumption as

$$dist\left[\begin{smallmatrix} x_{goal} \\ \Sigma' \end{smallmatrix}\right] < dist\left[\begin{smallmatrix} x_{goal} \\ \Sigma \end{smallmatrix}\right] \tag{4.4}$$

Now, consider the last node in P' before reaching the goal location is $\left[\begin{smallmatrix} v \\ \Sigma'' \end{smallmatrix}\right]$. At time t, vertex $\left[\begin{smallmatrix} v \\ \Sigma'' \end{smallmatrix}\right]$ can be present in $Set_{visited}$ or in Set_{open} or not being explored yet. For all of these cases, according to Corollary 4.3.1 the vertex $\left[\begin{smallmatrix} v \\ \Sigma'' \end{smallmatrix}\right]$ must have equal or larger combined distance and heuristic compare to that of vertex $\left[\begin{smallmatrix} x_{goal} \\ \Sigma \end{smallmatrix}\right]$. Therefore,

$$dist\left[\begin{smallmatrix} v \\ \Sigma'' \end{smallmatrix}\right] + heuristic\left[\begin{smallmatrix} v \\ \Sigma'' \end{smallmatrix}\right] \geq dist\left[\begin{smallmatrix} x_{goal} \\ \Sigma \end{smallmatrix}\right]$$

As heuristic must underestimate for optimality, $heuristic\left[\begin{smallmatrix} v \\ \Sigma'' \end{smallmatrix}\right]$ can be at most $d_{v,g}$, which is the distance between $\left[\begin{smallmatrix} v \\ \Sigma'' \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} x_{goal} \\ \Sigma' \end{smallmatrix}\right]$. By substituting this value in the above inequality, we can have the following.

$$\begin{aligned} dist\left[\begin{smallmatrix} v \\ \Sigma'' \end{smallmatrix}\right] + d_{v,g} &\geq dist\left[\begin{smallmatrix} x_{goal} \\ \Sigma \end{smallmatrix}\right] \\ dist\left[\begin{smallmatrix} x_{goal} \\ \Sigma' \end{smallmatrix}\right] &\geq dist\left[\begin{smallmatrix} x_{goal} \\ \Sigma \end{smallmatrix}\right] \end{aligned} \tag{4.5}$$

Here, Equation 4.5 is a direct contradiction of our assumption in Equation 4.4. Therefore, the first path in G' that reaches a vertex located at the goal location is the

shortest one among all the paths which can reach the goal location. □

By construction, the intermediate graph G' allows only those paths which have uncertainty within the user defined constraint t_{max}. And G' includes all the possible traversals in the given graph G which start at location $\boldsymbol{x}_{start}$ with initial uncertainty $\boldsymbol{\Sigma}_{start}$. Therefore, based on Lemma 4.3.2, we can say that the interesting path algorithm can compute the optimal shortest path from $\boldsymbol{x}_{start}$ to $\boldsymbol{x}_{goal}$ in given graph G within the uncertainty constraint t_{max}.

4.4 Implementation

We have implemented the interesting path algorithm using MATLAB. Ideally, priority queue should be used as the data structure for the algorithm, but in our experience we have found that a dynamic hybrid array structure works best in MATLAB environment. Some tweaking may be necessary for memory management in such data structure, but MATLAB can compute very efficiently in a vectorized array. Using the elevation data described in section 2.2.2.1, we ran the algorithm to compute interesting path between two given waypoints.

The data we received from the Centre for Applied Ocean Technology is gridded in longitude and latitude with 2 meters resolution in both direction. Based on the data, we created the input graph $G = (V, E)$ as a grid, but the algorithm should work with any regular directed weighted graph as well. The vertices in V are the locations in the ocean and the weight of an edge in E is defined by the distance between the adjacent vertices connected by that edge. Two meters distance between adjacent vertices can be too restrictive for glider maneuvering and the mission planner of the glider should set a suitable minimum distance between the vertices. In our experiment, we decreased the resolution of the given map and the vertices in V became at least 10 meters apart

from each other.

4.4.1 Interesting path between a pair of waypoints

In the first experiment, we took the graph G and marked two vertices as waypoints, where the first vertex is the starting location and the second one is the goal location. Then we ran the interesting path algorithm to compute the shortest path with $t_{max} = 38\ meters$. The resulted path from the algorithm is plotted in figure 4.3 along with the confidence ellipses in that path.

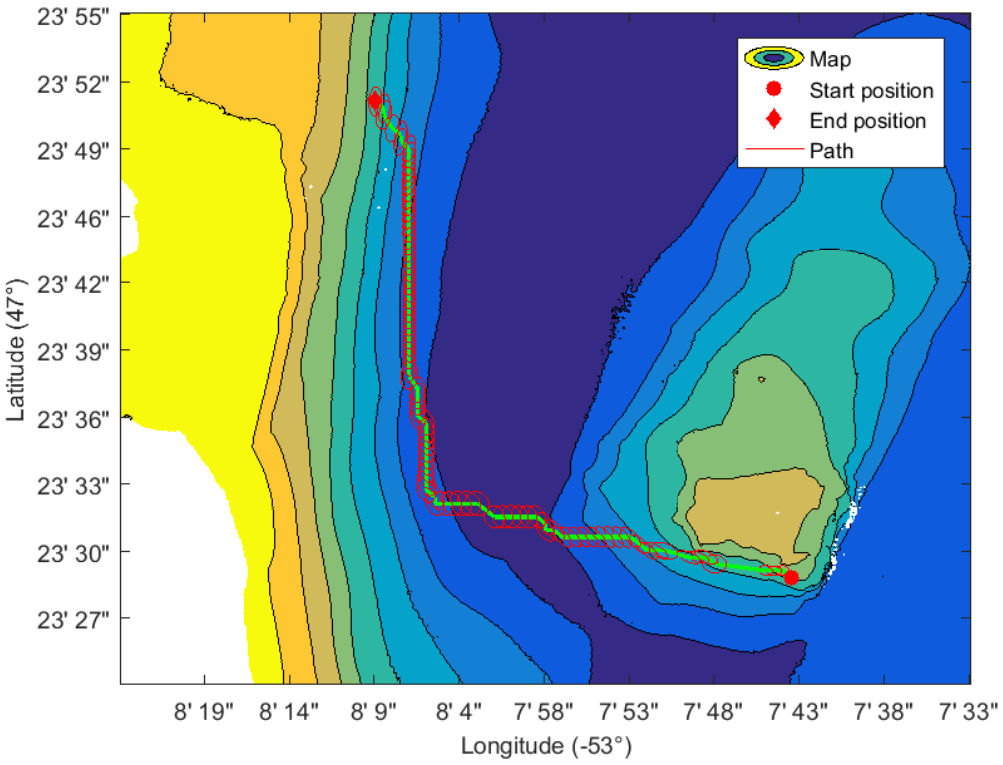


Figure 4.3: The shortest path between the start and the goal locations bounded by the uncertainty constraint $t_{max} = 38\ meters$

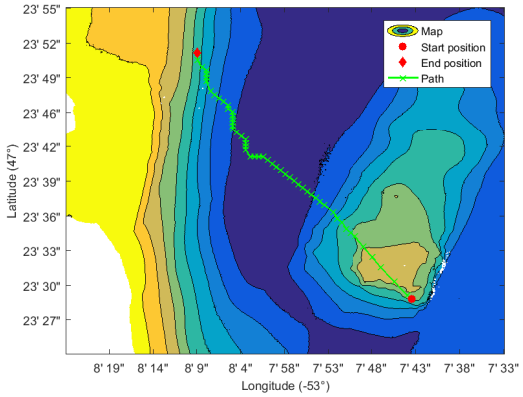
We chose the starting and goal locations in this experiment such a way that the usual shortest path (without any constraint) goes through a large flat area. The

yellow colored area in Figure 4.3 represents that flat area. We have already shown in chapter 3, that such flat areas are not good for reducing uncertainty and should be avoided where necessary. In Figure 4.3, we can see that the resulted path took a slight detour and crossed the flat surface through a narrow area compare to that of the usual shortest path. This detour helped the resulting path to keep the uncertainty below the given constraint. Another observation of result is that there exists a narrower crossing in the map with lesser flat area but larger path distance. The resulted path did not go through that area which indicates the algorithm only optimizes the distance within the given threshold and does not sacrifice distance for unnecessary reduction of uncertainty.

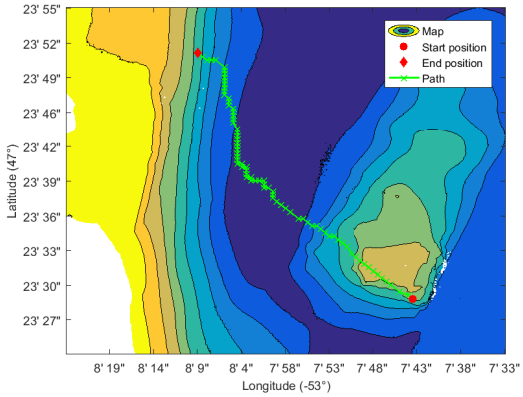
4.4.2 The impact of uncertainty constraint over interesting path

The interesting path algorithm is designed to facilitate shortest path within the given uncertainty constraint, thus allowing a balance between the safety and the travel cost. A flexible constraint value will produce a path similar to the shortest path of a traditional path planning problem, where as a restrictive constraint value can result in a longer path. In our next experiment, we want to demonstrate the impact of the constraint to the generated path from the interesting path algorithm. We took a pair of waypoints and ran the algorithm with different values of the t_{max}, starting with a flexible t_{max} and making it more restrictive in each run. Figure 4.4 shows the results of the experiment.

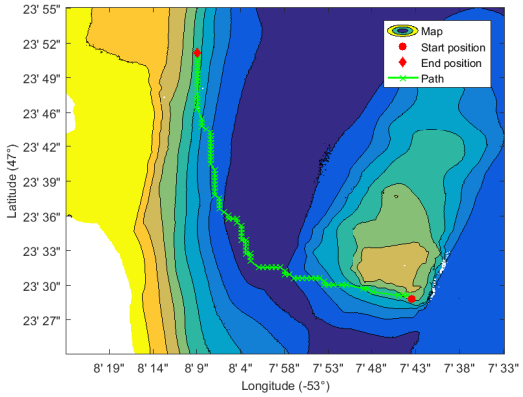
In the first run, we set $t_{max} = 48\ meters$, which is a very flexible constraint, and the resulted path is almost a straight line from the start to goal location (Figure 4.4a). A large portion of the path went through the flat surface where the uncertainty became large, but that is acceptable as the constraint was large as well. In the



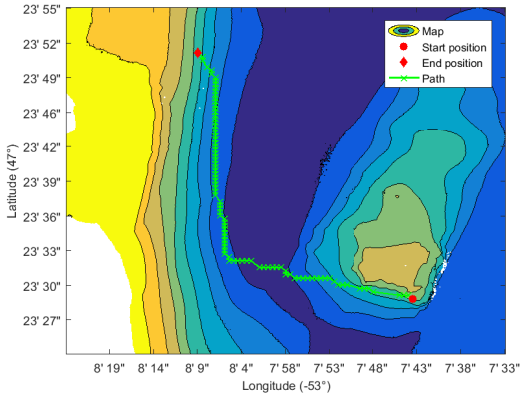
(a) $t_{max} = 48\ meters$



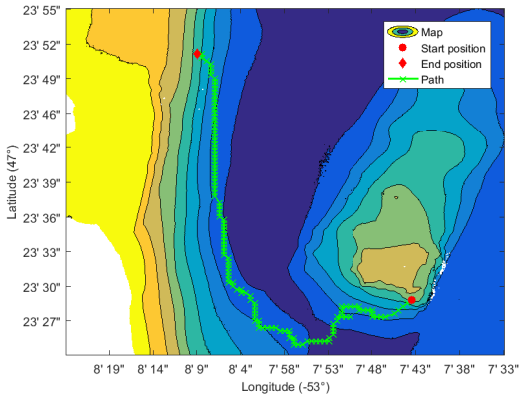
(b) $t_{max} = 44\ meters$



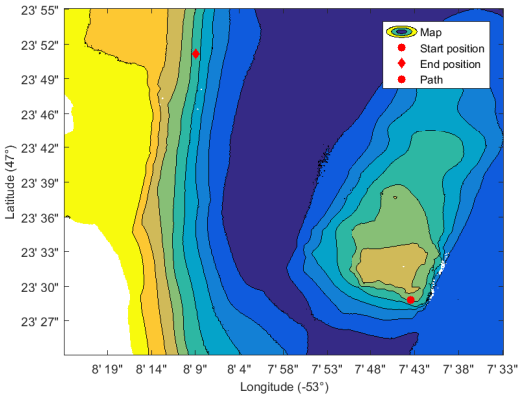
(c) $t_{max} = 40\ meters$



(d) $t_{max} = 38\ meters$



(e) $t_{max} = 36\ meters$



(f) $t_{max} = 34\ meters$

Figure 4.4: Impact of different uncertainty constraints on the resulting paths from the interesting path algorithm

next two run, we restrict the constraint with $t_{max} = 44\ meters$ (figure 4.4b) and $t_{max} = 40\ meters$ (Figure 4.4c), the resulted paths started to bend away from the flat area. With $t_{max} = 38\ meters$ in Figure 4.4d, the constraint is restrictive enough to generate a path that has only a small portion in the flat area. We push the constraint further in figure 4.4e with $t_{max} = 36\ meters$ and the resulted path become even longer so that it can avoid the flat surface with many detours. In the last run, we set $t_{max} = 34\ meters$ and the algorithm couldn't find a path indicating no travel is guaranteed without exceeding the given t_{max} value. Some details about the resulted paths from this experiment can be found in Table 4.1.

Path	$\boldsymbol{t_{max}}$ *in meters*	**# of intermediate vertices**	**Length** (meters)
4.4a	48	55	909.35
4.4b	44	79	983.77
4.4c	40	90	1089.89
4.4d	38	94	1091.02
4.4e	36	137	1459.28
4.4f	34	–	∞

Table 4.1: The impact of different uncertainty constraint in the resulted paths from the interesting path algorithm

Chapter 5

Conclusion

5.1 Summary

We have addressed a specific problem in glider path planning which affects the safety of the gliders during their missions. We have approached the problem from a new direction and our experiments show satisfactory results. We intend to use the paths generated by our algorithm in field trials and we are hopeful that this will help the glider's on-board navigation system to localize better.

The proposed rating function and our path planning algorithm are primarily designed for the gliders. However, our work is applicable to any autonomous vehicle that utilizes depth measurement for localization. We believe that the technique we introduced in this work can be adapted to many scenarios and there is scope for further improvements.

5.2 Future work

The technique we presented here can be a stepping stone for future research. Several directions of research can be possible which will benefit the underwater navigation. We

are planning to extend this work by incorporating the ocean currents while computing the interesting path. Including ocean currents in path planning is challenging in the presence of location uncertainty, but addressing the ocean current will improve the robustness of the interesting path algorithm.

We also considered planning an interesting path that visits a sequence of waypoints. In this variant of interesting path problem, we want to find the shortest path that goes through a given sequence of waypoints while maintaining the uncertainty below the user defined constraint. Computing interesting path for a sequence of waypoints can be advantageous for glider missions, specially for the ones which take place under the arctic ice. We believe that this can be addressed by running a dynamic programming algorithm as an outer loop, which would select a sequence of points for which our algorithm produces the best total path; implementing this is work in progress. Note that this problem should not be confused with *Traveling Salesman Problem*, where the sequence of waypoints is unknown and needs to be computed.

Another possible direction is to produce a smoother interesting path that accounts for the dynamics of the AUV. For instance, a glider has a limited maneuverability and can not make a sharp turn in the ocean. The interesting path algorithm can be modified, so that the resulted path does not include such sharp turns and the glider can easily follow that path.

Bibliography

[ACO04] Alberto Alvarez, Andrea Caiti, and Reiner Onken. Evolutionary path planning for autonomous underwater vehicles in a variable ocean. *IEEE Journal of Oceanic Engineering*, 29(2):418–429, 2004.

[Agh12] Mohammad Pourmahmood Aghababa. 3D path planning for underwater vehicles using five evolutionary optimization algorithms avoiding static and energetic obstacles. *Applied Ocean Research*, 38:48–62, 2012.

[AMGC02] M Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, Feb 2002.

[AYK15] Mansour Ataei and Aghil Yousefi-Koma. Three-dimensional optimal path planning for waypoint guidance of an autonomous underwater vehicle. *Robotics and Autonomous Systems*, 67:23–32, 2015.

[Ber93] Oddbjørn Bergem. *Bathymetric navigation of autonomous underwater vehicle using a multibeam sonar and a kalman filter with relative measurement covariance matrices*. PhD thesis, University of Trondheim, Trondheim, Norway, 1993.

[BLL92] Jerome Barraquand, Bruno Langlois, and J-C Latombe. Numerical potential field techniques for robot path planning. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(2):224–241, 1992.

[BTAH02] John S Bellingham, Michael Tillerson, Mehdi Alighanbari, and Jonathan P How. Cooperative path planning for multiple UAVs in dynamic and uncertain environments. In *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, volume 3, pages 2816–2822. IEEE, 2002.

[CB15] Brian Claus and Ralf Bachmayer. Terrain-aided Navigation for an Underwater Glider. *Journal of Field Robotics*, 32(7):935–951, oct 2015.

[cF05] ©Freezingmariner. (https://commons.wikimedia.org/wiki/file:Ru02_flying_in_sargasso_sea.jpg) CC BY-SA 3.0, 2005.

[CFLC10] Chi-Tsun Cheng, Kia Fallahi, Henry Leung, and K Tse Chi. An AUVs path planner using genetic algorithms with a deterministic crossover operator. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2995–3000. IEEE, 2010.

[CL06] Xin Chen and Yangmin Li. Smooth path planning of a mobile robot using stochastic particle swarm optimization. In *2006 International Conference on Mechatronics and Automation*, pages 1722–1727. IEEE, 2006.

[CMN+92] Kevin P Carroll, Stephen R McClaran, Eric L Nelson, David M Barnett, Donald K Friesen, and GN William. AUV path planning: an A* approach to path planning with consideration of variable vehicle

speeds and multiple, overlapping, time-dependent exclusion zones. In *Autonomous Underwater Vehicle Technology, 1992. AUV'92., Proceedings of the 1992 Symposium on*, pages 79–84. IEEE, 1992.

[DCZM12] Yong Deng, Yuxin Chen, Yajuan Zhang, and Sankaran Mahadevan. Fuzzy dijkstra algorithm for shortest path problem under uncertain environment. *Applied Soft Computing*, 12(3):1231–1237, 2012.

[Dek05] Frederik Michel Dekking. *A Modern Introduction to Probability and Statistics: Understanding why and how*. Springer Science & Business Media, 2005.

[DGA00] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10(3):197–208, 2000.

[Dij59] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[DNKF10] Kenny Daniel, Alex Nash, Sven Koenig, and Ariel Felner. Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research*, 39:533–579, 2010.

[DR12] Shandor Dektor and Stephen Rock. Improving robustness of terrain-relative navigation for AUVs in regions with flat terrain. In *2012 IEEE/OES Autonomous Underwater Vehicles (AUV)*, pages 1–7. IEEE, sep 2012.

[FHL+03] V. Fox, J. Hightower, Lin Liao, D. Schulz, and G. Borriello. Bayesian filtering for location estimation. *IEEE Pervasive Computing*, 2(3):24–33, July 2003.

[FPCGHS+10] Enrique Fernández-Perdomo, Jorge Cabrera-Gámez, Daniel Hernández-Sosa, Josep Isern-González, Antonio C Domínguez-Brito, Alex Redondo, Josep Coca, Antonio G Ramos, Enrique Álvarez Fanjul, and Marcos García. Path planning for gliders using Regional Ocean Models: Application of pinzón path planner with the ESEOAT model and the RU27 trans-Atlantic flight data. In *OCEANS 2010 IEEE-Sydney*, pages 1–10. IEEE, 2010.

[FPHSIG+11] Enrique Fernández-Perdomo, Daniel Hernández-Sosa, Josep Isern-González, Jorge Cabrera-Gámez, Antonio C Dominguez-Brito, and Víctor Prieto-Marañón. Single and multiple glider path planning using an optimization-based approach. In *OCEANS 2011 IEEE-Spain*, pages 1–10. IEEE, 2011.

[FS06a] Dave Ferguson and Anthony Stentz. Anytime RRTs. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5369–5375. IEEE, 2006.

[FS06b] Dave Ferguson and Anthony Stentz. Using interpolation to improve path planning: The Field D* algorithm. *Journal of Field Robotics*, 23(2):79–101, 2006.

[GSS93] N.J. Gordon, D.J. Salmond, and A.F.M. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. *IEE Proceedings F (Radar and Signal Processing)*, 140:107–113(6), April 1993.

[HDS15] Van T Huynh, Matthew Dunbabin, and Ryan N Smith. Predictive motion planning for AUVs subject to strong time-varying currents and forecasting uncertainties. In *2015 IEEE International Confer-*

ence on Robotics and Automation (ICRA), pages 1144–1151. IEEE, 2015.

[HNR68] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[HR84] Wayne E Hoover and MD Rockville. Algorithms for confidence circles and ellipses. Technical report, NOAA, 1984.

[IGHSFP+11] Josep Isern-González, Daniel Hernández-Sosa, Enrique Fernández-Perdomo, Jorge Cabrera-Gámez, Antonio C Domínguez-Brito, and Víctor Prieto-Marañón. Path planning for underwater gliders using iterative optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1538–1543. IEEE, 2011.

[JSCP15] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International journal of robotics research*, page 0278364915577958, 2015.

[KF11] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[KL00] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

[KSBB07] Dov Kruger, Rustam Stolkin, Aaron Blum, and Joseph Briganti. Optimal AUV path planning for extended missions in complex, fast-flowing estuarine environments. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 4265–4270. IEEE, 2007.

[KSLO96] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[LaV98] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[LD09] Liqiang Liu and Yuntao Dai. 3D space path planning of complex environmental underwater vehicle. In *Computational Sciences and Optimization, 2009. CSO 2009. International Joint Conference on*, volume 2, pages 204–209. IEEE, 2009.

[LG12] Shuai Li and Yi Guo. Neural-network based AUV path planning in estuary environments. In *Intelligent Control and Automation (WCICA), 2012 10th World Congress on*, pages 3724–3730. IEEE, 2012.

[LK01] Steven M LaValle and James J Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[MB12] James D Marble and Kostas E Bekris. Towards small asymptotically near-optimal roadmaps. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2557–2562. IEEE, 2012.

[NBK06] Evdokia Nikolova, Matthew Brand, and David R Karger. Optimal route planning under uncertainty. In *ICAPS*, volume 6, pages 131–141, 2006.

[NDKF07] Alex Nash, Kenny Daniel, Sven Koenig, and Ariel Felner. Theta*: Any-Angle Path Planning on Grids. In *Proceedings of the national conference on artificial intelligence*, volume 22, page 1177. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[New13] CBC News. www.cbc.ca/news/canada/newfoundland-labrador/yellow-robot-lost-off-coast-of-newfoundland-1.1381444, 2013.

[OS88] Stanley Osher and James A Sethian. Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulations. *Journal of computational physics*, 79(1):12–49, 1988.

[PBHS13] Arvind A Pereira, Jonathan Binney, Geoffrey A Hollinger, and Gaurav S Sukhatme. Risk-aware path planning for autonomous underwater vehicles using predictive ocean models. *Journal of Field Robotics*, 30(5):741–762, 2013.

[PBJ+11] Arvind A Pereira, Jonathan Binney, Burton H Jones, Matthew Ragan, and Gaurav S Sukhatme. Toward risk aware mission planning

for autonomous underwater vehicles. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3147–3153. IEEE, 2011.

[PPP+07] Clment Petres, Yan Pailhas, Pedro Patron, Yvan Petillot, Jonathan Evans, and David Lane. Path planning for autonomous underwater vehicles. *IEEE Transactions on Robotics*, 23(2):331–341, 2007.

[PPPL05] Clément Pêtrès, Yan Pailhas, Yvan Petillot, and Dave Lane. Underwater path planing using fast marching algorithms. In *Europe Oceans 2005*, volume 2, pages 814–819. IEEE, 2005.

[RW09] Dushyant Rao and Stefan B Williams. Large-scale path planning for underwater gliders in ocean currents. In *Australasian Conference on Robotics and Automation (ACRA), Sydney*, 2009.

[SBL08] Chunxue Shi, Yingyong Bu, and Jianghui Liu. Mobile robot path planning in three-dimensional environment based on ACO-PSO hybrid algorithm. In *2008 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 252–256. IEEE, 2008.

[Sci17] Eberly College Of Science. https://onlinecourses.science.psu.edu/stat414/node/147, 2017.

[Sim06] Dan Simon. *The particle filter*, pages 461–483. John Wiley & Sons, Inc., 2006.

[Sou11] Michael Soulignac. Feasible and optimal path planning in strong current fields. *IEEE Transactions on Robotics*, 27(1):89–98, 2011.

[SR94] Joao Sequeira and Maria Isabel Ribeiro. A two level approach for underwater path planning. In *OCEANS'94.'Oceans Engineering for Today's Technology and Tomorrow's Preservation.'Proceedings*, volume 2, pages II–87. IEEE, 1994.

[SS09] Gabor Szucs and Gyula Sallai. Route planning with uncertain information using dempster-shafer theory. In *Management and Service Science, 2009. MASS'09. International Conference on*, pages 1–4. IEEE, 2009.

[SY97] Kazuo Sugihara and Junku Yuh. GA-based motion planning for underwater robotic vehicles. In *International Symposium on Unmanned Untethered Submersible Technology*, pages 406–415. Citeseer, 1997.

[TSC05] Chiew Seon Tan, Robert Sutton, and John Chudley. Quasi-random, manoeuvre-based motion planning algorithm for autonomous underwater vehicles. *IFAC Proceedings Volumes*, 38(1):103–108, 2005.

[VGGGGBM13] Alberto Valero-Gomez, Javier Victorio Gómez González, Luis Santiago Garrido Bullón, and Luis Moreno. The path to efficiency: fast marching method for safer, more efficient mobile robot trajectories. 2013.

[War90] Charles W Warren. A technique for autonomous underwater vehicle route planning. *IEEE Journal of Oceanic Engineering*, 15(3):199–204, 1990.

[WLMHK16] Tong Wang, Olivier P Le Maître, Ibrahim Hoteit, and Omar M Knio. Path planning in uncertain flow fields using ensemble method. *Ocean Dynamics*, 66(10):1231–1251, 2016.

[YK15] Byunghyun Yoo and Jinwhan Kim. Path optimization for marine vehicles in ocean currents using reinforcement learning. *Journal of Marine Science and Technology*, pages 1–10, 2015.

[YZF+13] Gao Yun, Wei Zhiqiang, Gong Feixiang, Yin Bo, and Ji Xiaopeng. Dynamic path planning for underwater vehicles based on modified artificial potential field method. In *Digital Manufacturing and Automation (ICDMA), 2013 Fourth International Conference on*, pages 518–521. IEEE, 2013.

[ZKB07] Matt Zucker, James Kuffner, and Michael Branicky. Multipartite RRTs for rapid replanning in dynamic environments. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1603–1609. IEEE, 2007.

[ZSL+15] Zheng Zeng, Karl Sammut, Andrew Lammas, Fangpo He, and Youhong Tang. Imperialist Competitive Algorithm for AUV Path Planning in a Variable Ocean. *Applied Artificial Intelligence*, 29(4):402–420, 2015.

[ZSL+16] Zheng Zeng, Karl Sammut, Lian Lian, Fangpo He, Andrew Lammas, and Youhong Tang. A comparison of optimization techniques for AUV path planning in environments with ocean currents. *Robotics and Autonomous Systems*, 82:61–72, 2016.