Passive Mobile Robot Localization within a Fixed Beacon Field

by

Carrick Detweiler

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2006

© Massachusetts Institute of Technology 2006. All rights reserved.

Author .....
Department of Electrical Engineering and Computer Science
September, 2006

Certified by .....
Daniela Rus
Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Passive Mobile Robot Localization within a Fixed Beacon Field

by

Carrick Detweiler

Submitted to the Department of Electrical Engineering and Computer Science
on September, 2006, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

This thesis describes a geometric algorithm for the localization of mobile nodes in networks of sensors and robots using bounded regions, in particular we explore the range-only and angle-only measurement cases. The algorithm is a minimalistic approach to localization and tracking when dead reckoning is too inaccurate to be useful. The only knowledge required about the mobile node is its maximum speed. Geometric regions are formed and grown to account for the motion of the mobile node. New measurements introduce new constraints which are propagated back in time to refine previous localization regions. The mobile robots are passive listeners while the sensor nodes actively broadcast making the algorithm scalable to many mobile nodes while maintaining the privacy of individual nodes. We prove that the localization regions found are optimal–that is, they are the smallest regions which must contain the mobile node at that time. We prove that each new measurement requires quadratic time in the number of measurements to update the system, however, we demonstrate experimentally that this can be reduced to constant time. Numerous simulations are presented, as well as results from an underwater experiment conducted at the U.C. Berkeley R.B. Gump Biological Research Station on the island of Moorea, French Polynesia.

Thesis Supervisor: Daniela Rus
Title: Professor

Acknowledgments

This research would not have been possible without the help and support of my advisor Daniela Rus. I also want to thank John Leonard and Seth Teller for discussions and advice. I would like to thank everyone in the Distributed Robotics Lab, but in particular Iuliu Vasilescu with whom I had many valuable discussions and who was the main person behind the hardware used and presented in this thesis. The experiments in Moorea could not have happened without everyone at the Gump research station and Michael Hamilton from the James Reserve who made our trip possible. I would also like to thank the CSIRO team–Peter Corke and Matthew Dunbabin–for their great support and collaboration and for helping us collect the data for the final experiment with their robot Starbug. Most of all, I want to thank my family, especially my fiancee Courtney Hillebrecht, for their love and support.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

Chapter 1

Introduction

The oceans of the world are the next frontier. We know more about galaxies a billion light years distant than we do about the floor of the ocean mere kilometers down. This lack of knowledge is astounding given that the ocean covers over seventy percent of the earth. We cannot hope to understand, let alone reverse, the effects of global warming if we do not understand how humans are affecting seventy percent of the earth.

The reason that we know so little is that the ocean is a foreboding environment. There is no air to breath–anyone going underwater must bring their own air supply. This is true in outer-space as well. Unlike outer-space, however, underwater the pressures exerted are far more extreme. In Space the lack of an atmosphere means that space vehicles must be built to deal with one atmosphere of pressure. Underwater, for every 10 meters that we travel down into the water we must deal with another atmosphere of pressure. At just 100 meters down every square meter has 100000 kilograms of force on it!

To prevent pressure related problems like the bends a diver must return to the surface very slowly. A diver diving at 30 meters can only spend 22 minutes to explore the bottom. In shallow waters dives can last as long as two hours, but this is still very limiting. Coral reef environments have thousands of species of fish and even more types of other creatures. We cannot hope to fully understand the complex ecology of the oceans with only these short expeditions. We must focus on techniques that

allow us to study the ecology of the ocean over larger time scales without the high level of human intervention currently needed.

Long term monitoring is best done by using sensors which can log the collected data over long periods of time. In order to obtain a large enough sample of the ocean these sensors must be easy to deploy and collect information from. Unfortunately, radio frequencies are absorbed by water. This makes deployment difficult as the sensor locations cannot be determined with GPS. Additionally, the data cannot be easily collected by using radio. Furthermore, acoustic communication underwater has huge power requirements and very low data transmit rates. Thus, alternate methods for deployment and data collection must be developed.

Current systems rely on divers to manually place each sensor, surveying each area so that the location of the sensors are known accurately. The sensors are left underwater for a few months and then a diver returns to take the sensor out of the water. The data is then downloaded from the sensor when on land and then the sensor is returned to its previous location.

This process, however, is not always successful. Sensors may fail while they are deployed. If failure happens months of data and much effort are lost. It would be best to check the sensors on a weekly or daily basis to make sure that they are still functioning, but this is currently not possible as it is a very labor intensive process. Additionally the number of times a sensor is removed and returned to the reef must be minimized as returning the sensors to the same location is difficult.

We wish to automate this process. To do this we need a hardware system which can be easily deployed and the data frequently collected. In our lab we have developed such a system of underwater sensors network and autonomous underwater robots shown in Figure 1-1. The sensors are easily placed in the water manually by one person. The sensors then automatically configure and localize themselves to create an underwater GPS-like system. The sensor nodes have acoustic communications which allows ranges between nodes to be obtained. In previous work we developed a static localization algorithm which robustly localizes these static nodes. Thus, we have a system which is able to be deployed with little human intervention.

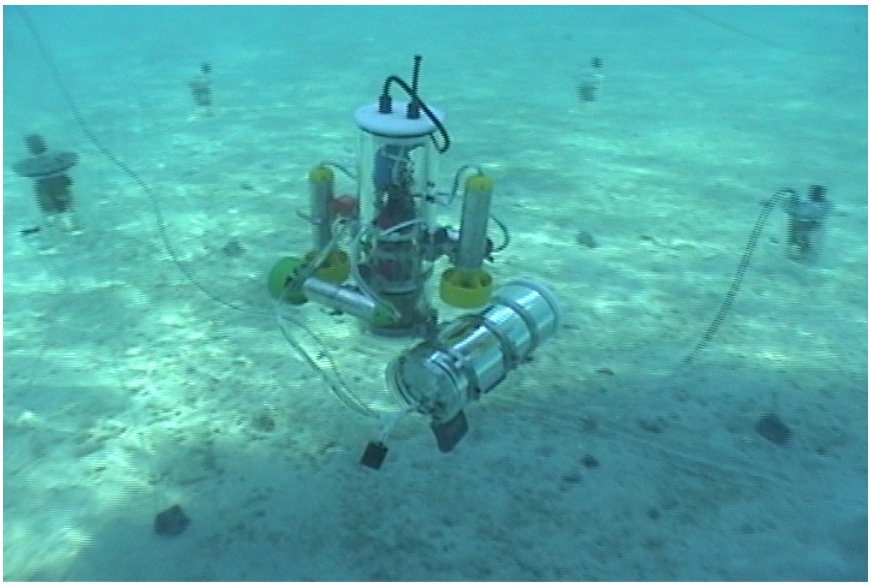


Figure 1-1: Our robot AMOUR with sensor boxes deployed at the Gump research station in Moorea.

The data collection problem is more challenging. The nodes could use their acoustic modems to send the data they collect back to scientists, however, acoustic modems have very limited bandwidth and require a lot of power. The modems are only able to transmit 300 *bits* per second. This makes the acoustic modems ill-suited for large amounts of data transmission. Instead, our system makes use of a robot which can autonomously visit each of the sensor nodes and retrieve the collected data using a high-bandwidth optical communications system. The robot can then surface and transmit the data over radio to the scientists. This allows near real-time monitoring of not only the location being studied, but also of the network itself.

The robot also is able to act as a mobile sensor node. If an interesting event is detected the robots can move to that location to provide a higher sampling density in that region. We envision systems with many robots to provide highly dynamic sampling capabilities. This will allow divers to focus on other important tasks besides the deployment and maintenance of the sensor network.

To enable data collection from the sensor nodes the robot must know precisely where it is as it moves through the sensor network. Additionally, to be useful as a mobile sensor node the robot must be able to even more accurately reconstruct where

it was when sensor readings were taken. The static localization algorithm does not work on the mobile robot because the ranges we obtain to the static sensor nodes are non-simultaneous. The robot will move significantly between range measurements and therefore the position computed will not be accurate.

Some systems exist for localization and tracking underwater, however, none are applicable to our system. One common method is dead-reckoning. Traditional in-air robots are able to use wheel encoders to obtain fairly good, inexpensive, dead-reckoning. Underwater, however, the unpredictable dynamics of the ocean (waves, current, etc.) make dead-reckoning very difficult. Relying on self-estimated speed and direction can lead to dead-reckoning errors on the order of 10% [1]. Using equipment which costs hundreds of thousands of dollars, such as advanced inertial navigation systems or Doppler velocity loggers, it is possible to have errors as low as 0.1% [1]. However, even relatively low errors will accumulate without bound over time. Thus, underwater it is impossible to rely on dead-reckoning information alone.

To rectify the unbounded dead-reckoning growth Long Base Line (LBL) networks have been used. In these systems dead-reckoning information is corrected by using range measurements to a set of sonar pinger buoys. These buoys are typically manually localized and deployed by a ship. The AUV then sends a sonar ping at a particular frequency and all of the LBL nodes respond to that ping at their own frequency. By measuring the round trip time of flight simultaneous range measurements are obtained. This method does not scale well to many mobile nodes. Additionally, the size of the region in which mobile nodes can be localized is effectively limited by the number of static nodes which is limited by the number of frequencies on which the LBL nodes can ping.

Most systems that use LBL networks rely on having dead-reckoning information. One of the goals of our system is to be inexpensive so that it is affordable to deploy large scale systems with many robots and sensors. Therefore, we do not have any advanced dead-reckoning equipment.

In this thesis, we present a localization algorithm for mobile agents for situations in which dead reckoning capabilities are poor, or simply unavailable. In addition to

underwater systems our algorithm is also applicable to the important case of passively tracking a non-cooperative target, other robot systems with limited dead-reckoning capabilities and non-robotic mobile agents, such as animals [3] and people [10].

Our approach, called passive localization in a beacon field, is based on a field of statically fixed nodes that communicate within a limited distance and are capable of estimating ranges to neighbors. These agents are assumed to have been previously localized by a static localization algorithm, such as our distributed localization algorithm using ranging based on work by Moore *et al.* [19]. A mobile node moves through this field, passively obtaining ranges to nearby fixed nodes and listens to broadcasts from the static nodes. Based on this information, and an upper bound on the speed of the mobile node, our method recovers an estimate of the path traversed. As additional measurements are obtained, this new information is propagated backwards to refine previous location estimates, allowing accurate estimation of the current location as well as previous states. This refinement is critical in the application to mobile sensor networks where it is important to know where sensor readings were taken as accurately as possible. The algorithm is passive because it is possible in our hardware implementation to obtain ranges by passively listening to broadcasts from the static nodes.

Our algorithm is a general distributed algorithm which can be applied to other measurements besides range e.g. angle-only, a combination, or any setup in which bounded locations are obtained periodically. In the range case we obtain an annulus/circle for each range. Our algorithm works in any system in which the mobile node can be bounded to some regions over time. We prove that our algorithm finds optimal localization regions–that is, the smallest regions that must contain the mobile node. The algorithm is shown, in practice, to only need constant time for each new measurement. We experimentally verify our algorithm both in simulations and in situ on our underwater robot and sensor system.

The passive localization algorithm is fully distributed and has nice scalability properties. Any number of robots can be localized within the field because the mobile nodes only need to listen passively to the broadcasts from the static nodes to obtain

ranges. No other information needs to be transmitted. The algorithm was shown to operate reliably on an underwater platform consisting of robots and sensor nodes.

This thesis is organized as follows. We first introduce the intuition behind the range-only and angle-only versions of the algorithm. We then present the general algorithm, which we instantiate with range and angle information, and prove that it is optimal. Then we discuss a simulator in which we implemented the range-only and angle-only algorithms. Finally, we present real-world experimental results of the range-only algorithm, including an ocean trial of the algorithm.

Chapter 2

Related Work

In this chapter we will explore in detail some other robot localization methods. We will start by relating the localization and tracking problem to classical motion planning. Then we will look at probabilistic techniques, systems that use bounded region models, geometric approaches and systems with limited dead-reckoning. Finally we will look at systems which have been used underwater.

2.1 Relation to Motion Planning

The robot localization problem bears similarities with the classical problem of robot motion planning with uncertainty. In the seminal work of Erdmann bounded sets are used for forward projection of possible robot configurations. These are restricted by the control uncertainty cone [11].

In our work the step to compute the set of feasible poses for a robot moving through time is similar to Erdmann's forward projection step in the preimage framework. We similarly restrict the feasible poses using the bound on the robot speed. The similarities come from the duality of these problems. Motion planning predicts how to get somewhere. In localization and tracking the goal is to recover where the robot came from.

Specifically, the preimage framework assumes an uncertainty velocity cone and an epsilon-ball for sensing. Within these uncertainty parameters the work computes

all the regions reachable by the robot such that the robot will recognize reaching the goal. Back-chaining can be used starting at the goal to compute a set of commanded velocities for the robot. In our work we use both forward and backward projections using a minimal amount of information, e.g. the bound on the maximum robot speed. Additionally, we iterate the forward and backward projection every time new information becomes available.

2.2 Probabilistic State Estimation Techniques

Many localization approaches employ probabilistic state estimation to compute a posterior for the robot trajectory, based on assumed measurement and motion models. A variety of filtering methods have been employed for localization of mobile robots, including extended Kalman filters (EKFs) [6, 15, 17], Markov methods [4, 15] and Monte Carlo methods [5, 6, 15]. Much of this recent work falls into the broad area of simultaneous localization and mapping (SLAM), in which the goal is to concurrently build a map of an unknown environment while localizing the robot.

Our approach assumes that the robot operates in a static field of nodes whose positions are known *a priori*. We assume only a known velocity bound for the vehicle, in contrast to the more detailed motion models assumed by modern SLAM methods. Most current localization algorithms assume an accurate model of the vehicle's uncertain motion. If a detailed probabilistic model is available, then a state estimation approach will likely produce a more accurate trajectory estimate for the robot. There are numerous real-world situations, however, that require localization algorithms that can operate with minimal proprioceptive sensing, poor motion models, and/or highly nonlinear measurement constraints. In these situations, the adoption of a bounded error representation, instead of a representation based on Gaussians or particles, is justified.

2.2.1 Kalman Filters

Kalman filters and extended Kalman filters have been used extensively for the localization of mobile robots [6, 15, 17]. The Kalman filter is a provably optimal recursive state estimator when the processes being measured and estimated are linear with Gaussian noise. Non-linear systems can be handled using an EKF, although it is no longer provably optimal. We will give a brief overview of Kalman filters, for a more detailed introduction see Welch and Bishop [28].

In the localization framework, a Kalman filter starts with an estimate of the position of the mobile node, a covariance matrix representing the Gaussian uncertainty of that position estimate, and a model of the node's uncertain motion. When new information about the location of the mobile node is obtained this information is mixed optimally with the information about the predicted new location of the node based on its previous location and its motion model. This ratio of the mixing is called the "Kalman gain" which is given by an equation that is provably optimal if the new measurement and motion have known Gaussian error characteristics. In addition to incorporating motion models and measurements into a Kalman filter, it is also possible to incorporate control inputs into the filter (e.g. to tell the filter the robot is turning).

The equations used are typically split into time update and measurement update equations [28]. The time update equations are:

$$\begin{aligned} \hat{x}_k^- &= A\hat{x}_{k-1} + Bu_{k-1} \\ P_k^- &= AP_{k-1}A^T + Q \end{aligned} \tag{2.1}$$

and the measurement update equations are:

$$\begin{aligned} K_k &= P_k^- H^T (HP_k^- H^T + R)^{-1} \\ \hat{x}_k &= \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \\ P_k &= (I - K_k H)P_k^- . \end{aligned} \tag{2.2}$$

Where the variables are described in Table 2.1.

$\hat{x}_k$	Location after incorporating latest measurement.
$\hat{x}_k^-$	Predicted location based on motion model and previous location.
A	Mapping of $\hat{x}_{k-1}$ to $\hat{x}_k$.
u_{k-1}	Control input.
B	Mapping of the control input into an affect on the motion.
P_k^-	The covariance before measurement incorporation.
P_k	The covariance after measurement incorporation.
Q	Noise in the process.
K_k	The computed Kalman gain term.
H	Mapping from state to the measurement.
R	Measurement noise
I	The identity matrix.

Table 2.1: The variables for the Kalman filter time (eqn. 2.1) and measurement (eqn. 2.2) update equations at some time k.

The EKF has a very similar set of equations, which are modified to provide good performance given non-Gaussian inputs. Unfortunately, the EKF is not provably optimal. See the paper by Welch and Bishop [28] for more details on the EKF.

Kalman filters (and EKFs) typically perform well given the right conditions. There are, however, a number of drawbacks which make them less than optimal to use in our localization setup. As can be seen in the filter equations, a lot of information must be known about the error characteristics of the motion and measurements. In our case we are assuming that we know very little about the motion characteristics of the mobile node–only an upper bound on speed.

Without a better model of the motion of the node the Kalman filter can be very sensitive to small changes in the other parameters (such as the process or measurement noise). These parameters are typically difficult to determine and change depending on local conditions, making it difficult to obtain good performance. Typically a badly tuned Kalman filter will either not smooth the path enough (leaving a jagged path) or it will smooth it too much (causing slow reactions to turns of the node).

Most localization algorithms which use Kalman filters avoid having to accurately model the motion of the robot by using dead-reckoning information. Dead-reckoning gives knowledge about the robot's current motion which can be fed into the filter.

Vaganay *et al.* [25], Olson *et al.* [21], Djugash *et al.* [6], Kurth [17], and Kantor and Singh [15] all use dead-reckoning as input into a Kalman filter to localize. Section 2.5 describe some systems which do not use dead-reckoning.

Another challenge is in the initialization of the filter. If a good estimate of the location of the mobile node is not know at the start then the filter might diverge. It is also possible that measurements that fall outside the measurement model will cause the filter to diverge and end up in a bad state. These states must be detected. One common way to do this is to use the Mahalanobis distance to determine if the filter is in a bad state [23]. However, when a bad state is detected, it is difficult to reinitialize the filter as a bad state could again be chosen.

Vaganay *et al.* [25] initialize their location in their range-only Kalman filter by fitting a line to the first N trilaterated locations. They assume that they are able to obtain synchronous range measurements and that their initial motion is a straight line. Djugash *et al.* [6] initialize their filter using a GPS reading. Kurth [17] performs experiments with different manually specified initial locations and concludes that poorly initialized filters never recover. Olson *et al.* [21] and Kantor and Singh [15] are SLAM algorithms which intrinsically address the initialization problem.

Kalman filters also have difficulties in cases where the probabilities involved are non-Gaussian. The extended Kalman filter handles this better than the Kalman filter, however it is still performs poorly with some distributions. Additionally, if the distribution is unknown, the filter will fail.

Finally, basic Kalman filters do not refine their previous location estimates when new measurements become available. This is problematic in our system where we need to know not our current location accurately, but also where we came from so that the sensor readings are meaningful. Kalman filters do not address this issue.

2.2.2 Markov and Monte Carlo Methods

Both Markov [4, 13, 15] and Monte Carlo [5, 6, 15] methods work by creating a discrete probability distribution over the space to represent where the mobile node could be. In Markov localization the space where the mobile node could be is divided

into a grid. Each cell in the grid is then labeled with a probability of the node being there. Monte Carlo methods, on the other hand, define a set of "particles." Each particle has a location and a probability of being in that location.

Both of these methods work well in the case where the initial location is unknown. They are able to handle this case because they have simultaneous estimates of where the node could be in the form of probabilities. Thus, as more measurements are obtained the probability at the true location will increase while the rest will decrease.

Another advantage of these methods is that they can model arbitrary probability distributions. However, since they are discrete in nature the memory requirements can be large if a highly accurate model of the distribution is desired. In fact, a Monte Carlo method representing all the probabilities as Gaussians becomes a Kalman filter as the sampling increases [13]. Both models use the same type of update equations as the Kalman filter to predict the motion of the mobile node.

The Markov method is very inefficient in storage space as it must store the entire space of possible locations when the mobile node can only be in a single location. Monte Carlo methods overcome this limitation by only storing locations of interest, allowing finer sampling in these areas. However, Monte Carlo methods can run into trouble if the sampling is too focused on the wrong area.

2.3 Bounded Region Models

Our algorithm assumes a bounded region representation. This makes sense in our framework as we do not have a probabilistic model of the motion. Thus, the mobile node could be in any reachable location with equal probability.

Previous work adopting a bounded region representation of uncertainty includes Meizel *et al.* [18], Briechle and Hanebeck [2], Spletzer and Taylor [24], and Isler and Bajcsy [14]. Meizel *et al.* investigated the initial location estimation problem for a single robot given a prior geometric model based on noisy sonar range measurements. Briechle and Hanebeck [2] formulated a bounded uncertainty pose estimation algorithm given noisy relative angle measurements to point features. Doherty *et al.* [8]

investigated localizations methods based only on wireless connectivity, with no range estimation. Spletzer and Taylor developed an algorithm for multi-robot localization based on a bounded uncertainty model [24]. Finally, Isler and Bajscy examine the sensor selection problem based on bounded uncertainty models [14].

2.4 Geometric Approaches

Researchers have also used geometric constraints to aid in localization. Olson *et al.* use a geometrically inspired approach to successfully filter out measurement noise before using an EKF to solve the SLAM problem [21]. While similar in concept, our algorithm is based on a different set of assumptions. Dead reckoning information is used as input to the filter in order to obtain information about beacon locations. Our solution does not rely on dead reckoning. Instead, we use the beacon locations to determine the mobile node trajectory.

Galstyan *et al.* use geometric constraints to localize a network of s tatic nodes by moving a mobile node through the network [12]. This system focuses on networks with bounded communication ranges and intersects discs formed to localize the network. This problem is in some senses an inverse problem from ours as we are trying to locate the mobile node instead of the network.

2.5 Limited Dead-reckoning

In previous work, Smith *et al.* also explored the problem of localization of mobile nodes without dead reckoning [23]. They compare the case where the mobile node obtains non-simultaneous ranges (passive listener) verses actively pinging to obtain synchronous range estimates. In the passive listening case an EKF is used. However, the inherent difficulty of (re)initializing the filter leads them to conclude a hybrid approach is necessary. The mobile node remains passive until it detects a bad state. At this point it becomes active to obtain synchronous range measurements. In our work we maintain a passive, non-simultaneous, state. Additionally, our approach intro-

duces a geometric approach to localization which can stand alone, as we demonstrate in simulation and experimentally, or be post-processed by a Kalman filter or other filtering methods as described in Chapter 4.3.

2.6 Underwater Localization

Underwater is an extremely challenging environment for localization. GPS signals are not present underwater, so underwater systems must make use of other methods to localize. Most systems rely heavily on dead-reckoning. Dead-reckoning errors accumulate over time leading to unbounded localization error. Even the largest and most expensive (hundreds of thousands of dollars) dead-reckoning systems can only achieve a drift of .1% [1]. While excellent for short-term deployment, this is unacceptable for long missions covering many kilometers. Typical, less expensive, dead-reckoning systems underwater have dead-reckoning errors on the order of 10% [1]. In our system we assume that there is no dead-reckoning information instead of trying to rely on highly erroneous information.

Regardless of the precision of the dead-reckoning other information must be used to bound the dead-reckoning error growth. The main way that this has been achieved underwater is using Long Base Line (LBL) systems [20, 21, 25]. More recently, some cooperative localization systems have been explored [1, 16]. In the following sections we will discuss these systems in some detail.

2.6.1 Long Base Line

LBL systems consist of a small number of static buoys deployed manually in the water. These buoys typically contain sonar pingers. When they receive a ping at a certain frequency they respond with a ping. Ranges can be obtained by measuring the round trip time of the ping. The response pings of the different buoys are on different frequencies so that simultaneous range measurements can be obtained, making localization easier.

The LBL buoys typically do not have any other means of communication with the

mobile node that is being localized. Because of this and since all of the buoys respond to pings the number of buoy that can be deployed in a single area is limited to the bandwidth of the acoustic communications channel. Typically at most four buoys are deployed. This limits the effective area of localization to the range of an individual LBL buoy (typically a few kilometers). This is fine for most current deployments, but is not sufficient to provide a GPS-type system over large areas of the ocean.

Vaganay *et al.* [25] use a Kalman filter to combine dead-reckoning information with the ranges obtained from LBL beckons. They test their system on an Odyssey II AUV. The setup of this system differs from our system in that we do not use dead-reckoning information (as it is not available to us) and we have more static nodes which have communication capabilities.

Traditionally, the individual LBL buoys had to be manually localized. This is done by a surface ship using GPS, a time intensive process. To try and avoid this much research has recently focused on automatically localizing the network using SLAM. Olson *et al.* [21] introduce geometric constraints to filter the range measurements obtained from the LBL system before applying a Kalman filter (see Section 2.4 for more details). The system requires dead-reckoning information. Newman and Leonard [20] use a non-linear optimization algorithm to solve for the vehicle trajectory as well as the LBL node locations. Their system assumes that the vehicle is traveling in a straight line at a constant velocity with Gaussian noise in the acceleration.

All of the systems that use LBL networks require detailed and accurate dead-reckoning information or limit the possible motions of the mobile node. Our approach to localization removes the necessity of dead-reckoning information. Additionally, our system (described in Chapter 5.2) is able to localize the static nodes autonomously which removes the necessity of performing SLAM.

2.6.2 Cooperative Localization

Another set of work related to our problem of localization underwater is the study of cooperative localization. In cooperative localization there is a team (or swarm) of underwater robots which need to be localized relative to each other and perhaps

globally.

Bahr and Leonard [1] and Vaganay *et al.* [26] look at a cooperative localization system between a team vehicles with different capabilities (e.g. some with expensive navigation equipment and others without). The vehicles with poor localization capabilities obtain ranges and location information from vehicles with good localization information. Vaganay *et al.* show that this type of system maintains accurate relative localization between the vehicles even after the absolute localization starts to drift due to inaccuracies in the more capable navigation systems [26]. Bahr and Leonard focus on minimizing the amount of communication that is needed while maximizing the information gain [1]. This is important in underwater systems where communication bandwidth is limited.

Kottege and Zimmer [16] present a localization system for swarms of robots underwater. In their system each robot is equipped with a number of acoustic receivers. Using these they are able to obtain range and angle between pairs of robots. This single reading gives them a relative location estimate between the pair which is sufficient for the swarm behavior in which they are interested. However, they do not discuss how this could be used to create a global coordinate system for the robots.

2.7 Summary

Table 2.2 summarizes the prior work on cooperative localization and lists several important attributes (if it is a SLAM algorithm, the region model, how the regions are initialized, if dead-reckoning is used, and how scalable it is). Our algorithm can be used to localize large numbers of mobile nodes. It is a distributed and scalable algorithm intended to work with any number of mobile nodes. Using a passive mode of our acoustic modems we are able to obtain ranges just by listening to a small number of localized nodes. Unlike these other algorithms our algorithm performs with the same accuracy regardless of the number of mobile nodes due to the passive nature of our mobile nodes.

A wide variety of strategies have been pursued for representing uncertainty in

robot localization. Our approach differs from the previous work in that it makes very minimalist assumptions on the information available for computation. We only require:

(1) A bound on the speed of the robot.

(2) Regions which bound the location of the robot at various times, possibly non-simultaneous times.

Using this minimal information we are able to develop a bounded region localization algorithm that can handle the trajectory estimation problem given non-simultaneous measurements. This scenario is particularly important for underwater acoustic tracking applications, where significant delays between measurements are common.

As new information is gained our algorithm refines previous location estimates. Most current work ignores the past and only tries to optimize the current location estimate. For mobile sensor networks, however, it is critical to know where sensor readings were taken. Additionally, our algorithm is distributed and supports scales well to many robots.

Reference	Approach	SLAM	Region Model	Region Init	Dead-reckoning	Scalable
Detweiler	geometric	no	bounded	automatic	no	yes
Bahr and Leonard [1]	cooperative	no	probabilistic	N/A	some	yes
Djugash *et al.* [6]	multiple	some	probabilistic	GPS	yes	some
Doherty *et al.* [8]	connectivity	no	bounded	automatic	no	yes
Galstyan *et al.* [12]	learning	no	geometric	automatic	no	yes
Gutmann[13]	Markov-Kalman	no	probabilistic	random	rough	yes
Kantor and Singh [15]	Multiple	yes	probabilistic	N/A	yes	possibly
Kottege and Zimmer [16]	bearing and range	no	N/A	none	N/A	yes
Kurth [17]	EKF	no	probilistic	manual	yes	possibly
Meizel *et al.* [18]	geometric	no	bounded	automatic	no	yes
Newman and Leonard [20]	non-linear opt	yes	probabilistic	N/A	yes	no–LBL
Olson *et al.* [21]	geometric and EKF	yes	probabilistic	N/A	yes	no–LBL
Smith *et al.* [23]	EKF and trilateration	no	probabilistic	trilateration	no	yes
Spletzer and Taylor [24]	bounded uncertainty	no	bounded	helpful	no–static	yes
Vaganay *et al.* [25]	Kalman	no	probabilistic	trilateration	yes	no–LBL
Vaganay *et al.* [26]	cooperative	no	probabilistic	N/A	some	yes

Table 2.2: A comparison of localization algorithms.

Chapter 3

The Algorithm for Passive Localization and Tracking

In this chapter we will formalize the problem and present our localization and tracking algorithm. The algorithm takes as input a set of non-simultaneous measurements and an upper bound on vehicle speed. Using this imput, the algorithm produces a set of optimal regions for each moment in time. These localization regions are the smallest which must contain the true location of the mobile node. As the algorithm receives new measurements the information gained from the new measurements are propagated backwards to refine previous location estimates.

In this chapter we prove the optimality of the algorithm and show that while worst case it is $O(n^3)$ for each new measurement added (where n is the number of previous measurements) in practice we can obtain a constant runtime. The algorithm is scalable to any number of mobile nodes without an impact on performance.

3.1 Algorithm Intuition

This section formally describes the localization problem we solve and the intuition behind the range-only version and angle-only version of the localization algorithms. The generic algorithm is presented in section 3.2.

3.1.1 Problem Formulation

We will now define a generic formulation for the localization problem. This setup can be used to evaluate range-only, angle-only, and other localization problems. We start by defining a *localization region*.

Definition 1 *A localization region at some time t is the set of points in which a node is assumed to be at time t.*

We will often refer to a localization region simply as a region. It is useful to formulate the localization problem in terms of regions as the problem is typically underconstrained, so exact solutions are not possible. Probabilistic regions can also be used, however, we will use a discrete formulation. In this framework the localization problem can be stated in terms of finding *optimal* localization regions.

Definition 2 *We say that a localization region is optimal with respect to a set of measurements at time t if at that time it is the* smallest *region that must contain the true location of the mobile node, given the measurements and the known velocity bound. A region is individually optimal if it is optimal with respect to a single measurement.*

For example, for a range measurement the individually optimal region is an annulus and for the angle case it is a cone. Another way to phrase optimality is if a region is optimal at some time t, then the region contains the true location of the mobile node and all points in the region are reachable by the mobile node.

Suppose that from time $1 \cdots t$ we are given regions $A_1 \cdots A_t$ each of which is individually optimal. The times need not be uniformly distributed, however, we will assume that they are in sorted order. By definition, at time k region A_k must contain the true location of the mobile node and furthermore, if this is the only information we have about the mobile node, it is the smallest region that must contain the true location. We now want to form regions, $I_1 \cdots I_t$, which are optimal given all the regions $A_1 \cdots A_t$ and an upper bound on the speed of the mobile node which we will call s. We will refer to these regions as *intersection regions* as they will be formed by intersecting regions.

3.1.2 Range-Only Localization and Tracking

Figure 3-1 shows critical steps in the range-only localization of mobile Node **m**. Node **m** is moving through a field of localized static nodes (Nodes **a**, **b**, **c**) along the trajectory indicated by the dotted line.

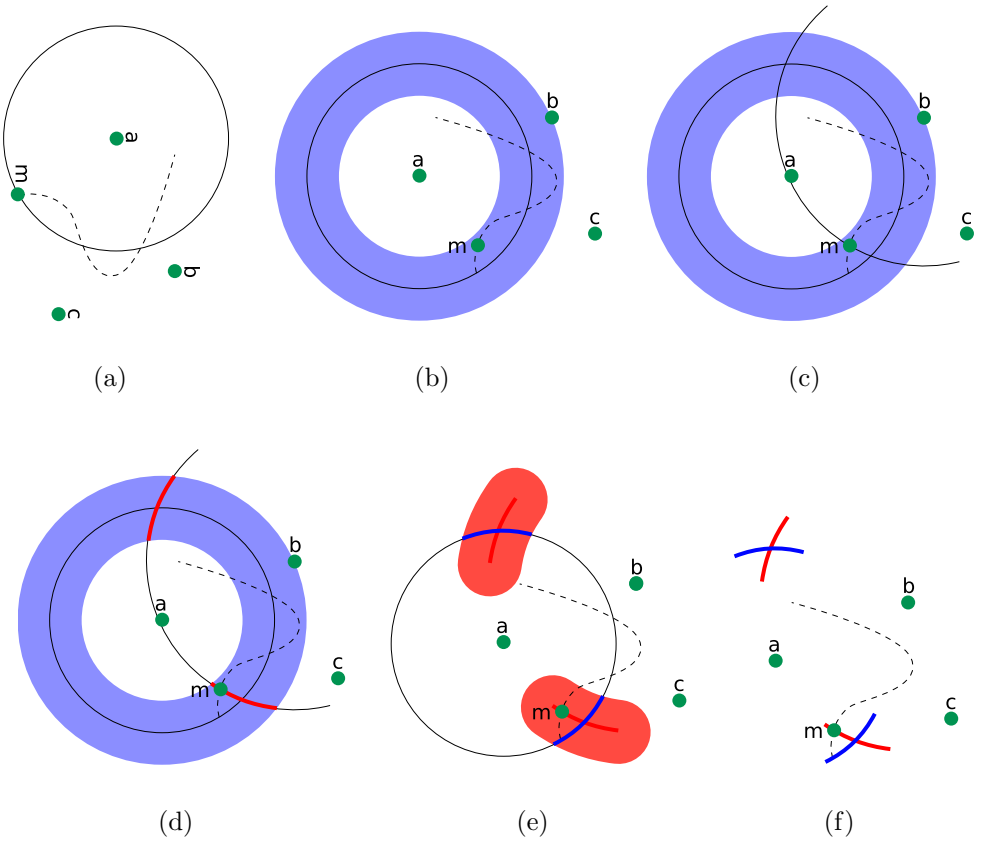


Figure 3-1: Example of the range-only localization algorithm.

At time t Node **m** passively obtains a range to Node **a**. This allows Node **m** to localize itself to the circle indicated in Figure 3-1(a). At time $t+1$ Node **m** has moved along the trajectory as shown in Figure 3-1(b). It expands its localization estimation to the annulus in Figure 3-1(b). Node **m** then enters the communication range of Node **b** and obtains a ranging to Node **b** (see Figure 3-1(c)). Next, Node **m** intersects the circle and annulus to obtain a localization region for time $t+1$ as indicated by the bold red arcs in Figure 3-1(d).

The range taken at time $t+1$ can be used to improve the localization at time t

as shown in Figure 3-1(e). The arcs from time $t+1$ are expanded to account for all locations the mobile node could have come from. This is then intersected with the range taken at time t to obtain the refined location region illustrated by the bold blue arcs. Figure 3-1(f) shows the final result. Note that for times t and $t+1$ there are two possible location regions. This is because two range measurements do not provide sufficient information to fully constrain the system. Range measurements from other nodes will quickly eliminate this.

3.1.3 Angle-Only Localization and Tracking

Consider Figure 3-2. Each snapshot shows three static nodes that have self-localized (Nodes **a**, **b**, **c**). Node **m** is a mobile node moving through the field of static nodes along the trajectory indicated by the dotted line. Each snapshot shows a critical point in the angle-only location and trajectory estimation for Node **m**.

At time t Node **m** enters the communication range of Node **a** and passively computes the angle to Node **a**. This allows Node **m** to estimate its position to be along the line shown in Figure 3-2(a). At time $t+1$ Node **m** has moved as shown in Figure 3-2(b). Based on its maximum possible speed, Node **m** expands its location estimate to that shown in Figure 3-2(b). Node **m** now obtains an angle measurement to Node **b** as shown in Figure 3-2(c). Based on the intersection of the region and the angle measurement Node **m** can constrain its location at time $t+1$ to be the bold red line indicated in Figure 3-2(d).

The angle measurement at time $t+1$ can be used to further refine the position estimate of the mobile node at time t as shown in Figure 3-2(e). The line that Node **m** was localized to at time $t+1$ is expanded to represent all possible locations Node **m** could have come from. This is region is then intersected with the original angle measurement from Node **a** to obtain the bold blue line which is the refined localization estimate of Node **m** at time t. Figure 3-2(f) shows the two resulting location regions. New angle measurements will further refine these regions.

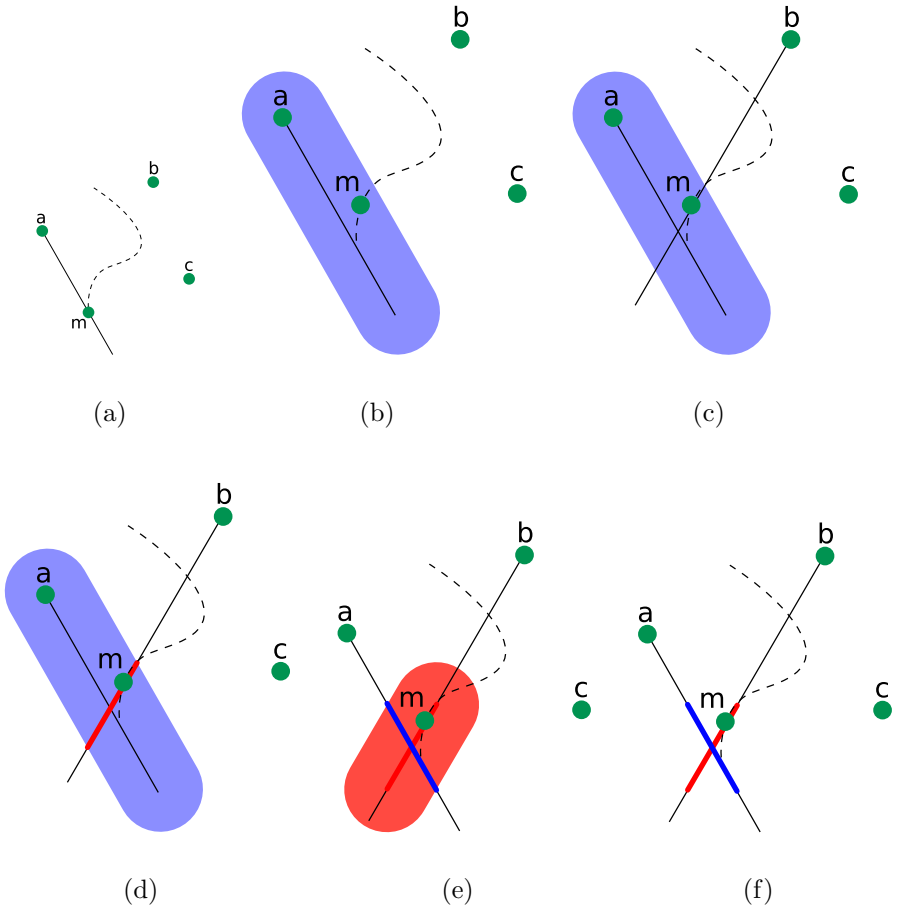


Figure 3-2: Example of the angle-only localization algorithm.

3.2 The Passive Localization Algorithm

3.2.1 Generic Algorithm

The localization algorithm follows the same idea as in Section 3.1. Each new region computed will be intersected with the grown version of the previous region and the information gained from the new region will be propagated backwards. Algorithm 1 shows the details.

Algorithm 1 can be run online by omitting the outer loop (lines 4-6 and 11) and executing the inner loop whenever a new region/measurement is obtained.

The first step in Algorithm 1 (line 3), is to initialize the first intersection region to be the first region. Then we iterate through each successive region.

Algorithm 1 Localization Algorithm

```
1:  procedure LOCALIZE(A_1 ··· A_t)
2:      s ← max speed
3:      I_1 = A_1                          ▷ Initialize the first intersection region
4:      for k = 2 to t do
5:          △t ← k − (k − 1)
6:          I_k = Grow(I_{k−1}, s△t) ∩ A_k      ▷ Create the new intersection region
7:          for j = k − 1 to 1 do                ▷ Propagate measurements back
8:              △t ← j − (j − 1)
9:              I_j = Grow(I_{j+1}, s△t) ∩ A_j
10:         end for
11:     end for
12: end procedure
```

The new region is intersected with the previous intersection region grown to account for any motion (line 6). Finally, the information gained from the new region is propagated back by successively intersecting each optimal region grown backwards with the previous region, as shown in line 9.

3.2.2 Algorithm Details

Two key operations in the algorithm which we will now examine in detail are `Grow` and `Intersect`. `Grow` accounts for the motion of the mobile node over time. `Intersect` produces a region that contains only those points found in both localization regions being intersected.

Figure 3-3 illustrates how a region grows. Let the region bounded by the black lines contain the mobile node at time t. To determine the smallest possible region that must contain the mobile node at time $t + 1$ we `Grow` the region by s, where s is the maximum speed of the mobile node. The `Grow` operation is the Minkowski sum [7] (frequently used in motion planning) of the region and a circle with diameter s.

Notice that obtuse corners become circle arcs when grown, while everything else "expands." If a region is convex, it will remain convex. Let the complexity of a region be the number of simple geometric features (lines and circles) needed to describe it. Growing convex regions will never increase the complexity of a region by more than

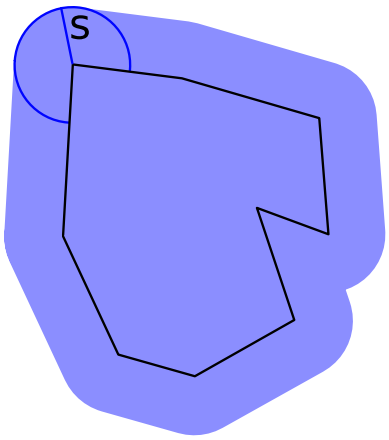


Figure 3-3: Growing a region by s. Acute angles, when grown, turn into circles as illustrated. Obtuse angles, on the other hand, are eventually consumed by the growth of the surroundings.

a constant factor. This is true as everything just expands except for obtuse angles which are turned into circles and there are never more than a linear number of obtuse angles. Thus, growing can be done in time proportional to the complexity of the region.

A simple algorithm for `Intersect` is to check each feature of one region for intersection with all features of the other region. This can be done in time proportional to the product of the complexities of the two regions. While better algorithms exist for this, for our purposes this is sufficient as we will always ensure that one of the regions we are intersecting has constant complexity as shown in Sections 3.3.2 and 3.3.2. Additionally, if both regions are convex, the intersection will also be convex.

3.3 Analysis

We now prove the correctness and optimality of Algorithm 1. We will show that the algorithm finds the location region of the node, and that this computed location region is the smallest region that can be determined using only maximum speed. We assume that the individual regions given as input are optimal, which is trivially true for both the range and angle-only cases.

3.3.1 Correctness

Theorem 1 *Given the maximum speed of a mobile node and t individually optimal regions, $A_1 \cdots A_t$, Algorithm 1 will produce optimal intersection regions $I_1 \cdots I_t$.*

Without loss of generality assume that $A_1 \cdots A_t$ are in time order. We will prove this theorem inductively on the number of range measurements for the online version of the localization algorithm. The base case is when there is only a single range measurement. Line 3 implies $I_1 = A_1$ and we already know A_1 is optimal.

Now inductively assume that intersection regions $I_1 \cdots I_{t-1}$ are optimal. We must now show that when we add region A_t, I_t is optimal and the update of $I_1 \cdots I_{t-1}$ maintains optimality given this new information. Call these updated intersection regions $I'_1 \cdots I'_{t-1}$.

First we will show that the new intersection region, I'_t, is optimal. Line 6 of the localization algorithm is

$$I'_t = \mathbf{Grow}(I_{t-1}, s\triangle t) \cap A_t. \tag{3.1}$$

The region $\mathbf{Grow}(I_{t-1})$ contains all possible locations of the mobile node at time t ignoring the measurement A_t. The intersection region I'_t must contain all possible locations of the mobile node as it is the intersection of two regions that constrain the location of the mobile node. If this were not the case, then there would be some point p which was not in the intersection. This would imply that p was neither in I_{t-1} nor A_t, a contradiction as this would mean p was not reachable. Additionally, all points in I'_t are reachable as it is the intersection of a reachable region with another region. Therefore, I'_t is optimal.

Finally we will show that the propagation backwards, line 9, produces optimal regions. The propagation is given by

$$I'_j = \mathbf{Grow}(I'_{j+1}, s\triangle t) \cap A_j \tag{3.2}$$

for all $1 \leq j \leq t-1$. The algorithm starts with $j = t-1$. We just showed that I'_t is optimal, so using the same argument as above I_{t-1} is optimal. Applying this recursively, all $I_{t-2} \cdots I_1$ are optimal. Q.E.D.

3.3.2 Computational Complexity

Algorithm 1 has both an inner and outer loop over all regions which suggests an $O(n^2)$ runtime, where n is the number of input regions. However, `Grow` and `Intersect` also take $O(n)$ time as proven later by Theorem 2 and 3 for the range and angle only cases. Thus, overall, we have an algorithm which runs in $O(n^3)$ time. We show, however, that we expect the cost of `Grow` and `Intersect` will be $O(1)$, which suggests $O(n^2)$ runtime overall.

The runtime can be further improved by noting that the correlation of the current measurement with the past will typically decrease rapidly as time increases. In Chapter 4.4.1 we show in simulation that only a fixed number of steps are needed, eliminating the inner loop of Algorithm 1. Thus, we can reduce the complexity of the algorithm to $O(n)$.

Range-Only Case

The range-only instantiation of Algorithm 1 is obtained by taking range measurements to the nodes in the sensor fields. Let $A_1 \cdots A_n$ be the circular regions formed by n range measurements. $A_1 \cdots A_n$ are individually optimal and as such can be used as input to Algorithm 1. We now prove the complexity of localization regions is worst case $O(n)$. Experimentally we find they are actually $O(1)$ leading to an $O(n^2)$ runtime.

Theorem 2 *The complexity of the regions formed by the range-only version of the localization algorithm is $O(n)$, where n is the number of regions.*

The algorithm intersects a grown intersection region with a regular region. This will be the intersection of some grown segments of an annulus with a circle (as shown in the Figure 3-4). Let regions that contain multiple disjoint sub-regions be called

compound regions. Since one of the regions is always composed of simple arcs, the result of an intersection will be a collection of circle segments. We will show that each intersection can at most increase the number of circle segments by two, implying linear complexity in the worst case.

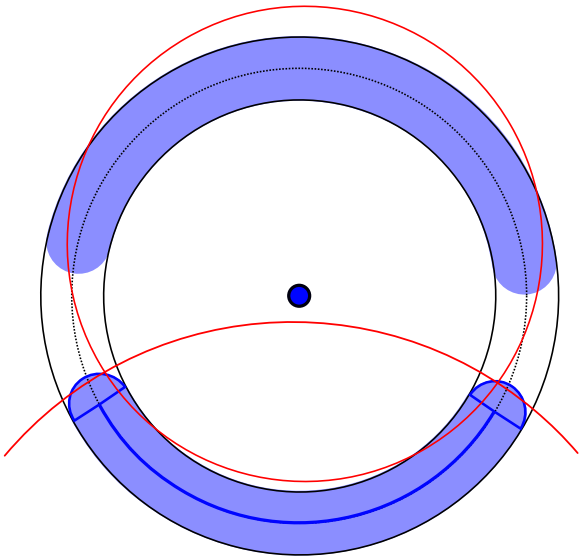


Figure 3-4: An example compound intersection region (in blue) and some new range measurement in red. With each iteration it is possible to increase the number of regions in the compound intersection region by at most two.

Consider Figure 3-4. At most the circular region can cross the inner circle of the annulus that contains the compound region twice. Similarly, the circle can cross the outer circle of the annulus at most twice. The only way the number of sub-regions formed can increase is if the circle enters a sub-region, exits that sub-region, and then reenters it as illustrated in the figure. If any of the entering or exiting involves crossing the inner or outer circles of the annulus, then it must cross at least twice. This means that at most two regions could be split using this method, implying a maximum increase in the number of regions of at most two.

If the circle does not cut the interior or exterior of the annulus within a sub-region then it must enter and exit though the ends of the sub-region. But notice that the ends of the sub-regions are grown such that they are circular, so the circle being intersected can only cross each end twice. Furthermore, the only way to split the

subregion is to cross both ends. To do this the annulus must be entered and exited on each end, implying all of the crosses of the annulus have been used up. Therefore, the number of regions can only be increased by two with each intersection proving Theorem 2.

In practice it is unlikely that the regions will have linear complexity. As seen in Figure 3-4 the circle which is intersecting the compound region must be very precisely aligned to increase the number of regions (note that the bottom circle does not increase the number of regions). In the experiments described in Chapters 4 and 5 we found some regions divided in two or three (e.g. when there are only two range measurements, recall Figure 3-1). However, there were none with more than three sub-regions. Thus, in practice the complexity of the regions is constant leading to $O(n^2)$ runtime.

Angle-Only Region Case

Algorithm 1 is instantiated in the angle-only case by taking angle measurements $\theta_1 \cdots \theta_t$ with corresponding bounded errors $e_1 \cdots e_t$ to the field of sensors. These form the individually optimal regions $A_1 \cdots A_t$ used as input to Algorithm 1. These regions appear as triangular wedges as shown in Figure 3-5(a). We now show the complexity of the localization regions is at worst $O(n)$ letting us conclude an $O(n^3)$ algorithm (in practice, the complexity is $O(1)$ implying a runtime of $O(n^2)$ see below).

Theorem 3 *The complexity of the regions formed by the angle-only version of the localization algorithm is $O(n)$, where n is the number of regions.*

Examining the Algorithm 1, each of the intersection regions $I_1 \cdots I_n$ are formed by intersecting a region with a grown intersection region n times. We know growing a region never increases the complexity of the region by more than a constant factor. We will now show that intersecting a region with a grown region will never cause the complexity of the new region to increase by more than a constant factor.

Each of these regions is convex as we start with convex regions and `Grow` and `Intersect` preserve convexity. Assume we are intersecting `Grow`(I_k) with A_{k-1}. Note

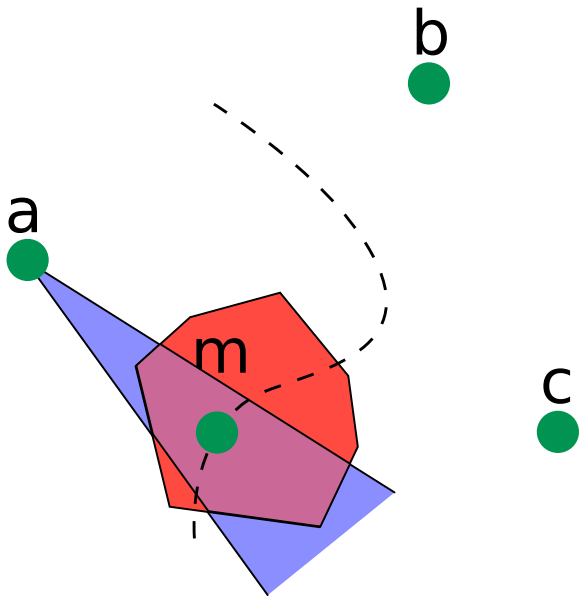


Figure 3-5: An example of the intersecting an angle measurement region with another region. Notice that the bottom line increases the complexity of the region by one, while the top line decreases the complexity of the region by two.

that A_{k-1} is composed of two lines. As $\texttt{Grow}(I_k)$ is convex, each line of A_{k-1} can only enter and exit $\texttt{Grow}(I_k)$ once. Most of the time this will actually decrease the complexity as it will "cut" away part of the region. The only time that it will increase the complexity is when the line enters a simple feature and exits on the adjacent one as shown in Figure 3-5. This increases the complexity by one. Thus, with the two lines from the region A_{k-1} the complexity is increased by at most two.

In practice the complexity of the regions is not linear, rather it is constant. When a region is intersected with another region there is a high probability that some of the complex parts of the region will actually be removed, simplifying the region. In Section 4.2 we show in simulation that complexity is constant for the angle-only case. Thus, the complexity of the regions is constant so the angle-only localization algorithm runs in $O(n^2)$.

Chapter 4

Implementation and Experiments in Simulation

We have implemented both the range-only and angle-only versions of the localization algorithm in a simulator we designed. In this chapter we will discuss the design of the simulator and illustrate it in action. We also look at a parametric study of the inputs to the algorithm.

The simulator is a Java application designed to give visual and numerical feedback on the performance of the localization algorithm. Figure 4-1 shows the simulator in action. A mobile node can be selected by clicking on it. The desired path for the selected mobile node can be drawn on the screen.

The measurements received can be customized to have particular error characteristics. The graphs on the right hand side record the error in the measurement being fed into the algorithm. The buttons and slider on the bottom allow the simulation to be paused, started, rewound, or saved. Additionally, there is a button which enables the recording of the sequence to a video.

On the right hand side there are also a number of check boxes which allow the user to display the recovered localization regions, an estimate of the path, a Kalman filtered version of the path, or an extended Kalman filtered path.

The simulator is implemented in such a way that makes it easy to try different types of inputs into the algorithm. The only methods that needs to be implemented

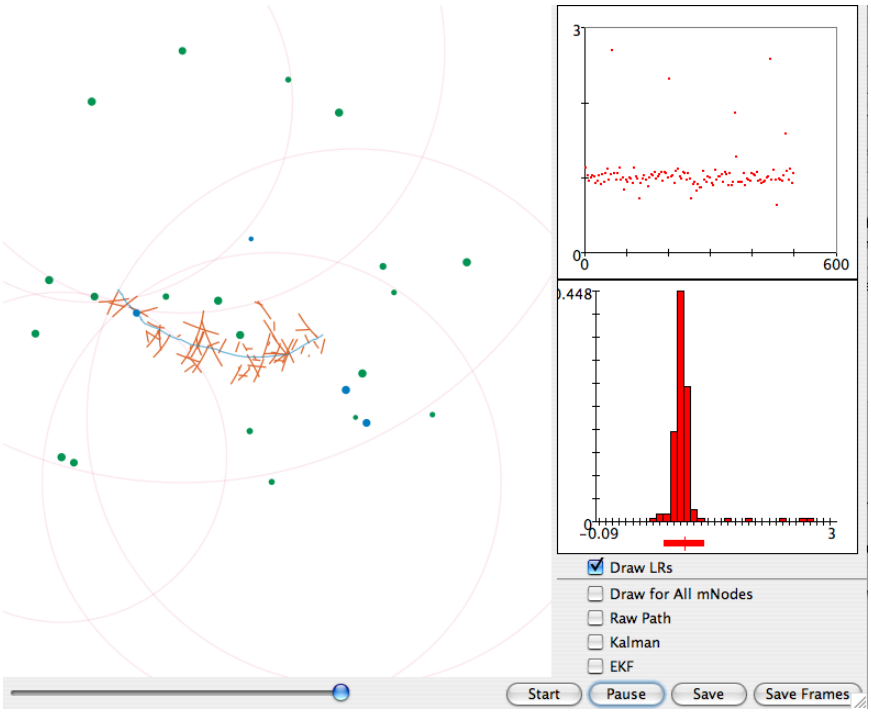


Figure 4-1: The simulator. A mobile node can be selected and its path drawn using a mouse. The plots at right show the probability distribution of the received ranges.

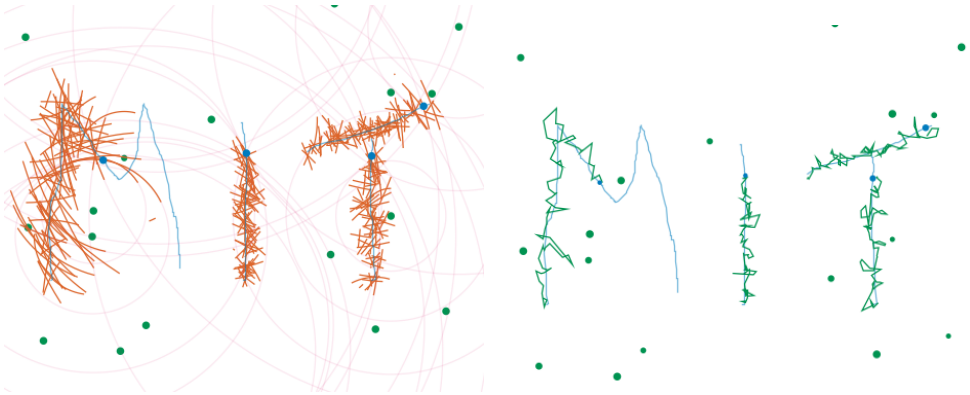
for different inputs are `intersect` and `grow` methods which tells the simulator how to perform the growing and intersecting that is needed for the localization algorithm. We have implemented both the range-only and angle-only versions, although any other measurement that produces bounded regions could be used.

The localization algorithm that we use in the simulator is almost a direct implementation of the algorithm discussed in Chapter 3. The main difference is that we do not propagate information from a new measurement all the way back. Instead, we only propagate it back a fixed number of steps. This, combined with the constant complexity of the regions, allows the algorithm to run in constant time per update. The main loop of the localization algorithm is shown below:

```
private void reallyUpdateLR(LocalizationRegion l){
  int nt = l.getTime();
  //Forward propagation
  LocalizationRegion reg = lr.getPrevious(l);
```

```
    l.intersect(reg,
                getMaxSpeed(reg.getTime(),nt)
                *(nt-reg.getTime()));
    //Back propagation
    for(LocalizationRegion reg : lr){
      /* only use it if it is a fairly recent measurement */
      if(nt - reg.getTime() <= pastTimeToProcess){
        reg.intersect(l,
                      getMaxSpeed(reg.getTime(),nt)
                      *(nt-reg.getTime()));
      }
    }
  }
```

4.1 Range-only



(a) (b)

Figure 4-2: Multiple mobile nodes drawing MIT. At left, the raw regions. Right, the recovered path using the center of each region.

Figure 4-2 shows the results of a range-only simulation with multiple nodes moving through the field of static nodes. This example uses a variety of mobile node speeds

ranging between .3m/s to .6m/s. The upper bound on the speeds of the mobile nodes was set fairly high to nearly 2.0m/s. Random measurement errors of up to 4% were used. Notice that all of the regions are fairly simple regions. We never found a region with a complexity greater than three in all of our simulations. A full analysis of the range-only system is presented in Section 4.4.

4.2 Angle-only

Figure 4-3 shows some snapshots of angle-only localization. The region shown is the current region where the mobile node is computed to be by the localization algorithm.

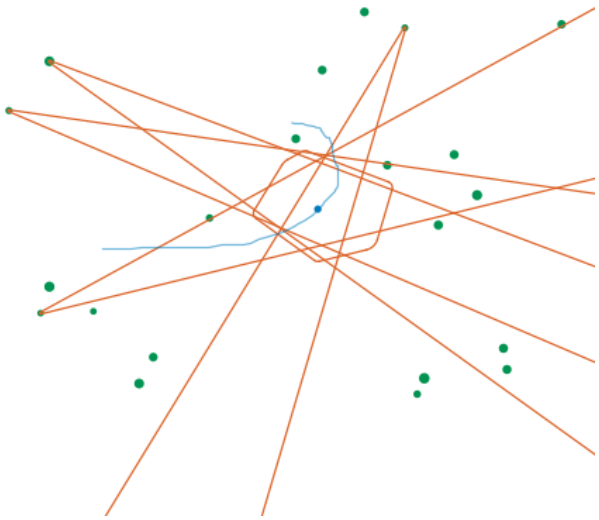


Figure 4-3: Angle-only simulation showing the current estimate of the mobile node and some recent measurements.

Figure 4-4 illustrates the results of a simulation where the complexity of a region was tracked over time. In the simulation a single region was repeatedly grown and then intersected with a region formed by an angle measurement from a randomly chosen node from a field of 30 static nodes. Figure 4-4 shows that increasing the number of intersections does not increase the complexity of the region. The average complexity of the region was 12.0 segments and arcs. Thus, the complexity of the regions is constant. Combining this with only a constant number of back propagations

we find that the angle-only algorithm only requires constant time for each new angle measurement.

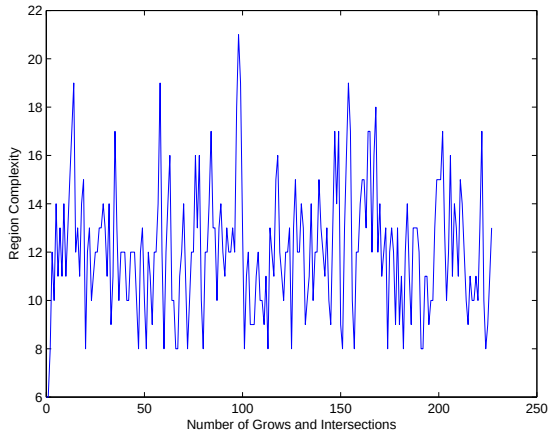


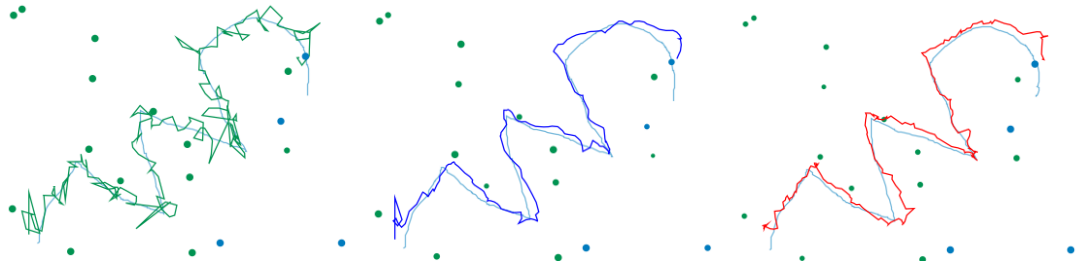
Figure 4-4: Growth in complexity of a localization region as a function of the number of *Grow* and *Intersect* operations.

4.3 Post Filters

There are a number of ways to compare the results of the algorithm to the true path of the mobile node. The first is to just determine if the true path is within the found region. This is useful, although for navigation purposes it is often useful to report a single point as the current location. To do this we implemented three different methods for recovering the path.

The first we call the "raw path." This, for the angle-only case, is the path formed by connecting the midpoints of the arcs formed by the localization algorithm. The second is using a Kalman filter as a filter on the raw path. This works, although it is not ideal as the range measurements introduce a non-linearity into the system. To better account for this we also implemented an extended Kalman filter which takes as input the regions found.

Figure 4-5 show a comparison of these three. Having a post-filter certainly smoothes the path, however, these filters require tuning for each particular setup



(a) (b) (c)

Figure 4-5: (a) Raw recovered path. (b) Kalman filtered raw path. (c) EKF path.

depending on factors such as the speed, acceleration, measurement certainty, etc. We found the tuning of these to be rather sensitive and difficult to optimize for even a single run. For instance, in order to get a smooth path while the mobile node was going straight required reducing the acceleration to the point where when the mobile node turned the filter would take too much time to compensate, leading to overshoots. This effect can be seen in Figure 4-5. Due to these effects it is difficult to use these filters on real-world systems where tuning online can be difficult or impossible. Section 4.4.6 contains an analysis of the error of the three different post filters under varying conditions.

4.4 Parametric Study

In this section we explore the effect of various parameters on the performance of the range-only implementation of the localization algorithm. We measured the performance by looking at the average difference in the actual location of the mobile node to that of raw recovered path. Unless otherwise noted we used the exact range measurements and did not introduce any noise. Each of the data points was obtained by averaging together the results of multiple (typically 8) trials, each with a different pseudo-random path. In all cases the paths stayed nearly within the convex-hull of the static nodes. The static nodes were randomly distributed over a 300 meter by 300 meter area.

4.4.1 Back Propagation

In this study we examined the effect of changing the amount of back propagation time. The results can be seen in Figure 4-6. We explored this using one mobile node traveling at .3m/s using an upper bound on the speed of .3m/s (solid line) and .6m/s (dotted line). There were twenty static nodes and a range was taken to a random static node every 8 seconds.

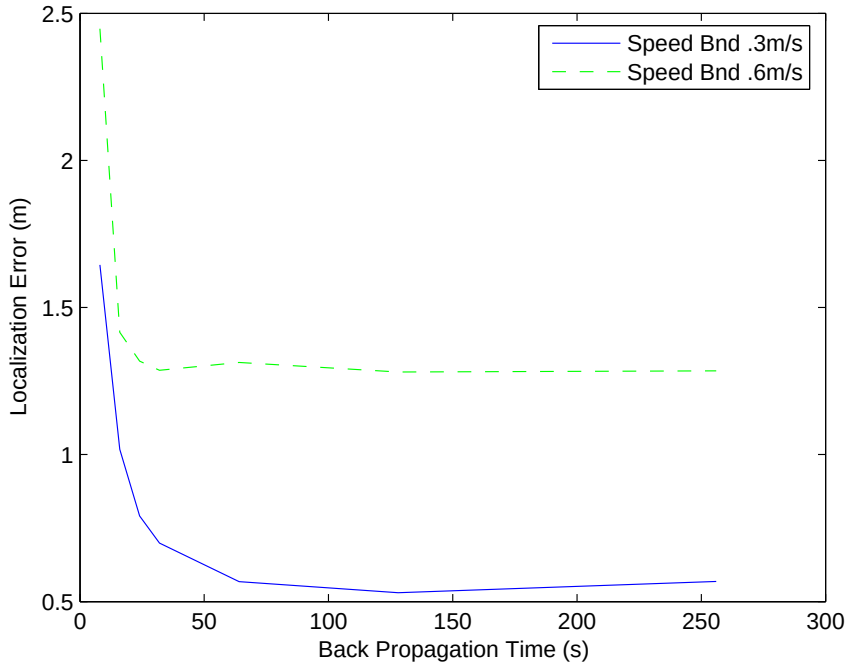


Figure 4-6: Effect of the back propagation time on localization error with two different speed bounds. This data shows that increasing the back propagation time to more than 75 seconds has little benefit.

The results shown in Figure 4-6 show that the back propagation helps significantly up to the point where the error levels off, however, not more after that point. The results are similar for a mobile node traveling at its maximum speed and for one traveling at half speed except that there is a large constant offset. From this plot it can be determined that a back propagation of 75 seconds is more than sufficient as after this point the localization error does not decrease. For nodes not moving at their maximum speed a back propagation of 25 seconds works well. This also shows

that we can use a constant number of back propagation steps allowing us to achieve constant time updates per each new measurement.

Figure 4-7 also shows a comparison of using back propagation (solid line) and using no back propagation (dotted line) while varying the upper bound on the speed. This shows that the error is more than 2 meters higher when not using the information gained from back propagation.

4.4.2 Speed Bound

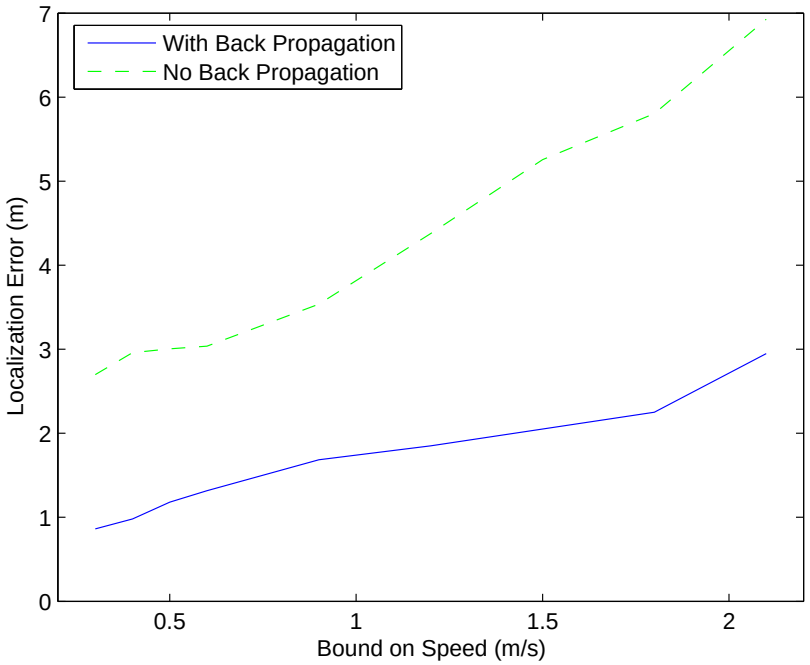


Figure 4-7: The effect of the upper bound on the speed of a mobile node moving at .3m/s using back propagation (solid) and not (dotted). The error increases fairly linearly, but when no back propagation is used there is a large initial offset.

Figure 4-7 shows how changing the upper bound of the mobile node speed effects the localization. Two plots are shown, one where back propagation is used and one where it is not. For both the real speed was .3m/s and the upper bound on the speed was adjusted. A range measurement was taken every 8 seconds to a random node of the 20 static nodes. The back propagation time used was 25 seconds.

As the upper bound on the speed increases, the error also increases. From the data collected this appears to be a fairly linear relationship implying that the errors will not be too bad even if the mobile node is not traveling at its maximum speed. The back propagation, however, is critical. With an upper bound on speed seven times larger than the actual speed we still achieve results comparable to that of the algorithm which does not use back propagation but knows the exact speed of the mobile node.

4.4.3 Number Static Nodes

Figure 4-8 shows how the number of static nodes effects the localization error. For this experiment the mobile node was moving at its upper bound speed of .3 m/s. A range measurement was taken every 8 seconds and the measurements were propagated back 80 seconds. In this experiment any regions which contained disjoint arcs were not used in computing the error. This means that the error was actually much worse when there were few nodes than is reported in the figure.

The main configuration that leads to large errors is the singular configuration where the nodes were nearly collinear. Additionally, if ranges were only obtained from a single pair of nodes (which happens with higher probability with few nodes), then there would be an ambiguity which could lead to a large error. To try and avoid this situation a large back propagation time of 80 seconds was used. This was chosen to be larger than the maximum back propagation we found useful in Section 4.4.1. As can be seen in the figure six or more static nodes produced good results. Thus, unless the static nodes can be carefully deployed, it is important to have at least six static nodes.

4.4.4 Ranging Frequencies

Figure 4-9 shows the effect of changing the frequency of ranging. Experiments were conducted in which the ranging frequency was varied between ranging every one second to ranging every ten seconds. Each time a range was taken a random node

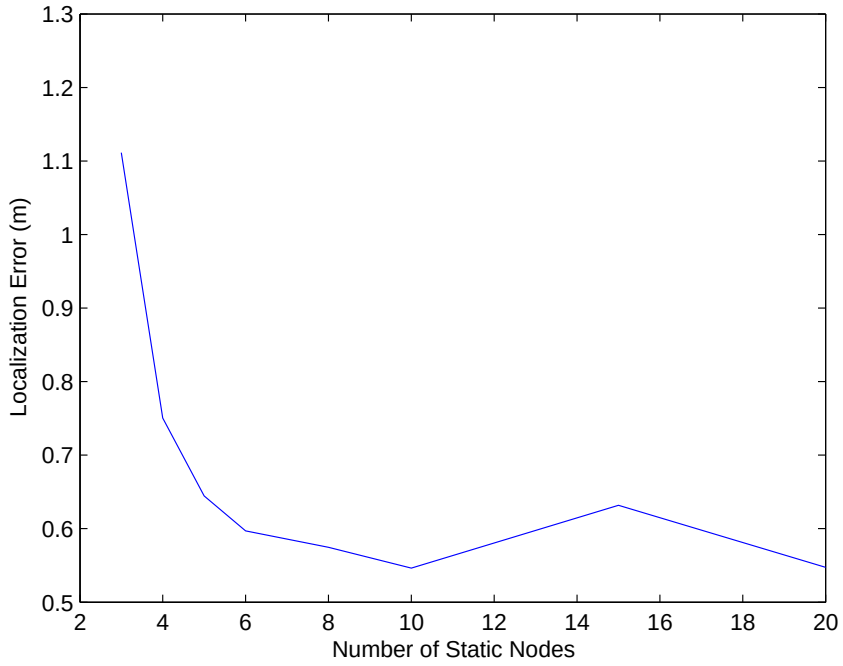


Figure 4-8: Adjustment of the number of randomly placed static nodes. Having fewer than six leads to larger error.

to range to was selected. A speed of .3m/s and a bound of .6m/s were used for the mobile node in this experiment. There were 20 static nodes and the measurements were propagated back to the four previous measurements.

The figure shows that the relationship is very linear. As the ranging frequency increases, the error decreases. Thus, it is always desirable to get as high a rate of measurements as possible.

4.4.5 Gaussian Error

Figure 4-10 shows the effect of Gaussian error in the measurements on the localization. The mobile node had a speed of .3m/s and an upper bound speed of .6m/s. It was moving through a field of 20 static nodes with range measurements taken every 8 seconds. The measurements were propagated back over 30 seconds.

Even with the error in the measurements the algorithm performs well. The localization error scales linearly with the measurement error. With three meter average

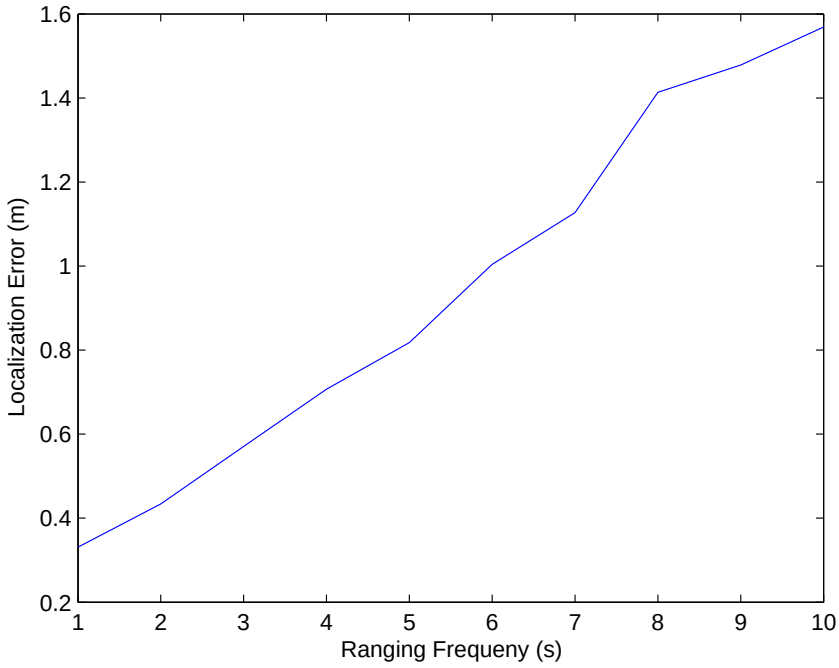


Figure 4-9: Experiment showing the effect of changing the frequency of the ranging. The relationship is linear and indicates that decreasing the ranging frequency is always preferable.

measurement error, the localization error is four meters. As the error is one meter when there is no Gaussian error we can say that the error, Err, will be approximately

$$Err = Err_G + 1 \text{ meters},$$

where Err_G is the Gaussian error in the measurement.

4.4.6 Post Filters

Figure 4-11 shows the results of using the three different post filters described in Section 4.3. The mobile node had a speed of .3m/s and the upper bound on the speed was varied. The mobile node was moving through a field of 20 static nodes and ranges were taken every 8 seconds. The measurements were propagated back over 30 seconds. The filters were tuned by hand to produce good results when the upper

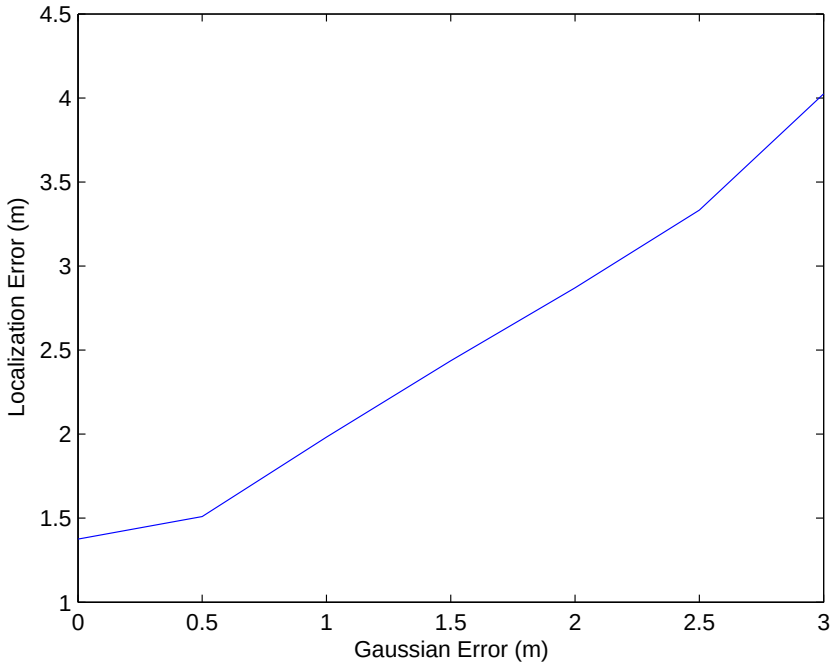


Figure 4-10: The effect of Gaussian noise in the measurements. The localization error is approximately one plus the Gaussian error over this range.

bound on the speed was 1.2m/s.

The figure shows that when the upper bound equals the real speed the raw recovered path has the lowest error, while the EKF produces the worst results. The error produced by the EKF, however, tends to remain fairly constant throughout as indicated by the near horizontal line. The regular Kalman filter performs a little bit better than the raw data, but not significantly. The EKF outperforms the regular Kalman filter due to the fact that it is able to take into account the non-linearities of the errors. The regions found are arcs, which are directly inputed into the EKF. The Kalman filter must make use of the point produced by the raw path filter with some linear variance which does not properly represent the information from the algorithm.

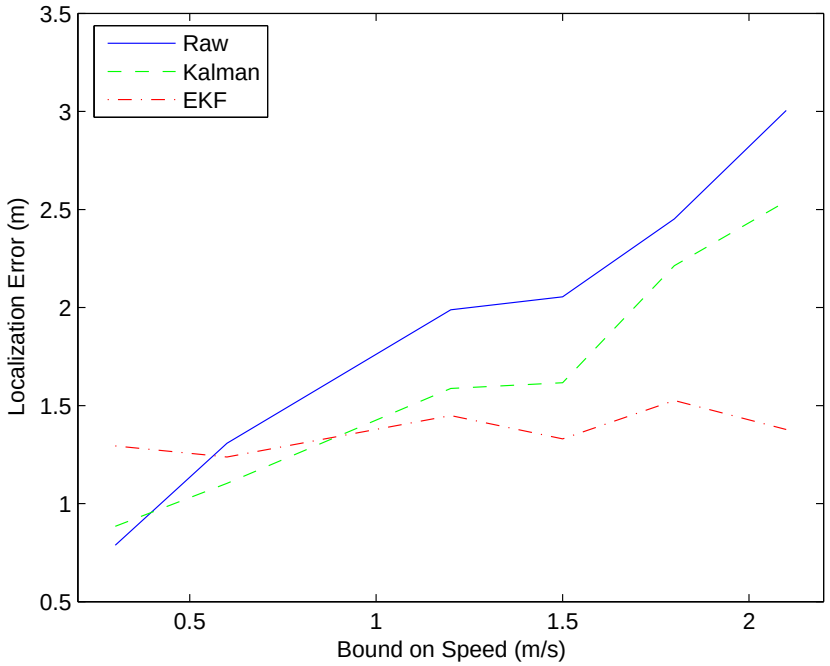


Figure 4-11: The errors of different post filters when changing the upper bound of the mobile node speed. The mobile node traveled at .3m/s. The Kalman filter is typically better than the raw data. The EKF performs best when the upper bound on the speed is more than twice the real speed

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 5

Hardware Experiments

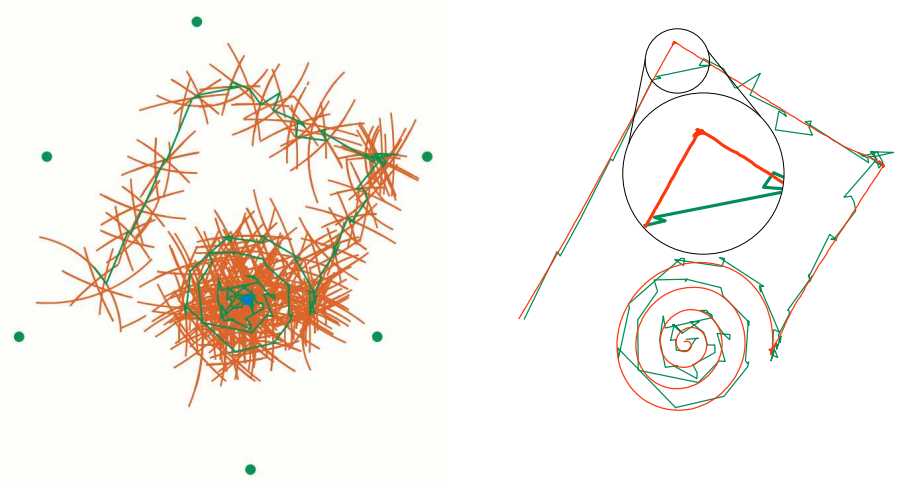
In addition to simulations, we have also performed numerous real-world experiments. We started using land-based sensors capable of obtaining inter-node range measurements. We then implemented the system on the AMOUR underwater system [27]. These experiments verify our theoretical results. In this chapter we will start by presenting the results of the experiments performed indoor and then present the results of the underwater experiments.

5.1 Indoor Experiments

We implemented the range-only version of Algorithm 1 and tested it on a dataset created by Moore *et al.* [19]. Range measurements were obtained from a static network of six MIT Crickets [22] to one mounted on a mobile robot. The robot moved within a 2.0 by 1.5 meter space. Ground truth was collected by a video capture system with sub-centimeter accuracy [19].

Figure 5-1(a) shows the static nodes (green), the arcs recovered by the localization algorithm (orange), and a path generated by connecting the mid-points of the arcs (green). Figure 5-1(b) shows the same recovered trajectory (green) along with ground truth data (red). Inset in Figure 5-1(b) is an enlarged view showing a location where an error occurred due to a lack of measurements while the robot was making a turn.

The mean absolute error from the ground truth was 7.0cm with a maximum of



(a) (b)

Figure 5-1: (a)The arcs found by the range-only implementation of Algorithm 1 and the path reconstructed from these. Orange arcs represent the raw regions, while the green line connects the midpoints of these arcs. (b)Ground truth (red) and recovered path (green). Inset is an enlarged portion illustrating error caused by a lack of measurements during a turn of the robot.

15.6cm. This was computed by taking the absolute error from the ground truth data at each point in the recovered path. This compares well with the 5cm of measurement error inherent to the Crickets [19]. Figure 5-2 shows the absolute error as a function of time.

The algorithm handles the error intrinsic to the sensors well. One reason for this is the robot did not always travel at its maximum speed in a straight line. This meant the regions were larger than if the robot were traveling at its maximum speed in a straight line, making up for measurement error. Most sensors also have occasional large non-Gaussian errors. To account for these, measurements which have no intersection with the current localization region are rejected as outliers.

This implementation only propagates information back to 5-8 localization regions. We did not find it sensitive to the exact number. We also never encountered a region with a complexity higher than three. These two facts gives an overall runtime of $O(n)$, or $O(1)$ for each new range measurement.

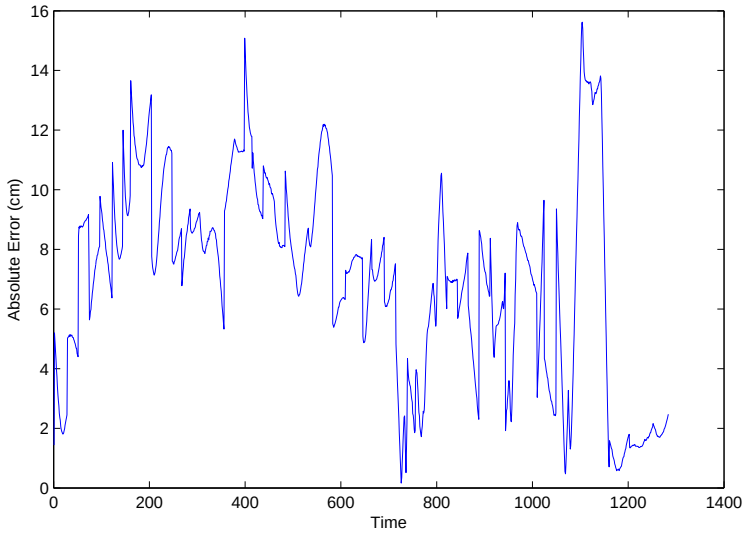


Figure 5-2: Absolute error in the recovered position estimates in centimeters as a function of time.

5.2 Underwater Experiments

During June 2006 we traveled to the University of California Berkeley Richard B. Gump Biological Research Station to run field experiments with our collaborators from CSIRO lead by Dr. Peter Corke. The Gump station is located in the South Pacific on the island of Moorea, French Polynesia. The goal of this trip was to collaborate with the marine biologists at the station to explore ways in which our underwater system of robots and sensors could aid in studying the coral reef environment. Studying and monitoring the coral reef ecology is critical to our understanding of global warming and how it can be prevented.

Our system consists of a number of statically placed sensor nodes which log a variety of data about the conditions at their locations over large periods of time. This data must be periodically recovered. As acoustic communications is slow (typically less than $300 bits$ per second), in our system we use our underwater robot to travel to each of the sensor nodes and directly download the data. To enable this the robot must be precisely localized within the field of sensor nodes. Ranges are obtained to the static nodes as the robot moves through the water. Using this information we

are able to apply the localization algorithm we developed to accurately localize the robot.

In this chapter we will discuss the system in detail and present the results of the localization algorithm. The data from the final experiment was collected on a robot from a CSIRO team that we collaborated with at the station [9].

Sensor Nodes

The sensor nodes, which were developed in our lab, are shown in Figure 5-3. They are deployed statically in the ocean via a small anchor. Each contains a temperature and pressure sensors and a camera. Additionally, they have inputs for a variety of other sensors such as Ph, conductivity, etc. For communications the sensors have both acoustic and optical modems. Radio modems are not included as this does not work well underwater.

Figure 5-3: A picture of some sensor nodes drying.

The sensor system was built in such a way that the system is very easy to deploy. The nodes can just be placed in the water and they will automatically localize themselves using ranges obtained from the acoustic modems. Before the nodes can localize themselves they must decide on a communication schedule. To do this we

use a self-synchronizing time division multiple access (TDMA) scheme to schedule messages.

The static localization algorithm is a robust localization algorithm we developed based on work by Moore et al. [19]. This takes as input the set of ranges between all of the sensor nodes.

The ranges are obtained using the acoustic modems, also developed in-house, which have a range of over 300 meters and a bandwidth of 300 *bits* per second. The ranges are obtained using two different methods. The first is to measure the round trip time of a message between a pair of nodes. This give us a ranges with an accuracy of ± 3 cm.

The second method is to synchronize the clocks on the nodes and then use a schedule to determine when each node should send a message. The modems have temperature compensated oscillators with less than one part per million drift which allows sub-meter ranging accuracy for about thirty minutes before the clocks need to be synchronized again.

In both of these cases we are only able to obtain a single range measurement at any one time. In the case where the ranging is taking place to the robot, this implies that between each range measurement the robot will have moved. We are currently able to obtain a single range measurement every one to four seconds.

Since the acoustic modems are very low data rate we have also developed an optical modem to be a short-range high-bandwidth communications system. The optical modem is capable of transmitting at 200 Kb/s. In order to work the optical modems must be within 2 meters of each other in a cone of about 90 degrees. This system is used between the robot and sensor boxes to download the data from the sensors.

Underwater Robot

The robot we have developed in our lab is called the Autonomous Modular Optical Underwater Robot (AMOUR), shown in Figure 5-4. It is 11kg. with a maximum speed of 1m/s. It has a battery life of 5 hours.

Figure 5-4: A picture of AMOUR and some sensor nodes.

The robot has all of the capabilities of the sensor boxes as well as a more advanced camera system for use in local obstacle avoidance. One of the design goals of the robot was to be inexpensive, so it does not have an expensive inertial measurement unit (IMU). Instead we rely on the range measurements we obtain to the sensor nodes to determine its trajectory through the water

The job of the robot is to travel around and download data from the senor nodes. Additionally it gives the network dynamic sampling capabilities. If an event is happening of interest the robot can move to that area to provide denser sensor sampling. We envision having many robots in the final system to provide highly dynamic sampling and faster download of the data from the static nodes.

To be able to find the sensor nodes the robot must know precisely where it is at all times. Additionally, it is important to have an accurate record of where it has been to give meaning to the sensor readings it takes as it moves through the water. This makes our localization algorithm we have developed in this thesis highly applicable to this system where we only have periodic range measurements and an upper bound on the speed of the robot.

Localization Software

The localization software on the robot is a `C++` implementation of the localization algorithm. The structure is very similar to that of the simulator described in Chapter 4. Unfortunately, it was not feasible to use the same Java libraries developed for the simulator as the central processor on the robot is an Analog Devices Blackfin processor which does not have a Java compiler. Additionally, the simulator has some overhead in the localization code associated with the visualization which we wished to avoid when running on the robot.

The other main difference that the robot software is a 2.5D implementation of the algorithm. While the robot moves in three dimensions in the water, all of the sensor boxes as well as the robot have pressure sensors which provide accurate depth readings. This constrains the problem to two dimensions. However, the equations used in the computations are still the 3D versions, just with one of the parameters known. Alternatively, it can be solved by projecting it into 2D and solving it there and then recovering the third dimension at the end. This is why the underwater problem is commonly known as a 2.5D problem.

At this point we have not fully integrated the localization algorithm into the navigation routines of the robot. We can only recover the course, we cannot use it to actively navigate the robot. This is merely due to a lack of time. We plan on fully integrating the system in the near future.

5.2.1 Experimental Results

We will now present the results of one of the experiments performed in Moorea.

Experiment Setup

For this experiment we had a statically deployed network of six sensor boxes with the configuration shown in Figure 5-5. These were manually placed in the water at different locations over an area of approximately 60 by 60 meters. The sensor nodes automatically localized themselves using our static node localization algorithm.

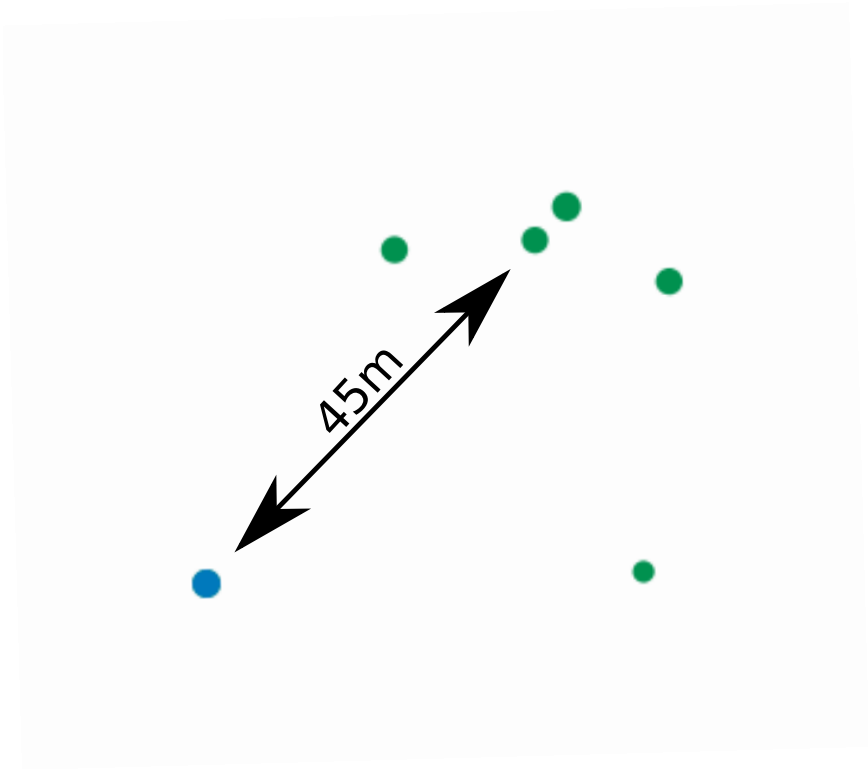


Figure 5-5: The self-localized static sensor nodes.

We used the CSIRO robot equipped with a GPS to provide us with ground-truth data. We attached to this one of our sensor nodes to collect the ranging information. For this experiment we did not run the localization algorithm online to control the robot as we did not have time to do the necessary software integration. Instead we analyzed the data offline and compared it to the GPS path.

Results

Figure 5-6 shows the raw ranges that were received from the nodes as the mobile node moved through the water. The result of running the localization algorithm on this data is shown in Figure 5-7. The orange arcs shown indicate the localization regions. The green line is a path reconstructed using the centers of the regions as an estimate of the location at that time.

This can be compared to the GPS ground truth data as shown in Figure 5-8. This shows that the algorithm performed well. The mean error was .6 meters and

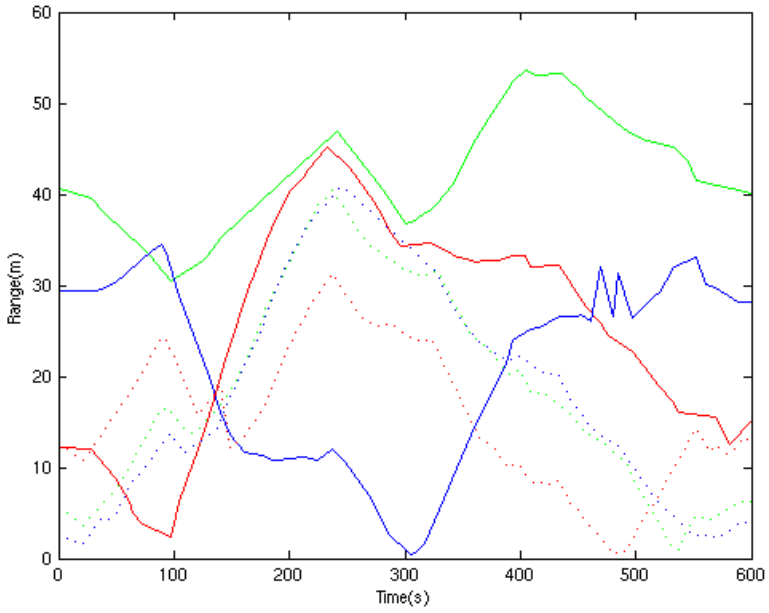


Figure 5-6: The raw ranges received from the sensor nodes.

the maximum error was 2.75 meters. Note the GPS path is always within the regions found. This means that the algorithm performed as desired and the true location of the robot was always within the computed regions.

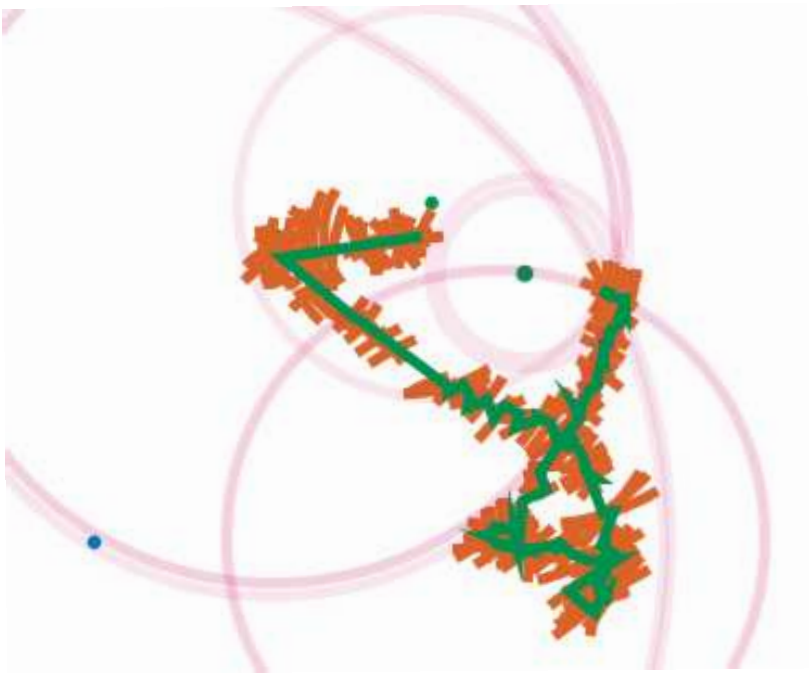


Figure 5-7: The result of the localization algorithm.

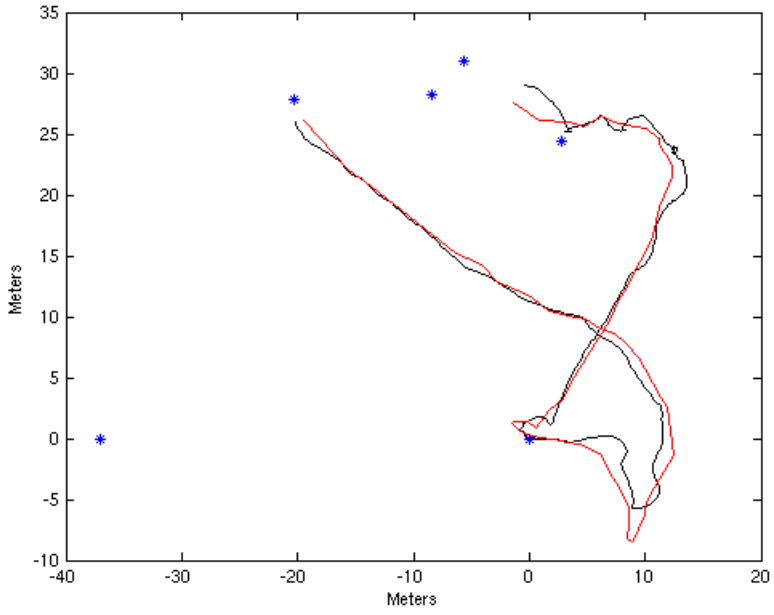


Figure 5-8: The GPS data compared to the recovered path.

Chapter 6

Conclusions and Future Work

In this thesis we presented an algorithm for the localization and tracking of mobile robots when dead-reckoning information is not available. The algorithm is a generalized algorithm which can be applied to any system which uses bounded region models. In particular we examined the range-only and angle-only cases. We developed a simulator and examined the performance of the algorithm under a variety of conditions. We implemented the range-only algorithm on our underwater robot AMOUR and underwater sensor network and experimentally verified our algorithm at the Gump research station in Moorea, French Polynesia, in June 2006.

In this chapter we will conclude by discussing some of the lessons we learned during the course of this research. Additionally, we will identify some future directions of research which we believe may improve the localization algorithm.

6.1 Lessons Learned

In the course of this work we learned a number of lessons. Field applications of robotics are inherently difficult. Especially underwater. Much of our time was spent dealing with the challenges associated with underwater field deployments. Reprogramming our sensor boxes often entailed washing off salt water, drying the boxes, opening them, reprogramming, closing, and finally sealing. Doing this for ten senor boxes would take over an hour for two people. In the next version of the senor box we

plan to implement reprogramming without having to open the sensor box via radio or optical communications.

The good thing about building our system from the ground up is that we are able to make modifications (such as changing the programming method), however, we also spent a lot of time designing and developing the hardware before any of our algorithms could be run in situ. Not only did we have to design the hardware, but we also had to write low-level drivers, networking protocols, and all of the software that ties the systems together. In our sensor boxes we have two microprocessors and one digital single processor all of which must work together and access a variety of peripherals.

In addition to the electronics we had to design the mechanical waterproof housings for the sensor boxes and robot. One trick we found invaluable was the use of a small vacuum pump to vacuum the sensor boxes to a half-atmosphere before we placed them in the water. This allowed us to check the sensor boxes for leaks before we placed them in the water. This also served to hold the lid of the sensor boxes on securely.

We also learned the importance and beauty of minimalist algorithms. Minimalist algorithms often get to the heart of problems without obfuscating the problem with layers of complexity. Additionally, with minimalist algorithms it is possible to prove properties and perform in-depth analyses which would be impossible to perform on more complex algorithms. And as we have shown with this work, the results of a minimalist approach not only work well in theory, but also practice.

An important lesson is the need to test the algorithm in simulation and in the lab before field deployment. As the hardware took much time to develop the simulator was an invaluable tool to test the algorithms. Performing experiments in the simulator, followed by in-door lab experiments allowed us to be confident in the algorithm before deploying it on the actual underwater hardware. The algorithm worked with the first real underwater data that we collected. This is probably something that can be said for very few systems. This is thanks to very good hardware and the extensive tests we performed on our algorithm.

In the field it is also very important to have real-time feedback and control of the entire system. Experiments can take hours and if there is no feedback then all of that work will be lost *when* something fails. For our system we created an underwater laptop case which allows us to interface with our system from a computer while we are in the water with it. This allows us to compensate for any problems we have with the system in situ.

While at the Gump station in Moorea we also learned a lot about the challenges of marine biology. On thing we had not considered previously was the extensive biofouling that occurs to objects underwater. In just a few months objects become completely covered in organic and inorganic materials which are extremely difficult to remove. How to prevent this is an area of active research. Some chemicals can be used, but these are toxic to organisms in the surrounding waters and must be replenished.

6.2 Future Work

The propagation of the information gained from new measurements back to previous measurements can take a significant amount of computation as the number of measurements grows. In the experiments presented in Chapter 5 we found that we only had to propagate the information back through a small, fixed, period of time. This led to a significant reduction in the runtime. There are, however, cases where further propagation may be needed. For instance, the localization of a mobile node traveling in a straight line at maximum speed could be improved by propagating the measurements back far. An adaptive system that decides at runtime how far to propagate information back may improve localization results while maintaining the constant time update.

Throughout we assumed that a maximum speed of the mobile node was known. This requirement can be removed if instead information about the sensor error characteristics is known. For instance, if we expect that measurements should produce regions with a radius of 30cm, then the maximum speed can be tuned online until

regions of this size are produced. The maximum speed estimate can be continuously changed to account for changes in the speed of the robot.

Additional knowledge about the motion characteristics of the mobile node, as used in many of the other localization techniques, can also be added to the system to further refine localization. Maximum acceleration is likely to be known in most physical systems. With a bound on acceleration the regions need only be grown in the current estimate of the direction of travel and directions that could be achieved by the bounded acceleration. In many systems this would significantly reduce the size of the localization regions.

The accuracy of the produced localization regions depends on a number of factors in addition to the properties of the sensors. One of the most important is the time between measurements. The more frequent the measurements the more precise the localization regions will be. Additionally, the selection of the static nodes to be used in measurements is important. For instance, taking consecutive measurements to nodes that are close together will yield poor results. Thus, a selection algorithm, such as that presented by Isler *et al.* [14], may improve results by choosing the best nodes for measurements.

We also plan to perform further real world experiments underwater. The main focus of these experiments will be to improve the underwater robot and sensor system we have developed so that marine biologists will be able to better study the underwater ecology. In these experiments we will further verify our work and test some of the extensions we have discussed. We hope that our system will better the understanding of the complex underwater environment and through this understanding help slow global warming.

Bibliography

[1] Alexander Bahr and John Leonard. Cooperative localization for autonomous underwater vehicles. In *International Symposium on Experimental Robotics (ISER)*, 2006.

[2] K. Briechle and U. D. Hanebeck. Localization of a mobile robot using relative bearing measurements. *Robotics and Automation, IEEE Transactions on*, 20(1):36–44, February 2004.

[3] Zack Butler, Peter Corke, Ron Peterson, and Daniela Rus. From animals to robots: virtual fences for controlling cattle. In *International Symposium on Experimental Robotics*, pages 4429–4436, New Orleans, 2004.

[4] M. Castelnovi, A. Sgorbissa, and R. Zaccaria. Markov-localization through color features comparison. In *Proceedings of the 2004 IEEE International Symposium on Intelligent Control*, pages 437– 442, 2004.

[5] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte carlo localization for mobile robots. In *IEEE International Conference on Robotics and Automation (ICRA99)*, volume 2, pages 1322–1328, May 1999.

[6] Joseph Djugash, Sanjiv Singh, and Peter Ian Corke. Further results with localization and mapping using range from radio. In *International Conference on Field and Service Robotics*, July 2005.

[7] David P. Dobkin, John Hershberger, David G. Kirkpatrick, and Subhash Suri. Computing the intersection-depth of polyhedra. *Algorithmica*, 9(6):518–533, 1993.

[8] Lance Doherty, Kristofer S. J. Pister, and Laurent El Ghaoui. Convex optimization methods for sensor node position estimation. In *INFOCOM*, pages 1655–1663, 2001.

[9] Matthew Dunbabin, Jonathan Roberts, Kane Usher, and Peter Corke. A new robot for environmental monitoring on the great barrier reef. In N. Barnes and D. Austin, editors, *Proceedings of the Australasian Conference on Robotics and Automation*, Canberra, Australia, December 2004.

[10] N. Eagle and A. Pentland. Reality mining: Sensing complex social systems. *Personal and Ubiquitous Computing*, pages 1–14, September 2005.

[11] M. Erdmann. Using backprojections for fine motion planning with uncertainty. *IJRR*, 5:19–45, 1986.

[12] Aram Galstyan, Bhaskar Krishnamachari, Kristina Lerman, and Sundeep Pattem. Distributed online localization in sensor networks using a moving target. In *IPSN'04: Proceedings of the third international symposium on Information processing in sensor networks*, pages 61–70, New York, NY, USA, 2004. ACM Press.

[13] J.-S. Gutmann. Markov-kalman localization for mobile robots. volume 2, pages 601–604, Aug 2002.

[14] V. Isler and R. Bajcsy. The sensor selection problem for bounded uncertainty sensing models. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 151–158, april 2005.

[15] George A Kantor and Sanjiv Singh. Preliminary results in range-only localization and mapping. In *Proceedings of the IEEE Conference on Robotics and Automation (ICRA '02)*, volume 2, pages 1818–1823, May 2002.

[16] Navinda Kottege and Uwe R. Zimmer. Mls-based, distributed bearing, range, and posture estimation for schools of submersibles. In *International Symposium on Experimental Robotics*, 2006.

[17] Derek Kurth. Range-only robot localization and SLAM with radio. Master's thesis, Robotics Institute Carnegie Mellon University, Pittsburgh PA, May 2004.

[18] D. Meizel, O. Leveque, L. Jaulin, and E. Walter. Initial localization by set inversion. *IEEE Transactions on Robotics and Automation*, 18(3):966–971, December 2002.

[19] David Moore, John Leonard, Daniela Rus, and Seth Teller. Robust distributed network localization with noisy range measurements. In *Proc. 2nd ACM SenSys*, pages 50–61, Baltimore, MD, November 2004.

[20] P. Newman and J. Leonard. Pure range-only sub-sea slam. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 1921–1926, Taiwan, 2003.

[21] Edwin Olson, John Leonard, and Seth Teller. Robust range-only beacon localization. In *Proceedings of Autonomous Underwater Vehicles*, pages 66–75, 2004.

[22] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 32–43, New York, NY, USA, 2000. ACM Press.

[23] Adam Smith, Hari Balakrishnan, Michel Goraczko, and Nissanka Priyantha. Tracking moving devices with the cricket location system. In *MobiSys '04: Proceedings of the 2nd international conference on Mobile systems applications and services*, pages 190–202, New York NY USA, 2004. ACM Press.

[24] J. R. Spletzer and C. J. Taylor. A bounded uncertainty approach to multi-robot localization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, volume 2, pages 1258–1265, 2003.

[25] J. Vaganay, J. G. Bellingham, and J. J. Leonard. Comparison of fix computation and filtering for autonomous acoustic navigation, 1998.

[26] J. Vaganay, J.J. Leonard, J.A. Curcio, and J.S. Willcox. Experimental validation of the moving long base-line navigation concept. In *Autonomous Underwater Vehicles*, pages 59–65, 2004.

[27] I. Vasilescu, K. Kotay, D. Rus, M. Dunbabin, and P. Corke. Data collection, storage, and retrieval with an underwater sensor network. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 154–165, New York, NY, USA, 2005. ACM Press.

[28] Greg Welch and Gary Bishop. An introduction to the kalman filter. Technical report, Chapel Hill NC USA, 1995.