Mapping of Underwater Environments with an ROV using a robotic Smart Tether in conjunction with Simultaneous Localization and Mapping (SLAM) Algorithms

Andrew Boik

Class of 2013

Advisor: Professor Christopher Clark

Submitted to the Department of Computer Science
Princeton University
Princeton, NJ

April 30, 2012

This thesis represents my own work in accordance with University regulations.

Andrew Boik

Abstract

This paper describes the problem of mapping with robots, especially underwater robots, and presents a system to solve that problem. It begins with a mathematical definition of the problem of Simultaneous Localization and Mapping, and details several popular approaches used to solve the problem including Kalman Filters and particle filters. Furthermore, these approaches are applied to the domain of underwater robotics, and existing work in two-dimensional underwater robotics is extended to the third dimension through the use of different models and equipment. The algorithms developed are then applied to an actual three-dimensional underwater environment, and the results are shown.

Contents

1 Introduction

In this paper, we develop a system that integrates robot sensor measurements from an ROV (remotely operated underwater vehicle) as well as measurements from a KCF Smart Tether, a tether with sensor nodes for accurately determining the location of the ROV to create accurate maps of underwater environments. The problem of accurately mapping an unknown environment using a robot is a difficult one. Through the use of sophisticated algorithms and sensors, however, we are able to create accurate maps of underwater environments that are potentially useful for fields outside of robotics.

1.1 ICEX 2012: The Maltese Mapping Project

The International Computer Engineering Experience Program, or ICEX, was started at California Polytechnic State University to allow students of computer science, computer engineering, and other related fields to apply their technical knowledge to real-world problems in an international context, as well as increase their cultural understanding. This year ICEX consisted of four students from Princeton, including myself, seven students from Cal Poly, and Profs. Jane Lehr and Zoe Wood of Cal Poly, and Prof. Chris Clark of Princeton University and Cal Poly. In March of 2012 ICEX participants traveled to the island nation of Malta and worked in collaboration with archaeologists from the Aurora Special Purpose Trust and the University of Malta to develop a system capable of creating maps of underwater cisterns through the use of a small ROV, sonar, and probabilistic SLAM algorithms.

Underwater cisterns, tunnel-like systems that are usually found in fortresses, churches, and even some private homes in Malta, historically provided a relatively clean and efficient water capture and storage system in a dry country with limited seasonal rainfall. Several of the cisterns that were visited by ICEX date back to 300 B.C., and the survey of such systems provides archaeologists with key insight into the origins of such tunnels and wells as well as information as to how they were integrated with modern water management systems. Previous attempts at human exploration of these cisterns have proven too

expensive and difficult, and the small size constraints as well as the possibility of damaging the cisterns makes this exploration method undesirable. However, through the use of small underwater robots, we are able to efficiently create maps of these environments in a non-damaging manner [4].

Previous trips to Malta have focused solely on creating better maps of new as well as previously explored cisterns. The most recent trip however, has extended the techniques used in cistern mapping to creating maps of other underwater environments, such as marine caves and shipwrecks. This application has special implications for the fields of marine biology and maritime archaeology. In this way, the field of robotics can greatly simplify the data collection and analysis process for seemingly unrelated fields of research.

1.2 Motivation

The application of previously developed mapping techniques to new underwater environments has implications for the fields aforementioned, and it also provides new technical challenges to solve. The process of mapping these new underwater environments is in many ways much more complicated than mapping underwater cisterns. Previous software for mapping cistern environments operated in two dimensions, as the third dimension was easily extrapolated by "growing" walls upwards from the base of the cistern. The cisterns also provided a static environment for exploration, with no current that would increase variation in localization measurements. Marine caves, however, possess irregular geometry as well as dynamic currents that add noise to our measurements.

1.3 Contributions

We were able to modify the previous cistern mapping software so that is creates three-dimensional maps. For this purpose, the ROV used was deployed in two configurations: one with a sonar mounted on top of the ROV scanning horizontally, and one with the sonar mounted on the front scanning vertically. These horizontal and vertical scans were then used to create a 3D map of the environment using a highly accurate KCF Smart Tether to provide location data and a particle filter to refine the observation estimates.

2 Background and Related Work

2.1 History and Overview of SLAM

Creating a map of an unknown environment requires the robot to move throughout the environment while simultaneously integrating its sensor measurements to update its hypothesis of what the map looks like. This process requires the robot to keep track of its current location within the area at all stages of the process in order to build a consistent map. Known as the SLAM problem, or Simultaneous Localization and Mapping, it essentially requires the robot to answer two questions without any a priori knowledge: "Where am I?", and "What does the world look like?" The solution to SLAM incorporates the processes used to answer these questions into a feedback loop, where the answer to one question is used to more accurately answer the other question. The combination of accumulative errors in localization as well as noisy environments make this problem especially difficult.

The Simultaneous Localization and Mapping problem has, in fact, only recently been solved within the past two decades. It originated in 1986 at the IEEE Robotics and Automation Conference with work by Randall C. Smith and Peter Cheeseman on the estimation of spatial uncertainty [2] and later with work by Hugh Durrant-Whyte [3] on describing relationships between objects in a map and manipulating geometric uncertainty. The key contributions of these works were that they showed there must be a high degree of correlation between location estimates of different objects in a map and they proved that these correlations would increase with the number of successive observations[1].

Today, the SLAM problem is formulated probabilistically, where an observation model characterizes the probability of making a specific observation when the robot's location as well as the location of landmarks in the map are known, and where a motion model describes the probability of a state transition to another location given some control inputs. The SLAM algorithm uses sequential prediction steps, which offer a hypothesis of the location given the motion model, and correction steps, which refine the hypothesis

by determining how well this location estimation fits the observation model. These two steps are bound in a loop, and the algorithm is usually implemented using Kalman filters or sequential Monte Carlo methods known as particle filters.

2.2 Probabilistic SLAM

SLAM is intended to be used with a mobile robot that moves through an environment while making observations about the surrounding area using sensors (whether they be sonar, laser, visual etc.). The high amount of uncertainty and sensor noise associated with robot mapping necessitates the use of probabilistic algorithms for mapping. Probabilistic algorithms model explicitly different source of sensor noise and their effects on measurements and their uncertainty [6]. Therefore, we introduce the following probabilistic formulation of SLAM as described in Durrant-Whyte & Bailey (2006). At time t, we define the following:

- $\mathbf{x}_t$: The state vector of the robot which describes its position and orientation.
- $\mathbf{u}_t$: The control input vector consisting of the robot control inputs at time $t-1$ used to drive the robot to state $\mathbf{x}_t$ at time t.
- $\mathbf{m}_k$: The vector of describing the true location of landmark k, which can be thought of as a point in the plane for our purposes.
- $\mathbf{z}_{tk}$: An observation taken by the robot at time t of the location of landmark k. This is abbreviated as $\mathbf{z}_t$ when referring to a set of multiple landmark observations.
- $\mathbf{X}_{\mathbf{0}:t}$: The history of robot locations.
- $\mathbf{U}_{\mathbf{0}:t}$: The history of control inputs.
- $\mathbf{M} = \{\mathbf{m_1}, \mathbf{m_2}, \cdots, \mathbf{m_K}\}$: The set of all landmark location vectors, i.e. the map.
- $\mathbf{Z}_{\mathbf{0}:t}$: The set of all landmark observations.

A solution to the SLAM problem involves, for all times t, computing the joint posterior density of the set of landmark locations and the robot state at time t conditioned on initial robot state, the history of observations up to and including time k, and the history of control inputs up to and including time t:

$$Pr(\mathbf{x}_t, \mathbf{M} \mid \mathbf{Z}_{\mathbf{0}:t}, \mathbf{U}_{\mathbf{0}:t}, \mathbf{x_0}) \tag{1}$$

If we start with an estimate of the joint posterior at time $t-1$, we can calculate the joint posterior at time t recursively using Bayes Theorem. First, we describe an observation model, the probability of observing $\mathbf{z}_t$ when the robot location and landmark locations are known:

$$Pr(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{M}) \tag{2}$$

We can assume that the observations are conditionally independent of each other given a robot location and map. Next we describe a motion model, a probability distribution of state transitions which are assumed to be Markov processes in which the next state only depends on its previous state and control input, and which are independent of observations and the map:

$$Pr(\mathbf{x}_t \mid \mathbf{x}_{t-\mathbf{1}}, \mathbf{u}_t) \tag{3}$$

The SLAM algorithm consists of a two-step recursive procedure with a prediction step that uses the motion model to predict our next state, and a correction step that uses the observation model to refine our state estimate:

Prediction Step

$$\begin{aligned} &Pr(\mathbf{x}_t, \mathbf{m} \mid \mathbf{Z}_{\mathbf{0}:t-\mathbf{1}}, \mathbf{U}_{\mathbf{0}:t-\mathbf{1}}, \mathbf{x_0}) \\ &= \int Pr(\mathbf{x}_t \mid \mathbf{x}_{t-\mathbf{1}}, \mathbf{u}_t) Pr(\mathbf{x}_{t-\mathbf{1}}, \mathbf{M} \mid \mathbf{Z}_{\mathbf{0}:t-\mathbf{1}}, \mathbf{U}_{\mathbf{0}:t-\mathbf{1}}, \mathbf{x_0}) \mathrm{d}\mathbf{x}_{t-\mathbf{1}} \end{aligned} \tag{4}$$

Correction Step

$$\begin{aligned} &Pr(\mathbf{x}_t, \mathbf{M} \mid \mathbf{Z}_{\mathbf{0}:t}, \mathbf{U}_{\mathbf{0}:t}, \mathbf{x}_t) \\ &= \frac{Pr(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{M}) Pr(\mathbf{x}_t, \mathbf{M} \mid \mathbf{Z}_{\mathbf{0}:t-\mathbf{1}}, \mathbf{U}_{\mathbf{0}:t}, \mathbf{x}_t)}{Pr(\mathbf{z}_t \mid \mathbf{Z}_{0:t-1}, \mathbf{U}_{0:t})} \end{aligned} \tag{5}$$

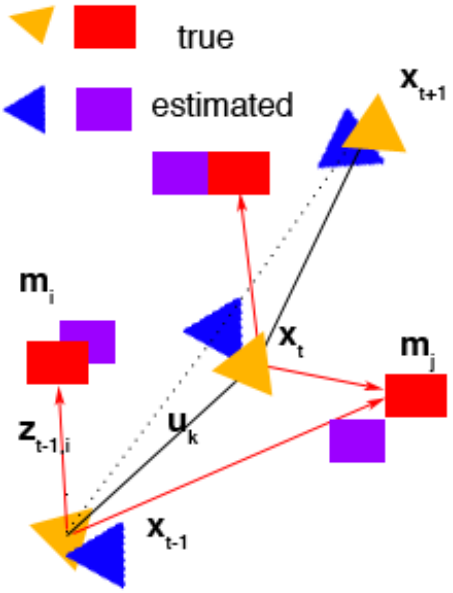


Figure 1: The SLAM problem. Triangles are robot positions and rectangles are landmark positions.

The key to SLAM is that the relative location between any two landmarks increases monotonically with the number of observations made [1]. This fact was determined by Newman et al. who proved that the estimated map converges monotonically to a relative map with zero uncertainty and that the absolute accuracy of the map and robot location approach a lower bound that is defined solely by the initial robot location uncertainty [5]. Their paper effectively proved the existence and convergence of a solution to the SLAM problem, which consequently means that it is possible to start at unknown location in an environment with no a priori information about the map of the environment, and incrementally construct a perfect map of the environment using only relative locations of landmark observations while at the same time achieving a bounded estimate of the robot location. Observations made by the robot are of the *relative* locations between landmarks, and these observations are 'nearly independent' because the observation errors will be correlated between movements of the robot [1].
In Figure 1, observe the robot at location x_{t-1}. The robot makes observations of landmarks m_i and m_j. When the robot moves to location x_t, the robot makes an observation measurement of m_j again. We can now update the estimated location of

the robot and the estimated location of the landmark observed relative to the robot's position at x_t. Because the relative locations of landmarks m_i and m_j are well known, the estimated locations at x_t back-propagate to update the position of m_i. As we observe new landmarks, these also become effectively 'linked' in the map with all other measurements by their relative positions. Successive observations only increase the accuracy of the relative locations, and thus the accuracy is bounded only by how accurate our starting position is.

2.3 Solutions to SLAM

So far we have described the SLAM problem and have noted that a solution is possible, but we have not yet discussed how to implement a solution to SLAM. The Extended Kalman Filter (EKF) and the Particle Filter are two of the most popular algorithms used to solve the SLAM problem. Each has their advantages, and the two methods are related as the particle filter is based on the Kalman Filter. While we chose to use a particle filter implementation for the mapping project, it is useful to briefly discuss the Extended Kalman Filter to note the particle filter's advantages over it.

2.3.1 Extended Kalman Filter SLAM

The Kalman filter is a type of Bayes filter, a recursive online algorithm which can be used to update a robot's belief state about its position from incoming sensor information. The Kalman filter uses Gaussian distributions to represent the motion model of the robot as well as the observation model. The filter is very efficient because it only has to update the Gaussian's mean and covariance during each prediction and correction step. The downside is that the assumptions made by the Kalman filter require that the initial belief state (of the robot location) be a Gaussian. This means that the robot's initial location must be known (to a degree of accuracy), or the robot will not be able to localize itself if it gets lost [7]. This is known as the "kidnapped robot problem." Besides the Gaussian assumptions that the Kalman filter makes, it also assumes that the models linear. However, in real-world applications this is often an

incorrect assumption. The Extended Kalman Filter is like the Kalman filter in that it assumes Gaussian error, but it is more attractive in the sense that its models need not be linear.

The EKF approximates the nonlinear models through a linear function obtained from first degree Taylor series expansions [6]. As in Durrant (2006)[1], the EKF describes the vehicles motion model as follows:

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_t) + \mathbf{w}_t \tag{6}$$

where f is determined by the robot's kinematics and $\mathbf{w}_t$ are additive, zero mean Gaussian random variables with covariance $\mathbf{Q}_t$. These Gaussians represent disturbances in the robot's motion, and are used to model uncertainty.

The observation model in the EKF is described as follows:

$$\mathbf{z}_t = h(\mathbf{x}_t, \mathbf{M}) + \mathbf{v}_t \tag{7}$$

where h represents the observation measurements and $\mathbf{v}_t$ are additive, zero mean Gaussian random variables with covariance $\mathbf{R}_t$. They are also used to represent measurement error.

With the Extended Kalman filter, we can compute the mean (our estimated state) and covariance of the joint posterior distribution $Pr(\mathbf{x}_t, \mathbf{M} \mid \mathbf{Z}_{\mathbf{0}:t}, \mathbf{U}_{\mathbf{0}:t}, \mathbf{x}_\mathbf{0})$ seen in section 2.2. Let $\mathbf{y}_t$ be the extended state vector comprised of $\begin{bmatrix} \mathbf{x}_{t|t} \\ \mathbf{m} \end{bmatrix}$ and let the observable state estimate $\hat{\mathbf{y}}_t$ be $\begin{bmatrix} \hat{\mathbf{x}}_{t|t} \\ \hat{\mathbf{m}}_t \end{bmatrix}$. The mean is computed as:

$$\hat{\mathbf{y}}_t = \mathbb{E}\left[\mathbf{y}_t \mid \mathbf{Z}_{\mathbf{0}:t}\right] \tag{8}$$

Recall that $\mathbf{Z}_{\mathbf{0}:t}$ is the set of all measurements taken up to and including time t. Also, we define error in the estimate to be $\mathbf{e}_t = \hat{\mathbf{y}}_t - \mathbf{y}_t$ The covariance matrix of the state is computed as follows:

$$\mathbf{P}_{t|t} = \begin{bmatrix} \mathbf{P}_{xx} & \mathbf{P}_{xM} \\ \mathbf{P}_{xM} & \mathbf{P}_{MM} \end{bmatrix}_{t|t} \tag{9}$$

$$= \mathbb{E}\left[\mathbf{e}_t \mathbf{e}_t^{\mathbf{T}} \mid \mathbf{Z}_{\mathbf{0}:t}\right] \tag{10}$$

In the covariance matrix, $\mathbf{P}_{xx}$ is the error covariance matrix associated with the robot state estimate, $\mathbf{P}_{MM}$ is the covariance matrix of the map associated with landmark state estimates, and $\mathbf{P}_{xM}$ is the cross-covariance matrix between the robot and landmark states [5]. The Extended Kalman Filter is comprised of a prediction step and a correction step, as in our SLAM formulation. Given an estimate $\hat{\mathbf{x}}_{t-\mathbf{1}|t-\mathbf{1}}$ of the state $\mathbf{x}_{t-\mathbf{1}}$ at time $k-1$ and an estimate of the covariance $\mathbf{P}_{t|t}$, the prediction step updates the following:

$$\hat{\mathbf{x}}_{t|t-\mathbf{1}} = f(\hat{\mathbf{x}}_{t-\mathbf{1}|t-\mathbf{1}}, \mathbf{u}_t) \tag{11}$$

$$\mathbf{P}_{xx,t|t-\mathbf{1}} = \mathbf{F}_{t-1}\mathbf{P}_{xx,t-1|t-1}\mathbf{F}_{t-1}^T + \mathbf{Q}_t \tag{12}$$

where $\mathbf{F}_{t-1}$ is the Jacobian of f evaluated at $\hat{\mathbf{x}}_{t-\mathbf{1}|t-\mathbf{1}}$. The correction step using the object measurements is accomplished as follows:

Let $\mathbf{S}_t = \mathbf{H}\mathbf{P}_{t|t-\mathbf{1}}\mathbf{H}^{\mathbf{T}} + \mathbf{R}_t$, and let $\mathbf{W}_t = \mathbf{P}_{t|t-\mathbf{1}}\mathbf{H}^{\mathbf{T}}\mathbf{S}_t^{-\mathbf{1}}$, where H^T is the Jacobian of h evaluated at $\hat{\mathbf{x}}_{t|t-\mathbf{1}}$ and $\mathbf{m}_{t-\mathbf{1}}$

Then the mean and covariance are updated:

$$\hat{\mathbf{y}}_t = \begin{bmatrix} \hat{\mathbf{x}}_{t|t-\mathbf{1}} \\ \hat{\mathbf{m}}_{t-\mathbf{1}} \end{bmatrix} + \mathbf{W}_t[\mathbf{z}_t - \mathbf{h}(\hat{\mathbf{x}}_{t|t-\mathbf{1}}, \hat{\mathbf{m}}_{t-\mathbf{1}})] \tag{13}$$

$$\mathbf{P}_{t|t} = \mathbf{P}_{t|t-\mathbf{1}} - \mathbf{W}_t\mathbf{S}_t\mathbf{W}_t^{\mathbf{T}} \tag{14}$$

The EKF possesses several desirable properties. One property is that the map converges, meaning that the error in relative position between landmarks decreases to a lower bound determined by the initial state accuracy, as proved in [5]. The fact that it can estimate non-linear models is another beneficial trait.. However, non-linearity is

only approximated by linear functions in EKF and can lead to inconsistent solutions as shown in Julier & Uhlmann (2001) [8]. It also assumes a Gaussian distribution in the models, a characteristic that is usually not found in real-world robotics [6]. The EKF is also limited in that it is very sensitive to incorrect data associations of landmarks. This happens when the robot incorrectly matches landmark m_i with landmark m_j [7]. This is very counterproductive in the context of SLAM, because SLAM depends on high correlations between landmarks. An additional restriction is computational complexity. In the correction step, all landmark locations and the joint covariance matrix must be updated every time an observation is made, which means that the time complexity is $O(|M|^2)$ where $|M|$ is the number of landmarks. In the original EKF approach, this means that the size of the map is usually limited to less than a thousand features [7]. However, efficient variations of the EKF SLAM algorithm have been created which allow for many thousands of map features [9] [10] [11].

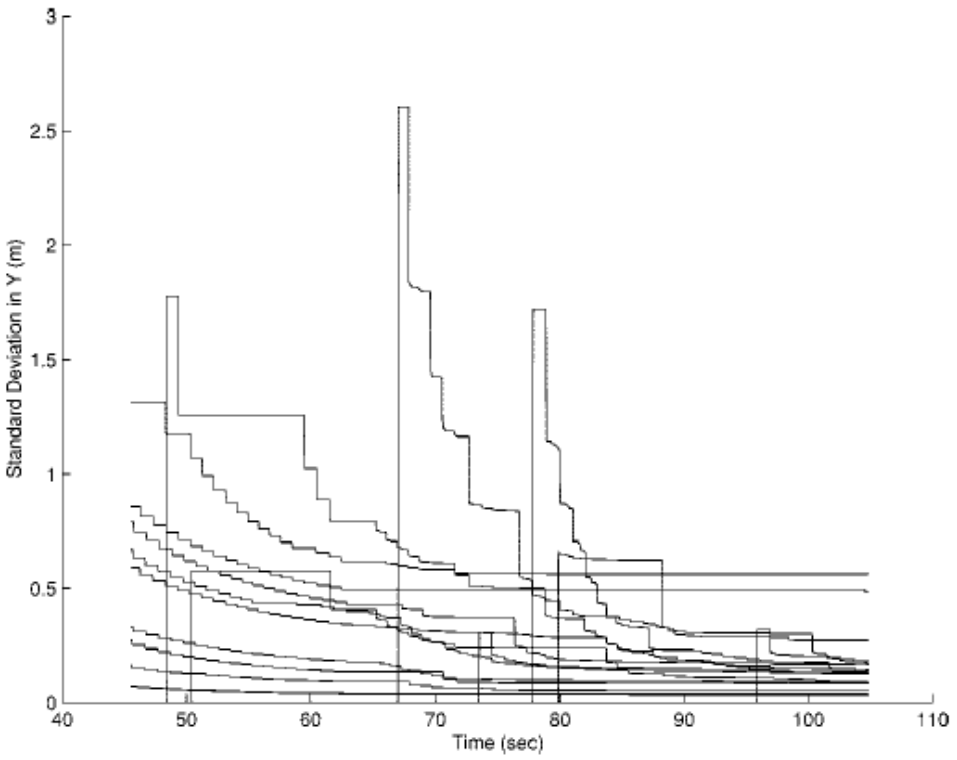


Figure 2: The standard deviation of landmark location estimates decreases monotonically to a lower bound. The spikes in standard deviation are due to the acquisition of new landmarks (from [5]).

2.3.2 Particle Filter

Another popular algorithm used in solving the SLAM problem is the particle filter. Whereas EKF SLAM represents the probability distribution of the robot state with a two-dimensional Gaussian, the particle filter implementation of SLAM represents the distribution as a set of samples, or particles, that are drawn from this distribution [7]. A particle filter is known as a sequential Monte Carlo method. Originally used in statistics and other related fields, the particle filter was made applicable to the SLAM problem by Murphy and Russell [14] using a type of particle filter called the Rao-Blackwell particle filter based on the works of Rao and Blackwell [12][13]. Rao-Blackwell particle filtering was made very efficient in the FastSLAM algorithm by Montemerlo et al.[15], a filter from which the particle filter we use is derived.

The FastSLAM algorithm[15] recursively estimates the full posterior distribution over robot states and landmark locations, but it does so much more efficiently than the EKF, taking only time logarithmic in the number of landmarks. The algorithm achieves this by exact factorization of the posterior distribution into a product of a distribution over robot paths and the conditional landmark distributions. Exact factorization is possible because of the fact that observations are conditionally independent of each other given a robot location and map.

In FastSLAM, the posterior is factored as follows:

$$Pr(\mathbf{X}_{\mathbf{0}:t}, \mathbf{M} \mid \mathbf{Z}_{\mathbf{0}:t}, \mathbf{U}_{\mathbf{0}:t}, \mathbf{N}_{\mathbf{0}:t}) = Pr(\mathbf{x}_t \mid \mathbf{Z}_{\mathbf{0}:t}, \mathbf{U}_{\mathbf{0}:t}, \mathbf{N}_{\mathbf{0}:t}) \prod_k Pr(\mathbf{m}_k \mid \mathbf{Z}_{\mathbf{0}:t}, \mathbf{X}_{\mathbf{0}:t}\mathbf{U}_{\mathbf{0}:t}, \mathbf{N}_{\mathbf{0}:t}) \tag{15}$$

where there are K landmarks, and $\mathbf{N}_{\mathbf{0}:t}$ is set of correspondences, or the indices of the landmarks perceived at times t, and k represents the index of a landmark. The FastSLAM algorithm uses a modified particle filter to estimate the path and estimates the landmark positions using a separate Kalman filter for each landmark. Each particle has its own path estimation and local landmark estimates (because these estimates

depend on the path estimation). Thus, for M particles and K landmarks, there are KM Kalman filters, each with dimension equal to the number of dimensions for landmark locations [15].

2.3.3 Path Estimation

The particle filter in FastSLAM has a fixed number of particles, M, and each particle m represents a guess of the robot's path. We obtain the most recent set of particles, $\mathbf{X}_t$ from the last set, $\mathbf{X}_{t-1}$, the control input to the robot $\mathbf{u}_t$, and a measurement $\mathbf{z}_t$. First, for each particle m at time t ($\mathbf{x_{t,m}}$), a motion model is used to predict a robot's state given $\mathbf{u}_t$ and the most recent set of particles $\mathbf{X_{t-1}}$:

$$\mathbf{x}_{t,m} \sim Pr(\mathbf{x}_t \mid \mathbf{u}_t, \mathbf{x_{t-1},}_m) \tag{16}$$

Each of these estimates is then added to a temporary set of particles, $\mathbf{X}'_t$. Each particle is given a weight, or importance factor, $w_{t,m}$. It is assumed that $\mathbf{S_{t-1}}$ is distributed according to $Pr(\mathbf{x_{t-1}} \mid \mathbf{z_{t-1}}, \mathbf{u_{t-1}}, \mathbf{n_{t-1}})$, and then the new particle is distributed according to $Pr(\mathbf{x_{t-1}} \mid \mathbf{z_{t-1}}, \mathbf{u}_t, \mathbf{n_{t-1}})$. This distribution is known as the *proposal distribution* of particle filtering [15]. We thus define the weight to be:

$$w_{t,m} = \frac{\text{targetdistribution}}{\text{proposaldistribution}} = \frac{Pr(\mathbf{x}_t \mid \mathbf{z}_t, \mathbf{u}_t, \mathbf{n}_t)}{Pr(\mathbf{x}_t \mid \mathbf{z_{t-1}}, \mathbf{u}_t, \mathbf{n_{t-1}})} \tag{17}$$

The new set of particles $\mathbf{X}_t$ is formed by drawing M particles, with replacement, from $\mathbf{X}'_t$ with probability proportional to $w_{t,m}$. Because the new particles only depend on the previous robot state, each particle's size is independent of time.

2.3.4 Landmark Location Estimation

Each landmark estimate in a particle is represented by a Kalman filter, and each is attached to a particle in $\mathbf{X}_t$. The contents of the set X_t are defined as the following for

every particle m:

$$\mathbf{X}_t = \{\mathbf{x}_{t,m}, \mu_{\mathbf{1},m}, \mathbf{\Sigma}_{\mathbf{1},m}, \cdots, \mu_{\mathbf{K},m}, \mathbf{\Sigma}_{\mathbf{K},m}\}_m \tag{18}$$

where $\mu_{k,m}$ is the mean of the m-th particle's k-th landmark's Gaussian, and $\Sigma_{k,m}$ is its covariance. In a two-dimensional map, the mean is a two-dimensional vector and the covariance is a 2x2 matrix; in a three-dimensional map, the mean is a three-dimensional vector and the covariance is a 3x3 matrix. To find the posterior over $Pr(\mathbf{m}_k)$, the position of the k-th landmark, we first need to know if the landmark was observed at time t. If it was not, the Gaussian does not change in the update step:

$$Pr(\mathbf{m}_k \mid \mathbf{X}_{\mathbf{0}:t}, \mathbf{Z}_{\mathbf{0}:t}, \mathbf{U}_{\mathbf{0}:t}, \mathbf{N}_{\mathbf{0}:t}) = Pr(\mathbf{m}_k \mid \mathbf{X}_{\mathbf{0}:t-\mathbf{1}}, \mathbf{Z}_{\mathbf{0}:t-\mathbf{1}}, \mathbf{U}_{\mathbf{0}:t-\mathbf{1}}, \mathbf{N}_{\mathbf{0}:t-\mathbf{1}}) \tag{19}$$

If the landmark is observed at time t, we modify the following as in [15]:

$$\begin{aligned}
&Pr(\mathbf{m}_k \mid \mathbf{X}_{\mathbf{0}:t}, \mathbf{Z}_{\mathbf{0}:t}, \mathbf{U}_{\mathbf{0}:t}, \mathbf{N}_{\mathbf{0}:t}) \propto \\
&Pr(\mathbf{z}_t \mid \mathbf{m}_k, \mathbf{X}_{\mathbf{0}:t}, \mathbf{Z}_{\mathbf{0}:t-\mathbf{1}}, \mathbf{U}_{\mathbf{0}:t}, \mathbf{N}_{\mathbf{0}:t}) Pr(\mathbf{m}_k \mid \mathbf{X}_{\mathbf{0}:t}, \mathbf{Z}_{\mathbf{0}:t-\mathbf{1}}, \mathbf{U}_{\mathbf{0}:t}, \mathbf{N}_{\mathbf{0}:t}) [\text{Bayes Rule}] \\
&= Pr(\mathbf{z}_t \mid \mathbf{m}_k, \mathbf{X}_{\mathbf{0}:t}, \mathbf{Z}_{\mathbf{0}:t-\mathbf{1}}, \mathbf{N}_{\mathbf{0}:t}) Pr(\mathbf{m}_k \mid \mathbf{X}_{\mathbf{0}:t-\mathbf{1}}, \mathbf{Z}_{\mathbf{0}:t-\mathbf{1}}, \mathbf{U}_{\mathbf{0}:t-\mathbf{1}}, \mathbf{N}_{\mathbf{0}:t-\mathbf{1}} [\text{Markov Assumption}]
\end{aligned} \tag{20}$$

FastSLAM uses an EKF for the update step, equation (20). Like other EKF SLAM implementations, it approximates the observation measurements with a linear Gaussian model. However, unlike the Extended Kalman Filter, FastSLAM's Gaussian is of dimension two or three as opposed to size $(dK + 3)$, with d dimensions, K landmarks, and 3 (or 4 if in 3D) elements of the robot state. Vanilla FastSLAM runs in $O(MK)$ time, but the version described in Montemerlo et al. runs in $O(M \log K)$ time by representing the set of Gaussians as a balanced binary tree. Additionally, FastSLAM is more likely to recover from data association failures than EKF SLAM is because it is able to pursue multiple data associations at the same time (through the use of particles). According to Montemerlo et al., the FastSLAM algorithm is able to create

accurate maps with a fixed number of particles, e.g. 100 particles. It has also been able to achieve this on maps with 50,000 features, a feat not possible for original implementations of EKF. FastSLAM's low time complexity and its ability to make accurate maps of complex environments make it a perfect choice to use in our underwater mapping.

2.4 Maltese Cistern Mapping Project

For our mapping project, we are building on an existing code base built by Prof. Clark and Billy McVicker as well as previous students at California Polytechnic State University. The previous code used the FastSLAM algorithm as specified in White et al. (2010) [4]. This version of FastSLAM uses a particle filter to construct occupancy grids and consists of a collection of M particles denoted as X_t that model the belief state. Each particle has an occupancy grid m_t with cells 0.20m by 0.20m in size, the robot's state x_t^k, and a weight w_t^k that represents the likelihood that the kth particle represents the true state. z_t represents that sensor measurements at time t, and u_t represents the control inputs to the robot at time t.

Algorithm 1 FastSLAM(X_{t-1}, u_t, z_t) from White et al. (2010)

$X'_t = X_t = 0$
for $k = 1 \rightarrow M$ **do**
  $x_t^k \leftarrow$ **sample_motion_model**(u_t, x_{t-1}^k)
  $w_t^k \leftarrow$ **measurement_model_map**$(z_{sonar,t}, u_t, m_{t-1}^k)$
  $w_t^k \leftarrow$ **measurement_smart_tether**$(\mathrm{z_{tether,t}}, \mathrm{u_t}, \mathrm{m_{t-1}^k}, \mathrm{w_t^k})$
  $m_t^k \leftarrow$ **updated_occupancy_grid**$(\mathrm{z_{sonar,t}}, \mathrm{u_t}, \mathrm{m_{t-1}^k})$
  $X'_t \leftarrow X'_t + \{x_t^k, m_t^k, w_t^k\}$
end for
for $k = 1 \rightarrow M$ **do**
  draw i with probability $\sim w_t^i$ from X'_t
  add $\{x_t^i, m_t^i\}$ to X_t
end for
return X_t

In order to account for tether snags and collisions with walls, the sample motion model is as modified as follows:

$$x_t^k = f[x_{t-1}^k, u_t(1+r_1) - \epsilon u_t(1+r_2)] \tag{21}$$

$$\epsilon = \begin{cases} 0 & \text{if } r_3 < \lambda \\ 1 & \text{else} \end{cases} \tag{22}$$

In the above equation, r_1 and r_2 are normally distributed random variables, r_3 is a uniformly distributed random variable, λ is the probability of a tether snag or collision, and ϵ is an indicator random variable that takes the value 0 if there is no snag or collision and 1 if there is.

Given the robot state x_t and the map m_{t-1}, an expected sonar measurement is calculated and compared to the actual sonar measurement z_{sonar}. If the two measurements are similar, a high weight is returned. If the two measurements are far apart, a low weight is returned. The weights for each particle based on the sonar measurements is calculated in **measurement_model_map** according to a Gaussian model as follows:

$$w_k = \sum_{i=1}^{B} \frac{1}{\sigma_z \sqrt{2\pi}} \exp\left[\frac{-(p_m^k - p_z)^2}{2\sigma_z^2}\right] \tag{23}$$

where B is the number of sonar measurement locations, and σ_z is the standard deviation of the model with expected probability p_m^k.

The expected smart tether measurement is the particle position x^k, and the **measurement_smart_tether** function strengthens the weight w^k calculated in the previous step if the x^k is similar to the actual smart tether measurement z_{tether}. Otherwise it decreases w^k. The weight is calculated according to the following Gaussian model:

$$w^k = w^k \frac{1}{\sigma_{tether}\sqrt{2\pi}} \exp\left[\frac{-(x^k - z_{tether})^T (x^k - z_{st})}{2\sigma_{tether}^2}\right] \tag{24}$$

The last function, **updated_occupancy_grid**, uses the new sonar measurements to update the map, with higher sonar signal return strengths associated with a higher

likelihood of a particular cell being occupied.

The last five lines of the algorithm are the resampling phase, where a new collection of particles X_t is generated by randomly selecting particles from X'_t to add, giving higher probability of selection to particles with higher weights (weights which have been normalized according to the sum of the particle weights).

While operating the ROV, sonar scans were taken so that they overlapped. This allowed for proper localization with the particle filter as it caused the relative locations of walls to be observed. Several different mapping techniques were used. First, sonar mosaics were created by manually stitching together raw sonar scans taken while the robot was stationary. This technique had the benefit of quickly being able to obtain a high quality map of an area, but was subject to human error:

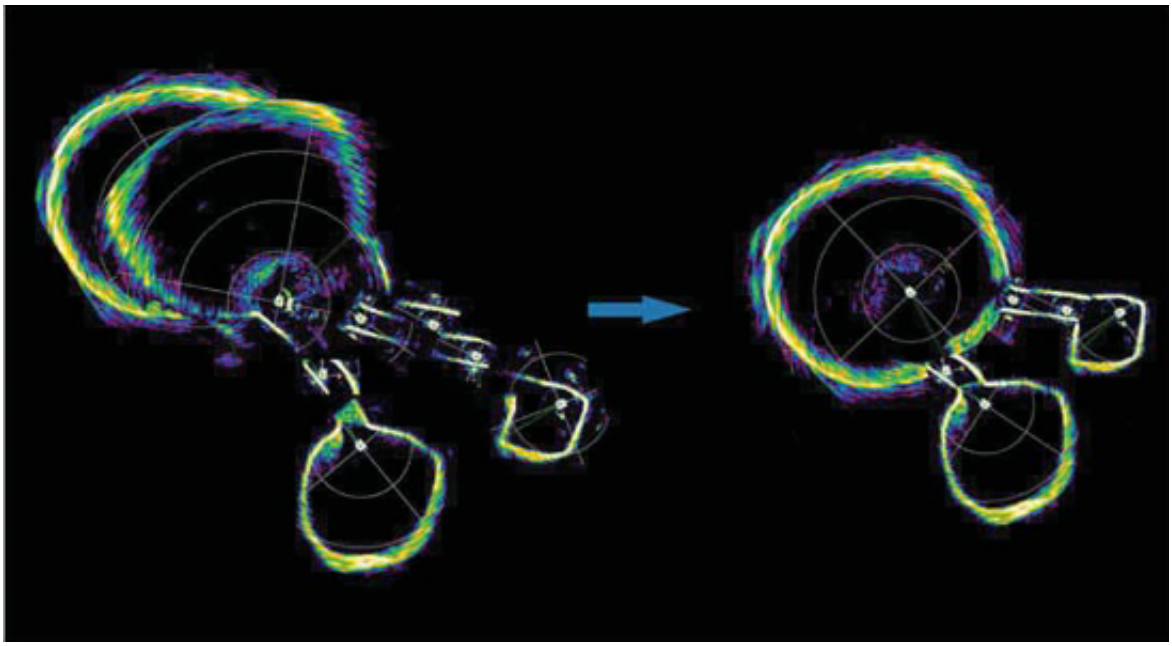


Figure 3: Scans from a monastery in Mdina, Malta. Seven overlapping scans were rotated to obtain a full map of the cistern, shown on the right. From White et al. [4]

Below is the result of running the mapping program on previously recorded sonar data in a cistern in Malta:

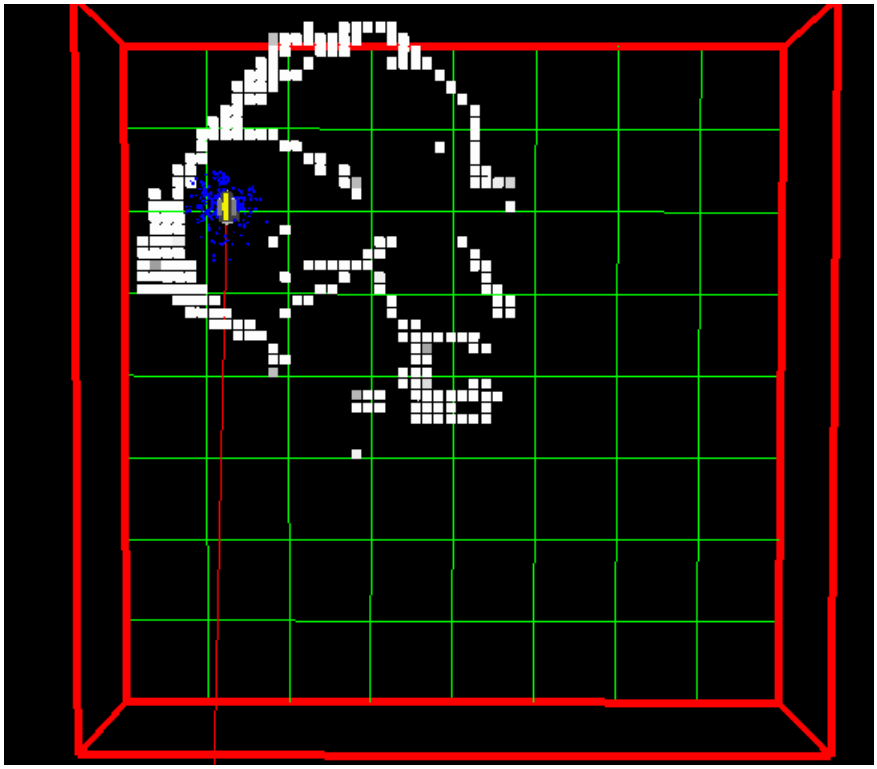
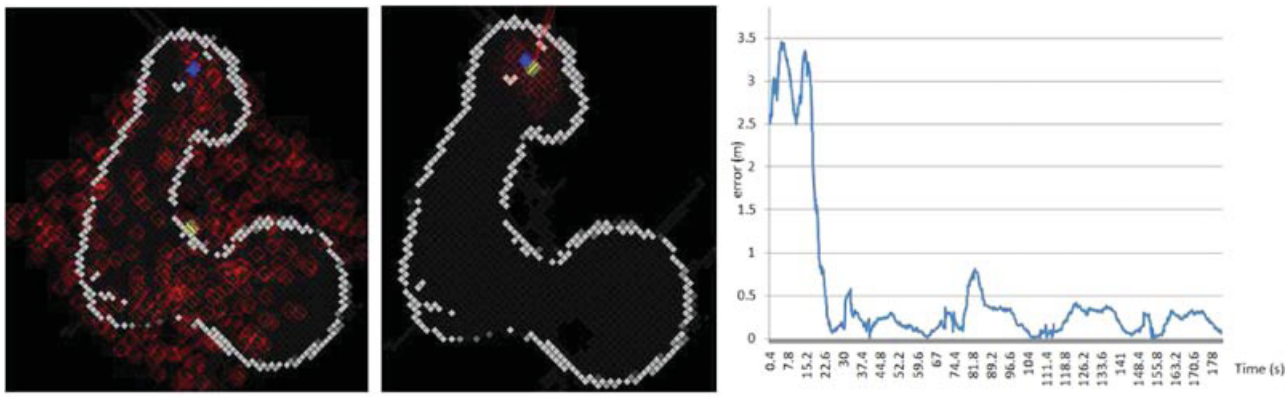


Figure 4: Map from sonar data

Other techniques involved stationary sonar mosaics using the Smart Tether for robot localization data, running SLAM with the ROV in motion, SLAM with stationary scans, and SLAM with smart tether data. It was determined that stationary SLAM was the most accurate method followed closely by manual sonar scan mosaics. SLAM while in motion was the least accurate because scanning while moving prevents one from obtaining full scans for any given location. This reduces the amount of overlap between features, hence lessening the accuracy of the relative locations of features.



(a) (b) (c)

Figure 5: In (a), the robot is at its initial state and its location is ambiguous, as the particles (red) are spread out randomly. After several sonar scans (b), the robot converges to its actual location (blue square). The error as a function of time is plotted in (c). From White et al.(2010) [4]

Even though there was no knowledge of the initial state of the robot, the FastSLAM algorithm used while the robot was stationary always allowed the robot to converge to within 0.5 m of its actual location [4]. Its localization accuracy was determined by flying the robot to directly below the access point for the cistern and observing the difference between estimated and true position.

3 Methods

The goal of this project was to apply the SLAM techniques to the mapping of underwater environments. The cisterns mapped in previous years have had relatively predictable geometries, and it is easy to obtain a three-dimensional map from a two-dimensional map by simply extrapolating the walls upward using the depth measurements from the ROV. Therefore, the two-dimensional SLAM algorithm used was perfect for those situations as it reduced the computational and coding complexity of adding a third dimension. However, the third dimension is essential to mapping complex environments such as marine caves. We used an underwater sonar for mapping data as well as a KCF Smart Tether and ROV sensors for localization data.

3.1 Equipment & Data Collection

We used a small VideoRay Pro 3 ROV as our robot of choice. Its small size makes it easily maneuverable, and it is equipped with horizontal thrusters, a vertical thruster for depth control, two front-facing halogen lights, a rear-facing LED light array, a forward facing camera, and a rear facing camera. The ROV is connected to a control box by a tether that sends control signals from the control box to the robot, and sends back depth and heading information. The control box contains a joystick for piloting the robot, knobs for depth control and lights, a video screen, and an LCD display showing time, depth gauge readings, compass direction, and time. The control box can connect to a laptop computer via USB and serial ports. In this way it is possible to interface with the robot directly from one's computer, allowing for autonomous control. However, none of the mapping was done autonomously for the sites we visited. A Tritech SeaSprite sonar was mounted to the top of the ROV to allow for scanning in the horizontal direction. In certain instances, the sonar was also mounted to the robot so that it scanned in the vertical direction. This allowed for the possibility of creating three-dimensional maps. The sonar was connected through the ROV to the tether, which sent signals to the control box which were then fed into a Panasonic Toughbook

computer. The SeaSprite software showed real time scans from the sonar, and provided the ability to save scans to .csv log files. The software also made it possible to change the sonar range, frequency, and gain.

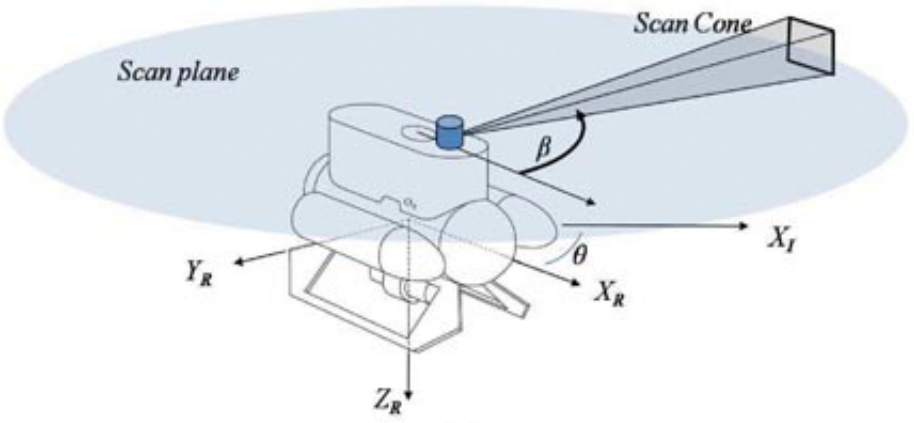


Figure 6: Model of the VideoRay Micro ROV. ROV is shown with the sonar mounted horizontally, with a head that rotates the sonar beam 360 degrees around the horizontal scan plane. X_R, Y_R, and Z_R represent the axes of the robot's local coordinate frame, while X_I represents the x-axis of the global (inertial) coordinate frame. θ is the robot's compass heading, with 0 degrees facing magnetic north. β is the angle between the scan beam and the robot's local x-axis. From White et al. (2010).

For certain sites, such as the marine cave and shipwreck site we visited, the underwater environment was too large to rely on motion models and observation models of the surrounding features alone. With such large areas, a position determined by the motion model would have accumulated a high amount of error over time, and the observation model may not be able to correct it if there are not enough overlapping features. Therefore, we decided to use the Smart Tether, manufactured by KCF, to enhance the localization accuracy of the ROV. The 40 meter long Smart Tether uses acceleration, magnetic, and rate-gyro sensors to track the position and measure the orientation of the ROV in real time. These sensors are embedded in six nodes along the length of the tether. Each node is six meters apart, except for the first two. The first node is attached to the top of the ROV and the second node is about a meter from the first

node. The smart tether comes with a GPS system that allows the operator to obtain absolute coordinates rather than coordinates relative to the base of the tether. This is especially important when the tether base is not stationary. In the case of the marine cave and shipwreck, we deployed the ROV from a 25-foot survey boat. We were fortunate enough to obtain a GPS fix which kept track of the base's position, because the boat kept drifting and our skipper had to constantly correct our position through adjustments to the throttle. The Smart Tether's control box connects to the ROV's control box as a normal tether does, and also has a USB port to connect to a computer. KCF's Smart Tether software was used on a Panasonic Toughbook to observe the location of the ROV in real time. The measurements were logged continuously while the robot was in the water, at the rate of about 5 Hz, and the measurements were saved into a .csv log file for use in the FastSLAM algorithm. In addition to sonar data and smart tether data, video from the ROV's front camera was also recorded. This was only used for archival purposes, however. It was not used in the SLAM algorithm.

Our team began logging smart tether data when the robot entered the water. The smart tether software requires the operator to keep track of how many nodes are currently in the water as well as the length of tether in between nodes that is in the water (to the nearest meter). Therefore, human error is a factor in the accuracy of the smart tether measurements. In normal usage, however, the smart tether is able to obtain accuracy better than 5 ft (1.5 m)[16]. Each team member was responsible for a certain task during the mapping process. These tasks included handling the tether and keeping track of the number of nodes and the extra length in between nodes currently in the water. Another student would sit at the Toughbook computer and operate the Smart Tether and sonar software while another student would drive the robot. We made decisions to record and save sonar scans based on the clarity of the image produced. For the case of the marine cave, scans were usually taken starting from the mouth of the cave and then subsequent scans were taken while going deeper into the cave. When it was time to take a scan, the computer operator would start recording a

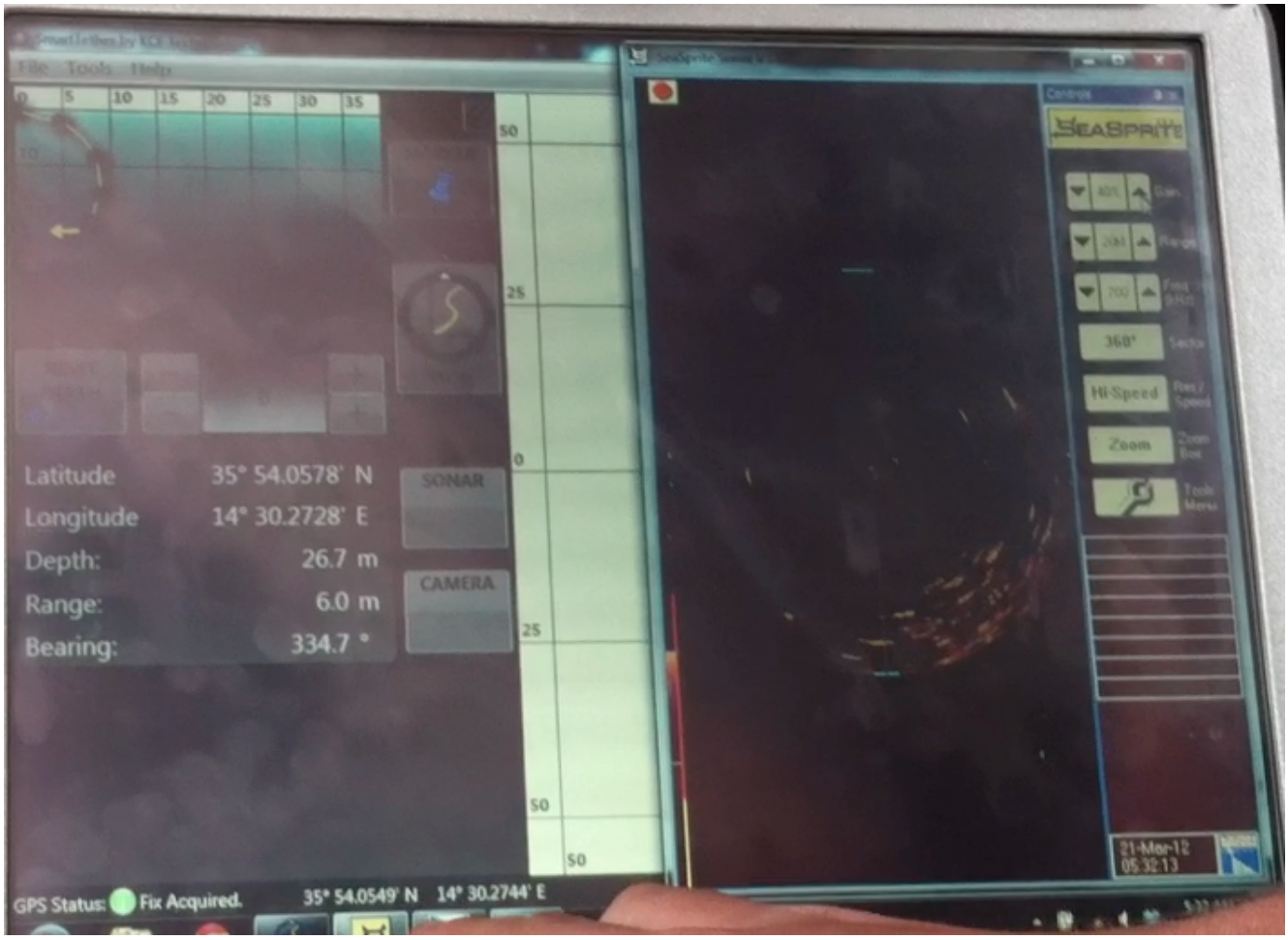


Figure 7: Toughbook computer displaying sonar information on the right side and smart tether information on the left side. Note that the smart tether gave depth and bearing information, but these values were not considered as accurate as the values from the ROV, so they were not used. The shape of the tether is displayed in real-time in the upper left-hand box.

sonar scan and stop recording a little over one rotation through. The ROV pilot would either have the ROV parked at the bottom of the cave so as to reduce variation in motion, or hover at a fixed position. Another student would add a sketch of the sonar scan to a rough hand-drawn map of the environment for later comparison, and record the compass heading, time, and depth reading of the ROV. This information was later turned into a log file for use in the localization part of the algorithm.

3.2 Occupancy grid

An occupancy grid approach was used to map our environments. We extended the occupancy grid approach used in the mapping of cisterns to three-dimensions, where the

map of the environment is discretized into of cubed cells, or voxels, of equal dimensions. We used 0.3m as the length, width, and height of each cell as it was determined that our measurements were not more accurate than that. An occupancy grid approach was used instead of other methods because no assumptions could be made about the geometric properties of the environments we were mapping beforehand. The occupancy grid is useful because it provides the flexibility needed to map to an arbitrary level of detail. If our measurements were more accurate, we could have simply decreased the size of each cell. The robot's position is kept track of in terms of real coordinates (which are more precise), not cell coordinates. The robot's state is represented as a vector $\mathbf{X}_t = [\mathbf{x}\ \mathbf{y}\ \mathbf{z}\ \theta]$. x, y, and z are the xyz-coordinates of the robot in the occupancy grid (with z increasing with depth) and θ is the angle between the robot and the robot's vertical axis and the global vertical axis.

Attached to each cell is a probability that the cell is occupied. Higher probability cells show up on the map as lighter than lower probability cells. Each cell's probability is determined by the particle filter when sonar scans and localization data are taken into account by the FastSLAM algorithm. Because of memory requirements, naively storing each cell in memory leads to inefficient results. Each particle that the particle filter uses stores its own map, and we would quickly run out of memory when using a higher number of particles if we just stored the full occupancy grid each time. Instead, Austin Walker, a member of the Princeton ICEX team, developed a hybrid octree data structure that stores the map, similar to the one in Fairfield et al. [17]. Octrees are trees in which every node has exactly eight children. They recursively subdivide a three-dimensional space into eight octants. They can be used to form a multi-resolution grid, meaning it is not necessary to store every cell in the grid. We can represent areas of the maps that are empty or sparsely populated by larger cells than the standard size. If we obtain data from a scan in this area, we can recursively divide the space into as many voxels as we need to obtain the proper resolution. In this way, our memory management becomes much more efficient and allows a particle filter to be run with a

higher number of particles (which usually increases its accuracy).

3.3 Particle filter

The particle filter used in this project is based on the FastSLAM variant given in White et al. (see section 2.4). It has been modified for use in our primary mapping environment, underwater marine caves. We have dispensed with the motion model in the prediction step, as we only take scans while stationary within the cave. We have also removed the step which weights the particle based on the data from the smart tether. Instead, for each scan we initialize the particles's XY location to the location given by the smart tether. We then distribute the particles around the initial position using a 4D Gaussian (in x, y, z, and θ), with standard deviation for the X-direction and Y-direction Gaussians equivalent to the tether accuracy of 1.5 m, and standard deviation for the Z-direction equal to the depth accuracy on the ROV. After some measurements of depth sensor error were taken in the DeNunzio Pool at Princeton, it was decided to use 0.2 m for the standard deviation of the depth sensor. The compass is assumed to be highly accurate, and the orientation Gaussian for θ was set to a standard deviation of 0.01 radians, or about 0.5 degrees. All Gaussians have zero mean.

The measurement model function that weights based on the sonar data has also been changed. Since we are now operating in a 3D environment while still only using 2D sonar, it is unlikely that we will achieve many scans that overlap. Indeed, we did not even try to attain overlap while taking scans in the cave, and our results confirm that scans do not overlap. However, because we have taken scans in the horizontal direction as well as vertical direction, it is possible that some of our horizontal and vertical scans intersect. If this is the case, it should result in a high weight for a particle. If it is not the case, the weight for the particle should not change. To determine intersections, a map of horizontal scans and a separate map of vertical scans are stored for each particle. We process horizontal and vertical scans in alternating order. At the particle weighting stage of the algorithm, it is determined whether the scan being processed is

horizontal or vertical.

If the scan is is horizontal, we look at the particle's map of vertical scans (referred to as the vMap). From the depth of the robot according to the particle, we extend a ray outwards in the XY plane for every value of the sonar bearing in the scan (we just rotate the ray clockwise around the vertical axis of the ROV in the XY plane as the). If this ray intersects a filled cell in the vMap (one that exceeds the probability for being a wall), we calculate the distance from the particle to this cell. This is our expected distance d_{exp_i} at angle α_i. For every d_{exp_i} we have, we return to the horizontal scan and for every angle α_i, we calculate the distance to the point where the sonar receives a high enough signal return strength for a wall to exist. This is our d_{actual_i}. For every i, we take the squared difference between d_{actual_i} and d_{exp_i} and return a weight from a Gaussian function:

$$w_i = \frac{1}{\sigma_z\sqrt{2\pi}} \exp\left[\frac{-(d_{exp_i} - d_{actual_i})^2}{2\sigma_z^2}\right] \tag{25}$$

where σ_z is the standard deviation. σ_z can be taken as 2$*$cellSize in this case, or 0.6. To compute the weight for a scan, we can average these weights w_i. If the weight w is greater than 0, then add it to the current particle weight.

We can follow a similar procedure for a vertical scan, but instead we look at the particle's map of horizontal scans (its hMap). For this, we must consider rays that are in the plane defined by the ROV's horizontal and Z-axis. Again, we calculate the expected distances to cells in this plane using the data from hMap, and for every angle α that we find a distance to a cell, we look at this angle α in the vertical scan and determined the actual distance to a wall from the scan. The weights are calculated in the same manner as before.

The algorithm that we actually use is slightly different , although in effect the same. For each iteration of its loop, the mapping software gets a chunk of data from a sonar scan file. This corresponds to one bearing in the scan. Therefore, FastSLAM only runs for a single sonar bearing, meaning we only have to check along the ray cast out by that

sonar bearing in each iteration of FastSLAM. If there is an expected distance found for this bearing, we look at the chunk of sonar data for an actual distance. If there is no expected distance found (i.e. none of the cells along the ray returned a high enough probability to be a wall) we exit the FastSLAM function, do nothing with the particle weight, and go to the next particle. In this way, most particles will have a weight of zero (because most won't have intersections), but those that do return a high weight, and because particle weights are normalized when they are resampled, particles for which an intersection exist have a very high probability of propagating throughout the algorithm. After we have gone through all the scans, the vMap and hMap of the highest weighted particle are combined into one map, and this is displayed as our final map. Below is the pseudocode for the new FastSLAM algorithm:

Algorithm 2 Mapping with FastSLAM

for each sonar scan s **do**
  for Particle p : Particles **do**
    Initialize p's position to Smart Tether/ROV Location Data + randomness
  end for
  for each sonar bearing β_i in s **do**
    if i mod resample rate = 0, RESAMPLE
    FASTSLAM: {
    $X'_t = X_t = 0$
    for $k = 1 \rightarrow M$ **do**
      $w_t^k \leftarrow$ **measurement_model_map**$(sonardata, particlemap)$
      $m_t^k \leftarrow$ **updated_occupancy_grid**$(sonardata, localizationdata)$
      $X'_t \leftarrow X'_t + \{\text{ROVstate } x_t^k, \text{ParticleMaps } m_t^k, \text{Weight } w_t^k\}$
    end for}
  end for
end for
RESAMPLE: {
for $k = 1 \rightarrow M$ **do**
  draw i with probability $\sim w_t^i$ from X'_t
  add $\{x_t^i, m_t^i\}$ to X_t
end for
return X_t }
Combine Horizontal/Vertical Maps of Highest Weighted Particle

3.4 Smart Tether Modeling

The smart tether gives location data in the form of a NMEA GPGGA String, a standard GPS string [18]. This gives lat/long coordinates that we wish to convert to X, Y coordinates in our local frame of reference (relative to the base). We can use Vincenty's formulae, which are used to obtain the distance between two points on the surface of a spheroid (the assumed shape of Earth), to obtain these relative coordinates. A detailed formulation is found is given by C. Veness [19].

The length of the smart tether is only 40 m, and therefore our ROV's range is quite limited. To extend the range, we added an 80 m long extension tether to the smart tether. However, the extension tether has no nodes and doesn't give us any position data. That means our only localization data once we put the extension in the water comes from the ROV depth sensor and compass. For this reason, the smart tether extension is usually used to drive the ROV deeper than what would be possible with the smart tether. This way, the X and Y coordinates are known from the Smart Tether, and the Z coordinate is known from the ROV depth sensor.

We did not use the extension for this purpose. Instead we used it to go further into the cave. This caused our scans in the deep sections of the cave to be drawn on top of the other scans because the smart tether had reached its maximum range and thus was placing scans at the same coordinates. We try to clean up the scans further into the cave by predicting the location of the ROV based on the ROV sensors and the extension tether length alone. The amount of extended tether in the water was kept track of by a team member, and this amount was generally accurate to 5 m. Note that we do not use the particle filter in this prediction.

The Smart Tether, according to its patent, can be modeled as a catenary curve. This holds for freshwater, where the smart tether is neutrally buoyant. The smart tether extension is negatively buoyant in fresh water, but we will assume that it can be

modeled as a catenary curve in saltwater.

The following is an example of a catenary curve:

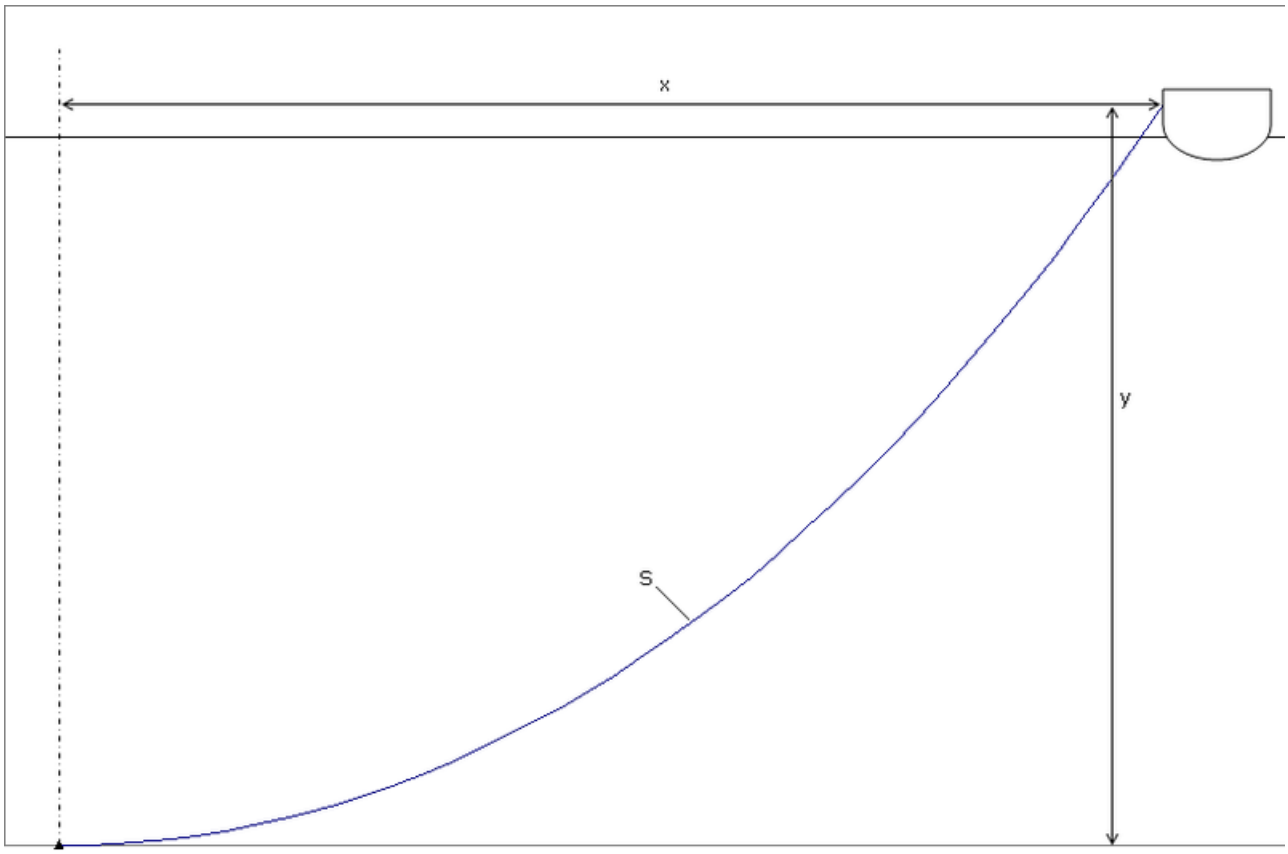


Figure 8: The catenary curve can be used to model the smart tether. In this drawing, s is the length of the curve (tether), x is the horizontal distance we are trying to find, and y is the vertical distance (difference in depth).Picture from http://en.wikipedia.org/wiki/File:Catenary.PNG

The equation of the catenary is $y = a \cosh \frac{x}{a} = \frac{a}{2}(e^{x/a} + e^{-x/a})$.
We are given the length l of the catenary curve (smart tether extension). Let v be the vertical distance from the base of the tether to the end, and let h be the horizontal distance. v is simply the new depth minus the previous depth and we are trying to find h. a is related to the tension in the tether. Experiments were conducted at DeNunzio pool with a tether, and the values of a range from 0.3 to 0.5. We will use 0.5 as a. We are trying to find k, the distance traveled between the previous position and the new position, and this can be calculated from the following formula [20]:

$$\sqrt{l^2 - k^2} = 2a \sinh \frac{h}{2a} \tag{26}$$

$$k = l^2 - 4a^2 \sinh^2 \frac{h}{2a} \tag{27}$$

Once we know k, we can calculate the translation in position. Let $\Delta\theta$ be the difference between the old ROV bearing and the new ROV bearing, Δy be the difference in vertical position, and Δx be the difference in horizontal position, and Δz be the difference in depth:

$$\Delta y = k \sin \Delta\theta \tag{28}$$

$$\Delta x = k \cos \Delta\theta \tag{29}$$

If (x_t, y_t, z_t) is the new position and $(x_{t-1}, y_{t-1}, z_{t-1})$ is the old position, then:

$$x_t = x_{t-1} + \Delta x \tag{30}$$

$$y_t = y_{t-1} + \Delta y \tag{31}$$

$$z_t = z_{t-1} + \Delta z \tag{32}$$

4 Results

The smart tether was a great tool to have while mapping. The scans near the mouth of the cave appear to be the cleanest, while the scans further in tend to become messier as errors in location accumulate. The vertical scans especially turned out nicely, and they give a good feel for the vastness of the cave. The vertical scans form a skeleton of the cave as they were taken relatively far apart from each other.

Unfortunately the particle filter did not perform as well as we had hoped. The fact that there were few intersections between scans limits the particle filter's effectiveness. One

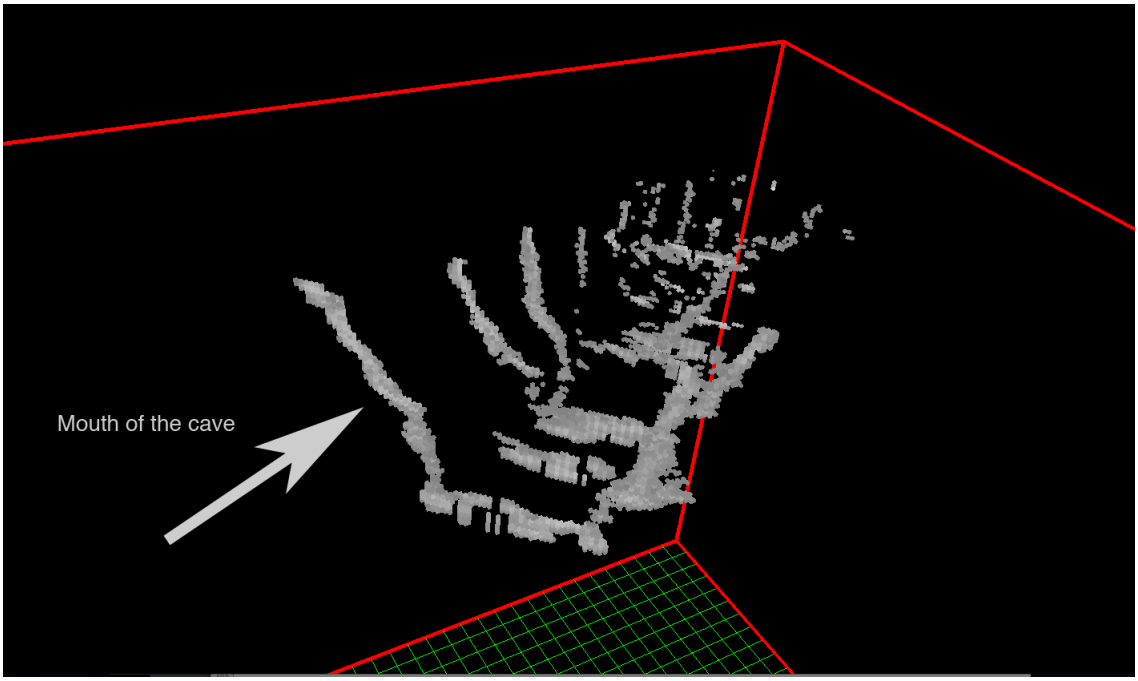


Figure 9: A map of the cave using only vertical scans. The scans form what look like ribs of a skeleton. Using these scans it is possible to extrapolate a solid wall between them. Note that this does not use the particle filter.

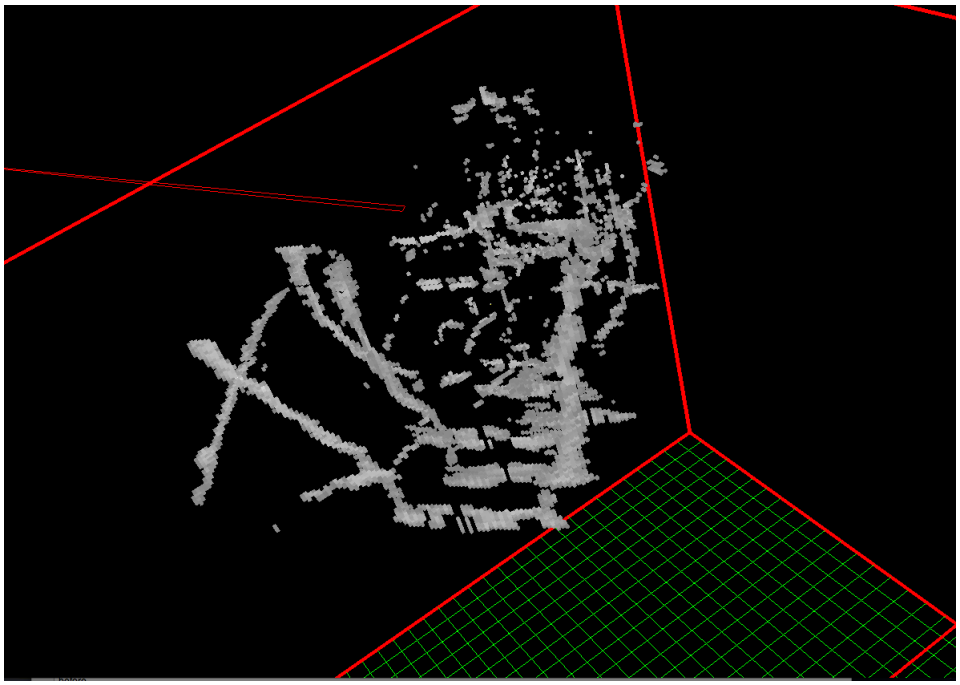


Figure 10: The map becomes much less clear when the horizontal and vertical scans are both drawn.

problem is with the scattered distribution of cells for each wall. If a wall-finding algorithm were used to connect nearby cells to form a wall, perhaps the particle filter could then find the intersections between horizontal and vertical scans. Perhaps more data would also help improve its effectiveness.

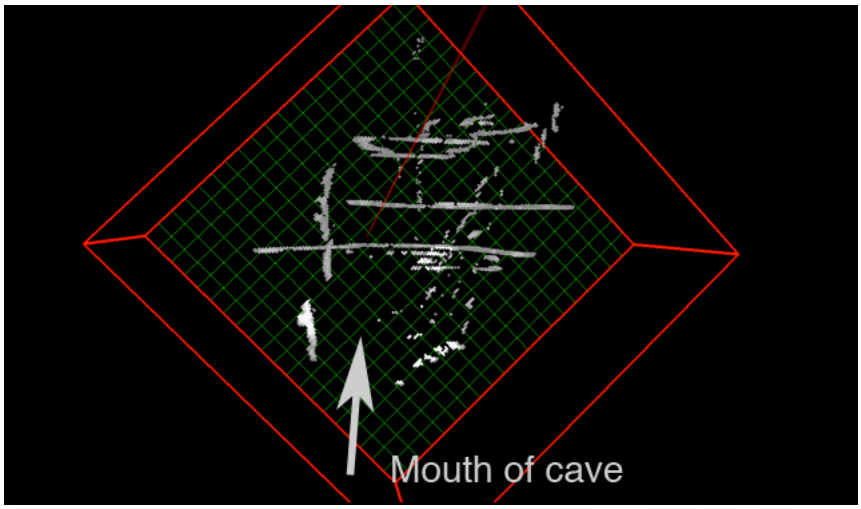


Figure 11: A top down view of the map when the particle filter is run (using 10 particles)

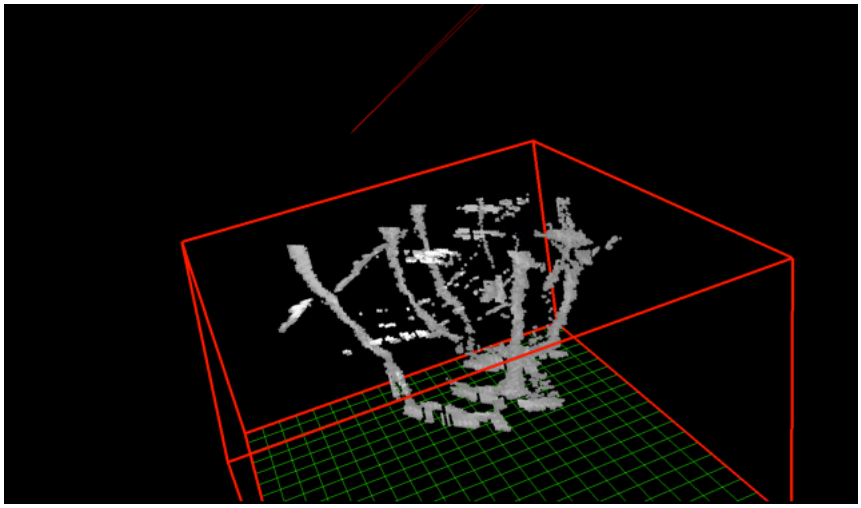


Figure 12: Another view of the same map after the particle filter (using 10 particles)

Even though the addition of horizontal scans produced a less than optimal map, we were able to extrapolate the walls of the cave from the vertical scans. Austin Walker wrote a grow walls function in the code that fits a wall through scans, the results of which are below:

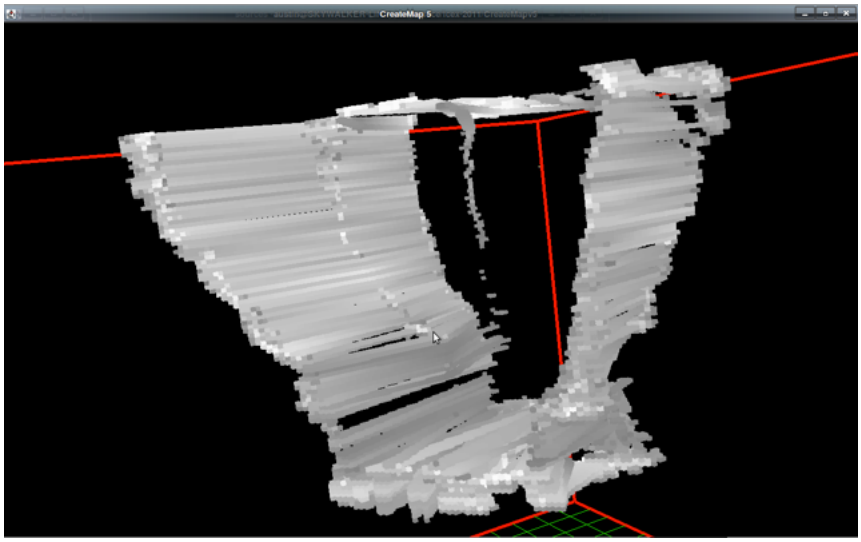


Figure 13: View from mouth of cave with walls filled in

The data from this map was sent to Jeff Forrester at Cal Poly who used his visualization software to create a realistic looking cave map with textures:

Figure 14: Visualization of the cave map with textures applied to walls

5 Conclusion

The use of the smart tether greatly helped simplify the mapping process. By not having to rely on motion models for the prediction step of the particle filter, we were able to reduce a lot of the error associated with such models. The smart tether was especially useful in the three-dimensional environments we mapped, because there was not much overlap between scans. We were able to produce good looking maps for the beginning of the cave, but this relied on hand-selecting the best scans to use. The results of the particle filter were less than optimal, but theoretically it is the correct approach given the constraints we had to deal with. Hopefully it can be developed further. Even though the maps were far from perfect, we were successfully able to accomplish our goal of extending the mapping algorithm for cisterns into three-dimensional environments.

6 Future Work

In the future, a side scanning sonar module will be mounted to the robot so that horizontal and vertical scans are produced at the same time, creating more accurate 3D maps.

Future work will also include rigorous testing of the Smart Tether to develop a better model for the error. Smart Tether extensions with sensor nodes can also be used to obtain more accurate localization data as the robot travels deeper into an environment.

One improvement in the SLAM algorithm that could be made is in determining the best order to process the sonar scans. By running the algorithm multiple times and comparing the standard deviation of the particles, we can pick the order of scans that produces the smallest standard deviation of particles, which should produce better scans.

7 Acknowledgements

I would like to thank my advisor, Prof. Christopher Clark, for his incredible help and guidance throughout the past year as well as for the opportunity to work on this exciting project.

I would also like to acknowledge the National Science Foundation as well as the Princeton University School of Engineering and Applied Science for their generous financial support of this project.

I would also like to acknowledge the owners of the private homes in Malta who allowed us to explore their homes' cisterns.

Many thanks to Dr. Timmy Gambin and the Aurora Trust for their continued support of the ICEX program.

Also, thank you to Keith Buhagiar and Dr. Matthew Montebello of the University of Malta.

Thank you to all member of the ICEX team, including Profs. Jane Lehr and Zoe Wood.

Thank you to the Princeton ICEX team, including Austin Walker who re-wrote and improved a large portion of the existing Malta mapping code.

Also, thank you to Prof. David Blei for accepting my request to advise me as the second reader of this thesis.

Last, but not least, I would like to thank my parents for giving me support during the time of writing this paper.

8 References

[1] Durrant-Whyte, H.; Bailey, T. (2006). *Simultaneous Localization and Mapping (SLAM): Part I The Essential Algorithms*. Robotics and Automation Magazine 13 (2): 99110.

[2] Smith, R.C.; Cheeseman, P. (1986). *On the Representation and Estimation of Spatial Uncertainty*. The International Journal of Robotics Research 5 (4): 5668.

[3] Durrant-Whyte, H.F. (1988). *Uncertain geometry in robotics*. IEEE Trans. Robotics and Automation, 4(1):23-31, 1988.

[4] White, C., Hiranandani, D., Olstad, C., Buhaglar, K., Gambin, T., & Clark, C. (2010, July). *The Malta Cistern Mapping Project: Underwater Robot Mapping and Localization within Ancient Tunnel Systems*. Journal of Field Robotics, 27 (4), 399-411.

[5] Dissanayake, M.W.M.G.; Newman, P.; Clark, S.; Durrant-Whyte, H.F.; Csorba, M.; *A solution to the simultaneous localization and map building (SLAM) problem* Robotics and Automation, IEEE Transactions on , vol.17, no.3, pp.229-241, Jun 2001

[6] Thrun, S. *Robotic mapping: A survey*. In G. Lakemeyer and B. Nebel, editors, Exploring Artificial Intelligence in the New Millenium. Morgan Kaufmann, 2002.

[7] Siegwart, R., Nourbakhsh, I., Scaramuzza, D. *Introduction to Autonomous Mobile Robots* MIT Press, Cambridge, MA, 2011.

[8] S.J. Julier and J.K. Uhlmann. *A counter example to the theory of simultaneous localization and map building*. In Proc. IEEE Int. Conf. Robotics and Automation, pages 4238-4243, 2001.

[9] J.E. Guivant and E.M. Nebot. *Optimization of the simultaneous localization and map-building algorithm for real-time implementation*. IEEE Trans. Robotics and Automation, 17(3):242-257, 2001.

[10] J.J. Leonard and H.J.S. Feder. *A computational ecient method for large-scale concurrent mapping and localisation*. In J. Hollerbach and D. Koditscheck, editors, Robotics Research, The Ninth International Symposium (ISRR'99), pages 169-176. Springer-Verlag, 2000.

[11] Bailey, T.. *Mobile robot localization and mapping in extensive outdoor environments*. Ph.D. thesis, University of Sydney, 2002.

[12] Blackwell, D. *Conditional expectation and unbiased sequential estimation* Annals of Mathematical Statistics 18: 105-110, 1947.

[13] Rao, C.R. *Information and accuracy obtainable in estimation of statistical parameters* Bulletin of the Calcutta Mathematical Society 37: 81-91, 1945.

[14] Murphy, K., Russell, S. *Rao-Blackwellized particle filtering for dynamic Bayesian networks* In Sequential Monte Carlo Methods in Practice, ed. by A Doucet, N. de Freitas, N. Gordon, 499-516, Springer, 2001.

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

[16] *Smart Tether*. http://www.smarttether.com/. Retrieved April 30, 2012.

[17] Fairfield, N., Kantor, G., & Wettergreen, D. (2006). *Real-time SLAM with octree evidence grids for exploration in underwater tunnels*. Journal of Field Robotics, 24, 321.

[18] DePriest, D. *NMEA data*. http://www.gpsinformation.org/dale/nmea.htm. Retrieved 2012-02-07

[19] Veness, C. *Vincenty formula between two Latitude/Longitude points*. http://www.movable-type.co.uk/scripts/latlong-vincenty.html. Retrieved 2012-02-07.

[20] Todhunter, Isaac. *XI Flexible Strings. Inextensible, XII Flexible Strings. Extensible*. A Treatise on Analytical Statics. Macmillan. 2002.