6DOF Pose Estimation Using 3D Sensors

Bart Verzijlenberg

Technical Report CSE-2010-08

July 2010

Department of Computer Science and Engineering
4700 Keele Street, Toronto, Ontario M3J 1P3 Canada

6DOF POSE ESTIMATION USING 3D SENSORS

BARTHOLOMEUS J. VERZIJLENBERG

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

MASTER OF SCIENCE

GRADUATE PROGRAM IN COMPUTER SCIENCE AND ENGINEERING
YORK UNIVERSITY
TORONTO, ONTARIO
JULY 2010

6DOF POSE ESTIMATION USING 3D SENSORS

by **Bartholomeus J. Verzijlenberg**

a thesis submitted to the Faculty of Graduate Studies of York University in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE

6DOF POSE ESTIMATION USING 3D SENSORS

by **Bartholomeus J. Verzijlenberg**

By virtue of submitting this document electronically, the author certifies that this is a true electronic equivalent of the copy of the thesis approved by York University for the award of the degree. No alteration of the content has occurred and if there are any minor variations in formatting, they are as a result of the conversion to Adobe Acrobat format (or similar software application).

Examination Committee Members:

1. Michael Jenkin
2. Patrick Dymond
3. Regina Lee
4. John K. Tsotsos

Abstract

Advanced autonomous robotic platforms are increasingly deployed in outdoor environments; these environments can be considered more complex than the traditional office or lab environment, in that they are less structured and allow a wider variety of robot motion. In order for an autonomous system to plan its path through the operating environment, an estimate of its position and orientation is (often) important, if not critical. Pose estimation is the task which deals with estimating the pose, i.e., the position and orientation of a device, based on sensor measurements. Traditional approaches often focus on 2D environments, such as offices, where the device's limited mobility allows for simplification of the required estimation. This work instead focuses on unconstrained six degree of freedom vehicles operating in true 3D environments.The approach developed here decouples the process for estimating a device's orientation from the device's position; this assumes that orientation may be efficiently estimated, provided a sensor such as an IMU. An algorithm is developed that performs orientation estimation efficiently using an extended Kalman filter, while using a particle filter to estimate the position.

Acknowledgements

Some 150 pages, and this may be the hardest one to write. This thesis has been a long time in the making, and it would not have been possible without the help of various people along the way. First of all, I would like to thank my supervisor, Michael Jenkin, for his support and patience with me. He has calmly guided me through the different stages of this thesis, despite various delays and setbacks. His reading, rereading and comments have helped shape this thesis into what it is. Of course, this thesis would not have been finished without the support and encouragement of my parents and siblings. Whether it was talking or care packages, you were there for me. When the writing got to me, there was always some wood, hay or ditch to be worked on; it let me maintain (the pretense of) at least a *little* sanity. I would like to thank Susan in the office for all the help she has given me; anytime I needed help, she was there with a smile. Finally, various people at school have been very helpful, specifically; Anna with various discussions, Andrew Hogue with the egomotion datasets, and Mabel with moral support and encouragement.

And now, at long last, I can acknowledge that this thesis is done.

Table of Contents

Chapter 1 Introduction

> Using sensory information to locate the robot in its environment is the most fundamental problem to providing a mobile robot with autonomous capabilities. Ingemar J. Cox, 1991 [15].

Imagine you are suddenly teleported into the middle of an unknown town as illustrated in Figure 1.1. Checking your pockets, you find a map of the area you are in; it shows some of the local landmarks and streets, but none of them are labeled by name. You want to get to the local bus station, marked on the map, to go home; however, you have no idea as to where you are currently located. You look around in the hope of finding some recognizable landmark. Noticing a phone-booth across the street, you check your map and find that there are 20 phone booths in town. Unfortunately, that is too general to help you. Heading up the street, you walk past a restaurant and, again checking your map, you see that only three restaurants are located near a phone booth. Having narrowed down your possible positions, you again walk a little further, and pass a park. Although there are six parks in town, your map indicates that only one of the three "restaurants near a phone booth" are also located near a park. Having determined where you are on the map, and knowing which way you are facing, you can now determine the easiest way to get to the bus terminal. As you walk along, you occasionally check the map, making sure that the things you see around you match what the map tells you. What you have done in this example is to estimate your pose (i.e., your position and what direction you are facing), and then continued to track your pose as you make your way to the bus terminal.

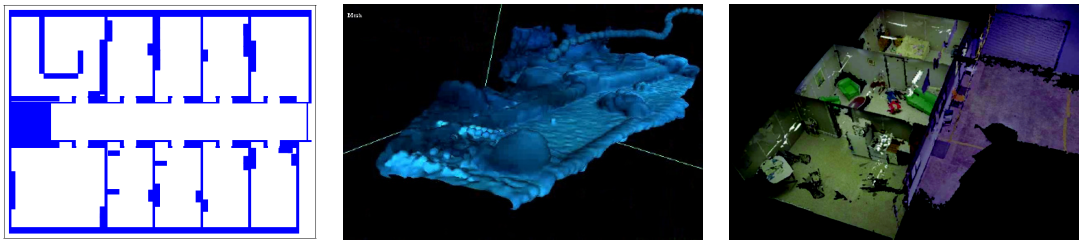
Clearly, it is important for people to know where they are if they are to determine a route to their destination. People perform variants of this simple example everyday to

(a) 2D Map (b) Your initial view after teleporting

Figure 1.1: Estimating your pose based on a map (a) showing the surrounding area of the town, and scenes such as the one in (b), you have learned to automatically find your way.

get their bearings and to plan their routes. Sometimes the process is as simple as looking at the names of two intersecting streets and matching them to a mental map. Other times, the process is more involved; sometimes even requiring us to ask for directions, i.e. to request additional information. In either case, without knowing where you are, it is difficult or perhaps even impossible to plan to get to your destination. This ability to determine where we are is not just important for people, but also for autonomous systems, such as robots. Technologies such as the Global Positioning System (GPS) [55, 54] have been developed in an attempt to directly determine the location of a device. While GPS may solve the position problem in certain cases (i.e., in vehicle navigation systems), such a solution is only available when GPS signals can be received by the system. The GPS signal is generally not available when underwater, underground or even indoors. Active agents cannot always rely on the availability of GPS signals and rather must infer their position through some alternative approach. Even outdoor vehicles may not always be able to access GPS signals, and when they are available may not provide a sufficiently accurate location fix.

How then can an autonomous vehicle localize itself within an environment? The task of determining the position and orientation of a device is called *pose estimation*. Mimicking the approach taken in the example above, given a map of its (extended) surroundings, and allowed enough time to move around and gather information, the vehicle will attempt to identify regions on the map in which it is likely to be located. This may be accomplished if the vehicle is given a map which describes the environment and information regarding past movements and observations. The map may be specific, such as a road map or floor plan, or it may be more abstract like a mental map of features within an area. In considering solutions to the problem of pose estimation it can be



(a) A simple 2D map (b) Partial 3D model of a barge (c) Crime scene model

Figure 1.2: Examples of environment maps. (a) A very simple 2D environment map. Reprinted from [7]. (b) A partial barge model created using the AQUA sensor. Reprinted from [42]. (c) A crime scene model from the C2SM project. Reprinted from [68].

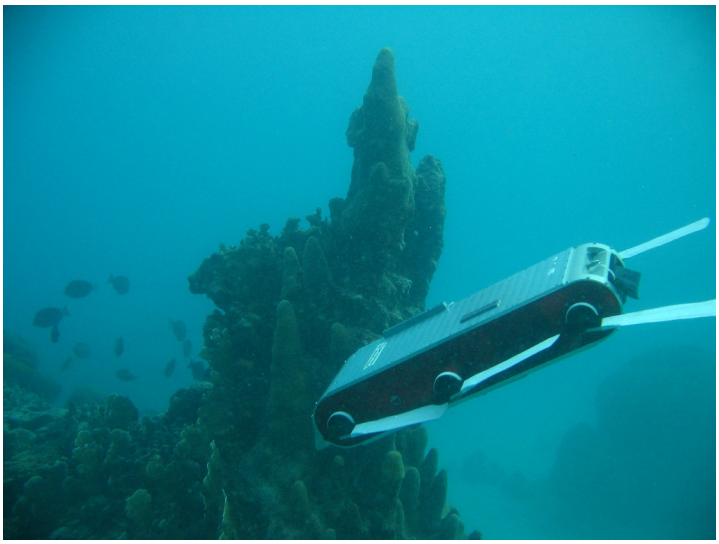


Figure 1.3: The 6DOF amphibious underwater robot AQUA [29]. Pose estimation algorithms for such a device must consider the device's ability to assume nearly any orientation in 3D space.

worthwhile to divide the information available to the vehicle into two groups; *control inputs* and *sensor measurements*. Control inputs are requests to physically move around, while sensor measurements indirectly record the actual result of the control inputs.

Since estimating the position for a stationary robot is relatively trivial (it doesn't move!), the robots considered in this work are all assumed to be equipped with some mechanism allowing a change in position over time. Wheels, legs, fins, wings and various thrusters (jets) are commonly used to accomplish this. For example, the

AQUA robot [29] in Figure 1.3 is a small amphibious hexapod robot able to swim both underwater and at the surface. The AQUA robot is equipped with six fins, allowing it to move in any direction (except sideways) when deployed in water. The operating environment for which the AQUA robot is designed provides a good example where the ability to determine the robot's pose is desirable, but complex. Suppose that the AQUA vehicle is being deployed to inspect a previously surveyed coral reef autonomously. The robot is tasked to return to a specific area of the reef to model the current state of the reef, so that it can be compared to existing models of the same region. Once released near the coral reef, the robot must plan an appropriate route to the particular area of interest. In order to do so, the robot must first determine its current location relative to the reef. It can attempt to accomplish this by using its sensor readings and (possibly) its motion commands in order to estimate where it is, and then refine the estimate over time as it moves toward its destination.

Pose estimation is a well studied problem for autonomous systems (see Chapter 2 for details on various approaches), due to its importance in many tasks. Existing work in pose estimation has typically focused on vehicles that remain in contact with the ground. This limitation of a ground position makes the pose estimation problem one of estimating three unknown parameters which vary as a function of time: the $(x,\ y)$ position coordinates and the orientation θ of the vehicle. The limited number of parameters required to describe the pose simplifies the computational requirements. However, if the vehicle is not restricted to ground movement (as is the case for underwater, aerial and space vehicles), then describing the pose with only three parameters is insufficient. Such vehicles are said to exist within a six degrees of freedom (6DOF) environment, referring to the fact that six parameters are required to uniquely describe the vehicle's pose. These parameters describe the vehicle's ability to move in three dimensions and assume an arbitrary orientation relative to those axis. These additional parameters result in an exponential increase (with respect to the number of dimensions) in computational requirements. For each position estimate $(x,\ y,\ z)$ we must also estimate the three orientation parameters; a lack of knowledge regarding the direction the vehicle is facing may make the evaluation of a range sensor very complicated, possibly preventing its timely computation. The increase in the space that needs to be searched in order to estimate the pose generally makes it impractical to use current 3DOF pose estimation solution techniques to solve the 6DOF problem. This is largely due to the increased time requirement for computing the estimate. If the estimate that is computed by the robot is five minutes out of date, the robot may then make poor choices and, for example, run into objects. This is similar to looking at the cars driving on the road, waiting five minutes and then crossing the street; odds are that an accident will occur.

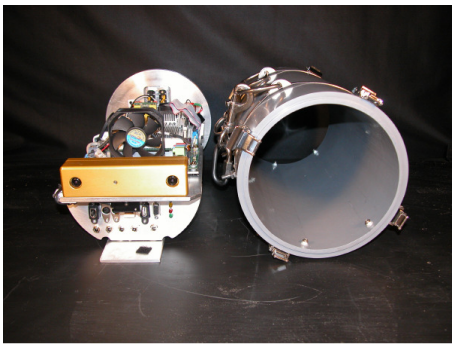
Just as the nature of the vehicle's ability to move influences the pose estimation problem, the nature of the vehicle's sensors also has a great impact on the complexity of the problem. We would have a much harder time moving around a room if we lacked the ability to sense depth visually, as is the case for someone who is blind and instead relies on the limited sensing ability of their cane to help them navigate a room. In the case of ground contact robotics, there has been great success with the use of 1D sensors such as sonar or 2D sensors such as laser scanners (i.e., a touchless 'cane'). See [25, 6, 27, 50] for examples. More sophisticated sensing technologies are beginning to emerge, including the ability to obtain dense 3D depth (distance from the observer) and intensity measurements, such as 3D laser scanners (e.g., [1, 45, 47]) and 3D vision-based systems (e.g., [38, 31]). These sensors determine the location of points in 3D space, and the resulting cloud of points can be used to represent the environment visible to the vehicle. Sensors that can provide more constraints on the vehicle's state can be considered to be more powerful in terms of solving the pose estimation problem. As the dimensionality of the pose estimation problem increases, so does the need for more powerful sensor technologies.

While robots can be equipped with a wide variety of sensors and manipulators, some are more common than others. Common navigation-related sensors include GPS (and similar technologies), various range sensors (laser, sonar, etc.), and orientation sensors such as a compass or an inertial measurement unit (IMU). Sensors can be separated into two groups; some provide an absolute measure of a specific property while others merely indicate the presence of "something." For example, a compass measures directly the earth's magnetic field and as such, with some level of error, always points north. On the other hand, a bump sensor simply establishes that the robot has hit something, without indicating what, or where, it might be (other than touching the robot).

Clearly, it is much simpler if a property can be measured directly, instead of having to infer it. An inertial measurement unit (IMU) typically provides 3D information about a robot's orientation and movement by incorporating accelerometers and gyroscopes. IMU's are able to estimate the orientation component of a device's pose directly, assuming noise in the signal can be filtered sufficiently. While useful for determining the *orientation*, IMUs are generally ineffective for estimating the *position* of a device[1]. For very short time spans, an IMU may be used to *track* movement by integrating the acceleration of a device over time. However, without corrective inputs (such as absolute

[1] Very sophisticated IMU's have been built for maintaining long-range estimates of position (e.g., for submarines, and aircraft prior to the widespread deployment of GPS). Although their accuracy was sufficient to enable transpacific flights with errors under a few hundred miles (and to enable ICBMs), such systems are prohibitively large and expensive to use on autonomous robots.

(a) An Inertial Reference Sensor.



(b) The AQUASensor

Figure 1.4: Examples of advanced 3D sensors. (a) an InteriaCube IMU [39], (b) AQUASensor, an underwater 3D range and intensity sensor. [42]

readings from GPS), the position estimate tends to drift due to noise and error accumulation, and the position result will quickly become useless. Further, even if perfect motion tracking would be possible, the result would still be limited to merely tracking a relative pose. Without prior knowledge of the robot's initial position and orientation, the position estimate would simply be an offset from the start state. Without knowledge of an absolute pose on the map where the device is or was located, it is challenging to plan a route to the device's destination.

1.1 Problem statement

This thesis investigates the efficient global pose estimation of an unconstrained 6DOF device using sophisticated 3D sensors. Unlike a ground contact robot which is typically provided with a "flat" map of its environment, like the one shown in Figure 1.2 (a), the 6DOF devices considered in this thesis are provided with a fully 3D map such as the ones shown in Figure 1.2 (b) and (c). This makes the pose estimation problem considerably more difficult than the 2D pose estimation problem. A variety of sophisticated sensors are available for sensing a robot's environment. This work concentrates on the sensing abilities of devices such as the AQUA [29] and C2SM [68] robot platforms. Although datasets from those projects are used in this thesis, the algorithm developed in this work has wider application to devices operating in 3D environments with alternative sensor technologies. This work assumes a dense point cloud sensor for distance information as well as a sensor for determining the 3D orientation of the device. The AQUASensor [38] uses stereo vision to generate a point cloud representing the distance to objects visible to the sensor, while comparing changes over time for egomotion estimation

(estimating the sensor's motion) [37]. The latter is also important, since many 6DOF vehicles are not in contact with a reference surface, and must work actively to simply remain stationary, resulting in motion without a corresponding command from the robot. The output of an Inertial Measurement Unit (IMU) is used to monitor the approximate orientation of the robot. This task is well studied, and relatively efficient, since the orientation of a device can be sensed directly through gravity and magnetic forces (i.e., a compass). This thesis describes an algorithm that makes use of the strengths of both these technologies, while allowing either to be upgraded independently.

This work makes use of a particle filter approach to estimate the likelihood that the robot is located at a particular pose. Particle filters are an example of a sampling approach to solving the pose estimation problem. In typical 3DOF approaches this involves using many particles ("guesses") to estimate the three pose parameters x, y, θ, as shown in Figure 1.5. Each particle is used to represent a potential solution to the problem and known properties of sensor measurements and commanded robot motion are used to update each of the individual guesses independently. Guesses that score well are retained while guesses that score poorly are discarded. Particle filters are discussed in some detail in the following chapter. Although particle filters have proven to be effective for 2D pose estimation, when considering 6DOF devices, steps must be taken to reduce the search space due to the increased dimensionality of the problem. If a portion of the pose may be estimated more efficiently, then the effort required to estimate the remaining parameters can be reduced. For example, an inertial orientation reference sensor, such as the one in Figure 1.4(a), provides an estimate of the device's absolute orientation. Through the use of an independent orientation sensor it is possible to decouple the orientation and position estimation process. Although affected by external magnetic forces, these orientation sensors are not (for our purposes) affected by the drift errors typically associated with the position estimate provided by IMUs or with the errors associated with odometry. Corrective inputs such as gravity and magnetic forces are constantly incorporated to minimize drift. By modeling the noise typically associated with these sensors, it becomes possible to estimate efficiently the true orientation being measured independently of its position. If this efficient orientation estimate can be incorporated into the pose estimation filter, then computational resources can be focused on the (relatively) simplified position estimation problem.

1.2 Contributions of the work

The primary contribution of this work is the development of a framework to estimate the 6DOF pose of an autonomous agent. This framework allows the orientation of an autonomous agent to be estimated efficiently and separately from the position. Without

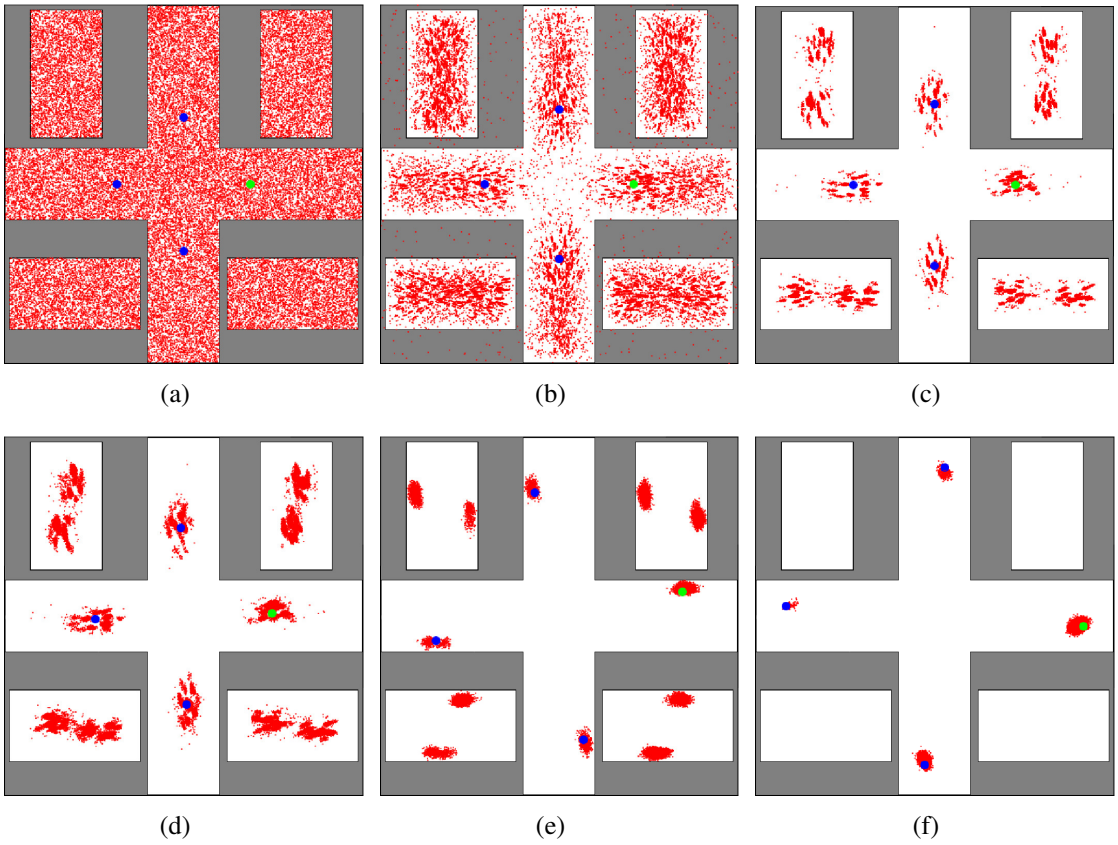


Figure 1.5: 3DOF Pose estimation using a particle filter in an ambiguous environment. The green circle represents the robot's true pose, while the blue circles represent the symmetrically identical poses. (a) Shows a uniform distribution of particles, representing the initial uncertainty regarding the robot's pose. (b-e) Show the progressive clustering of particles as more information is added to the estimate. (f) shows the final estimate, with four symmetric poses. See [50] for problems encountered in symmetric environments such as this.

this framework, the dimensionality of the problem becomes large. This makes real-time 6DOF pose estimation challenging in many situations, or even unrealistic for the existing robots and computational resources available on the AQUA robot. A secondary contribution is the novel use of the Normal Distribution Transform (NDT) [53] for the efficient likelihood estimation of range data. Although ray-casting is a well studied field, high frame-rate video generation remains time-consuming. The NDT on the other hand allows a direct lookup of the (approximated) range likelihood, saving considerable computational time when compared to the ray-casting approach. The combination of these two contributions provides a computationally affordable approach to pose estimation, making it feasible for 6DOF devices.

1.3 Structure of this work

The remaining sections are laid out as follows. Chapter 2 presents a brief history of pose estimation and existing approaches to its solution, along with a description of the use of the Extended Kalman filter for estimating the orientation. Chapter 3 derives the necessary filter for 6DOF pose estimation, and along with examples of its operation, outlines the computation of the various required distributions. Chapter 4 examines the results of various experiments used to validate the filter. Results comparing the filter to a naïve particle filter are also presented. Finally, Chapter 5 provides a summary and discusses potential future work and applications for the filter.

Chapter 2 Previous work

More and more tasks are being assigned to robots, as the required technologies enabling their use develop. Mobile robots are increasingly employed in cutting-edge automated warehouses and deployed to monitor workspaces, such as nuclear facilities, that may be hazardous for people to enter. Operating in restricted or controlled environments, these robots are often able to rely on direct positioning systems, such as patterns embedded in floor tiles [32], GPS systems [55], and other external positioning cues. Being unaware of their location could be disastrous for any of these robot systems. While mistakenly retrieving a book from the wrong shelving unit at an Amazon warehouse would be bad for business, a wrong turn at a nuclear facility might result in disaster. Even the seemingly trivial task of bringing a cup to the kitchen may be next to impossible if the robot is unaware of where the kitchen is in relation to itself. Although wandering around aimlessly is workable for a simple device such as the Roomba vacuum cleaner [40], it clearly would not be effective for more complicated systems and tasks. As the workspaces accessible to robots become more complex, it similarly becomes more critical for a robot to be able to determine its location within the workspace.

Pose estimation is the task of determining the most likely pose of a robot, based on information gathered over time, such as control inputs and sensor readings. There are two broad classes of pose estimation: *local pose estimation* and *global pose estimation* [21]. Local pose estimation (also known as pose tracking) deals with the problem of tracking a robot as it moves, given an approximate initial pose. Since the initial pose of the robot is known approximately, the search process for identifying the robot's location can be focused on a local subset of locations within the known map. Although local pose estimation may not be needed in a perfect world, it becomes necessary when considering the robot's imperfect execution of control inputs. Unintended results like wheel slippage, overshooting a target distance or being bumped by outside influences (i.e., people pushing the robot) cause errors to occur in the robot's estimate of its position. These errors accumulate over time resulting in a highly degraded estimate of the robot's position, as shown in Figure 2.1. Given the importance of this problem a wide range of different algorithmic solutions have been proposed to address this problem.

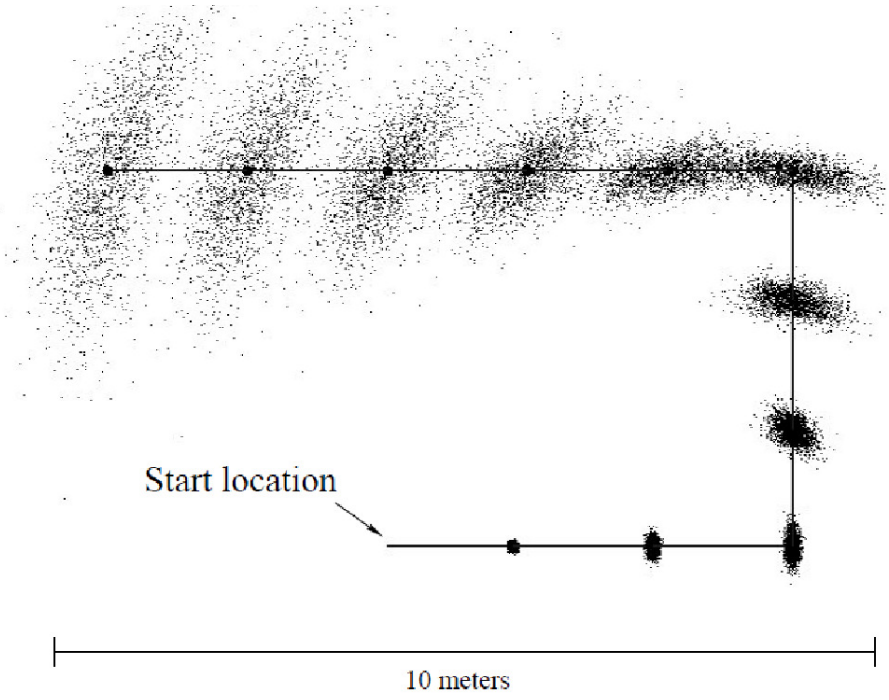


Figure 2.1: A typical expansion of a robot's evolving belief due to motion. The motion of the robot is modeled, including uncertainty, resulting in an increasingly uncertain estimate of the robot pose. Rotation, typically being less reliable than translation, results in an increase in uncertainty after the first turn. After the second turn and some forward motion, the robot's belief has degraded considerably. Outside correction, using sensors, is needed in order to reduce the uncertainty. Figure taken from [7].

Global pose estimation is concerned with locating a robot lacking prior knowledge of its location [41]. Since the initial pose is unknown, there is no good prior estimate upon which to build processes, such as those widely used for local pose estimation. Given a set of sensor measurements from a single location, there are multiple, simultaneous poses which are equally likely or possible. This often prevents the use of methods which are successfully used for local pose estimation. However, once the global pose estimation process converges around a single estimate, the global pose estimation problem essentially simplifies to one of local pose estimation. Given the need to represent multiple possible poses in solving either the local or global pose estimation process, the use of a probabilistic representation to encode potential robot pose(s) is common. Two basic approaches have emerged in the literature, Kalman- and particle-filtering: the (extended) Kalman Filter represents the robot pose as a unimodal distribution while various particle-filtering systems use a sampling approach. See [52] for an approach

that uses Kalman filters, and [28, 58, 14, 5, 19, 27, 52, 63] for a variety of example approaches using particle filters.

This chapter reviews existing work on global pose estimation. Section 2.1 covers theoretical probabilistic estimation. Section 2.2 reviews various approaches that have been used to implement the theory in realistic situations. Section 2.3 provides an overview of Kalman filtering and also outlines the extended Kalman filter used in this work for estimating a vehicle's orientation. Finally, Section 2.4 outlines the Normal Distribution Transform (NDT), which generates a likelihood function defined on a map or range scan (typically used for map building). Components of the NDT are used in Chapter 3.

2.1 Probabilistic estimation in theory

The problem of pose estimation is an inherently probabilistic one, due to the need to model uncertainty and noise in the collected data, as well as error in the actual motion of the robot. In practice it is very difficult to determine with certainty a single pose which satisfies the motion estimates and sensor measurements. Instead, a more effective approach is to generate a probability distribution which represents the likelihood that the robot is located at a certain pose X, given a set of observations and controls Y. This distribution $p(X|Y)$, the probability that X occurs given Y, is called the *posterior probability*; $p(X|Y)$ is also known as the *belief*. Peaks in the belief function correspond to likely pose estimates. Estimating the robot's true state then becomes the task of collapsing the belief into a single estimate, typically by choosing the maximum a posteriori (MAP) value, although other approaches are possible, such as the minimum mean square error (MMSE) (see [60] for a comparison). Unfortunately, there is always a possibility that the belief function has converged to an incorrect pose estimate. The *kidnapped robot problem* [11] deals with detecting when the belief no longer represents the actual state, and correcting it. This failure can occur simply because of accumulated sensor errors, it may be due to only sampling in an incorrect region of the map, or it may be because the robot has literally been picked up and "kidnapped" to a new location.

What is desired, then, is a robust approach which is able to estimate a mobile agent's state based on past observations and commanded motion. Section 2.1.1 discusses an approach that has been developed for general, recursive state estimation. The application of this approach to robot pose estimation is discussed in Section 2.1.2.

$$
\begin{aligned}
bel(x_t) &= p(x_t|z_{1:t}, u_{0:t-1}) \\
&= p(x_t|z_t, z_{1:t-1}, u_{0:t-1}) \\
&\qquad \text{Expand according to Bayes rule:} \\
&= \tfrac{p(z_t|x_t, z_{1:t-1}, u_{0:t-1})\, p(x_t|z_{1:t-1}, u_{0:t-1})}{p(z_t|z_{1:t-1}, u_{0:t-1})} \\
&\qquad \text{Define } \eta \text{ as a normalizing constant} = \tfrac{1}{p(z_t|z_{1:t-1}, u_{0:t-1})} \\
&= \eta\, p(z_t|x_t, z_{1:t-1}, u_{0:t-1})\, p(x_t|z_{1:t-1}, u_{0:t-1}) \\
&\qquad \text{By the Markov assumption:} \\
&= \eta\, p(z_t|x_t)\, p(x_t|z_{1:t-1}, u_{0:t-1}) \\
&\qquad \text{Expand the last term using the theorem of total probability} \\
&= \eta\, p(z_t|x_t) \int p(x_t|x_{t-1}, z_{1:t-1}, u_{0:t-1})\, p(x_{t-1}|z_{1:t-1}, u_{0:t-1}) dx_{t-1} \\
&\qquad \text{By the Markov assumption} \\
&= \eta\, p(z_t|x_t) \int p(x_t|x_{t-1}, u_{t-1})\, p(x_{t-1}|z_{1:t-1}, u_{0:t-1}) dx_{t-1} \\
&\qquad \text{But } p(x_{t-1}|z_{1:t-1}, u_{0:t-1}) = bel(x_{t-1}) \\
&= \eta\, p(z_t|x_t) \int p(x_t|x_{t-1}, u_{t-1})\, bel(x_{t-1}) dx_{t-1}
\end{aligned}
$$

Figure 2.2: Derivation of the recursive belief updating rule, following the derivation found in [7] and [66].

2.1.1 Bayes filter

The Bayes filter [61] provides a general framework with which to recursively estimate the state of a system given a series of measurements. It is used to estimate the posterior probability density over the state space conditioned on (noisy) data and (noisy) commanded motions. The posterior probability density is also called the belief, which reflects the filter's "belief" regarding the state of the parameters being estimated. Bayes filters are used to estimate a belief for some state x_t at time t, given observations $z_{1:t}$ and inputs $u_{0:t-1}$ (where $z_{m:n}$ or $u_{m:n}$ indicate observations between time m and n)[2]. As time goes on, the amount of data $z_{1:t}$ and $u_{0:t-1}$ that must be included in the estimate grows, eventually becoming computationally unattractive. It would be beneficial to eliminate past data by making the filter recursive, and only condition it on the previous state and current sensor data. The data accumulated prior to the last state would not need to be considered, since a recursive filter would depend only on the most recent information.

Following the derivation found in [7] and [66], a recursive version of $bel(x_t)$ may be

[2] We assume that command u_0 is executed first, resulting in state x_1. Sensor measurement z_1 is then observed while in state x_1. Command u_1 is then executed, resulting in state x_2, etc.

calculated. Without any simplifying assumptions (i.e., using only mathematical identities) one can show the following (see Figure 2.2):

$$bel(x_t) = \eta\, p(z_t|x_t, z_{1:t-1}, u_{0:t}) \int p(x_t|x_{t-1}, z_{1:t-1}, u_{0:t-1})\; bel(x_{t-1}) dx_{t-1} \tag{2.1}$$

In this formulation the probability density functions (pdfs) rely on past data; estimating these probabilities in general is complex. In some cases however, it is possible to use realistic assumptions to simplify the parameters for the pdfs. A common assumption is the Markov assumption [7], which in this context states that past and future data are (conditionally) independent if the current state is known. Assuming the state x_{t-1} is complete (that is, no collected data or states prior to x_{t-1} would improve our prediction), we may then simplify the pose pdf as follows, resulting in the *state transition probability*:

$$p(x_t|x_{t-1}, z_{1:t-1}, u_{0:t-1}) = p(x_t|x_{t-1}, u_{t-1}) \tag{2.2}$$

Similarly, we may model the *measurement probability* and simplify it as follows (again assuming x_t is complete and using the Markov assumption):

$$p(z_t|x_t, z_{1:t-1}, u_{0:t-1}) = p(z_t|x_t). \tag{2.3}$$

Using these simplifications, we may substitute equations 2.2 and 2.3 into equation 2.1, resulting in the following (full derivation in Figure 2.2):

$$\begin{aligned} bel(x_t) &= \eta\, p(z_t|x_t, z_{1:t-1}, u_{0:t-1}) \int p(x_t|x_{t-1}, z_{1:t-1}, u_{0:t-1})\; bel(x_{t-1}) dx_{t-1} \\ &= \eta\, p(z_t|x_t) \int p(x_t|x_{t-1}, u_{t-1})\; bel(x_{t-1}) dx_{t-1}. \end{aligned}$$

This form is known as the Bayes filter, and reduces the problem of estimating $bel(x_t)$ to computing three (relatively) basic probability distributions:

$p(x_t\|x_{t-1}, u_{t-1})$	the state transition probability
$p(z_t\|x_t)$	the measurement probability
$bel(x_0)$	the initial belief

Being recursive, the filter must have an initial state $bel(x_0)$ from which to start. Various methods may be used for initializing this, although two are prevalent. The first simply assumes a uniform distribution over the entire space, indicating a total lack of knowledge regarding the true state. The second approach assumes some prior information about the initial state. If the initial pose (in the case of robot pose estimation) is roughly known, then some normal distribution with its mean at the approximately known pose may be

used.

Once initialized, only two distributions remain that must be estimated at each iteration of the filter. The meaning of the state transition and measurement probabilities depend on the context in which Bayes filter is employed. Note that if the measurements are independent of time that $p(z_t|x_t)$ may be simplified to $p(z|x)$. This might permit an *a priori* computation of (a portion of) the PDF, resulting in a reduction of the required computational time for each update.

2.1.2 Markov Localization

The application of Bayes filter, with the Markovian assumptions described above, to the problem of robot localization is known as Markov Localization [7]. See Figure 2.3 for a simple one dimensional example of Markov Localization. The previously defined state x_t being estimated is now defined as the state of the robot (its pose, and any other state-relevant information) at time t. The belief regarding the state x_t may now be estimated using the available data $y_t = \{u_{t-1}, z_t\}$. Let u_t be control data directing the robot's motion at time t and let z_t be sensor measurements observed at time t. Further, let $u_{t_i:t_j}$ and $z_{t_i:t_j}$ denote the control data and sensor measurements collected between time t_i and t_j and assume that control action u_{t-1} is executed prior to sensor measurement z_t. This application of Bayes' filter to robot pose estimation involves the development of appropriate representation formalisms for the pdfs and estimation of the vehicle motion $p(x_t|x_{t-1}, u_{t-1})$ and measurement processes $p(z|x_t)$.

Due to the nature of robot motion, it is extremely difficult to say with certainty what the outcome of any control input might be: even a command to remain stationary may not be successfully executed, since the robot could be bumped or slide down a slope. For this reason the state transition probability makes use of a motion model to estimate the result of control inputs. The motion model is generally dependent on the specifics of the robot platform employed, although certain implementations are robust enough that a general model can be used. Given an estimate of the previous pose x_{t-1} and a control input u_{t-1}, the state transition probability $p(x_t|x_{t-1}, u_{t-1})$ is used to estimate the likelihood of the state x_t. A probabilistic model of the robot's motion is used to compute where, with varying likelihood, the robot will arrive. Since a single control input may result in a variety of outcomes, the state transition introduces uncertainty into the pose estimate of the robot. The uncertainty introduced for this is dependent on the characteristics of the robot being used. For example, a robot may be quite accurate in linear translational motion, but tend to lose accuracy during rotational motion (common in differential drive robots). Figure 2.4 shows the results of various noise parameters for

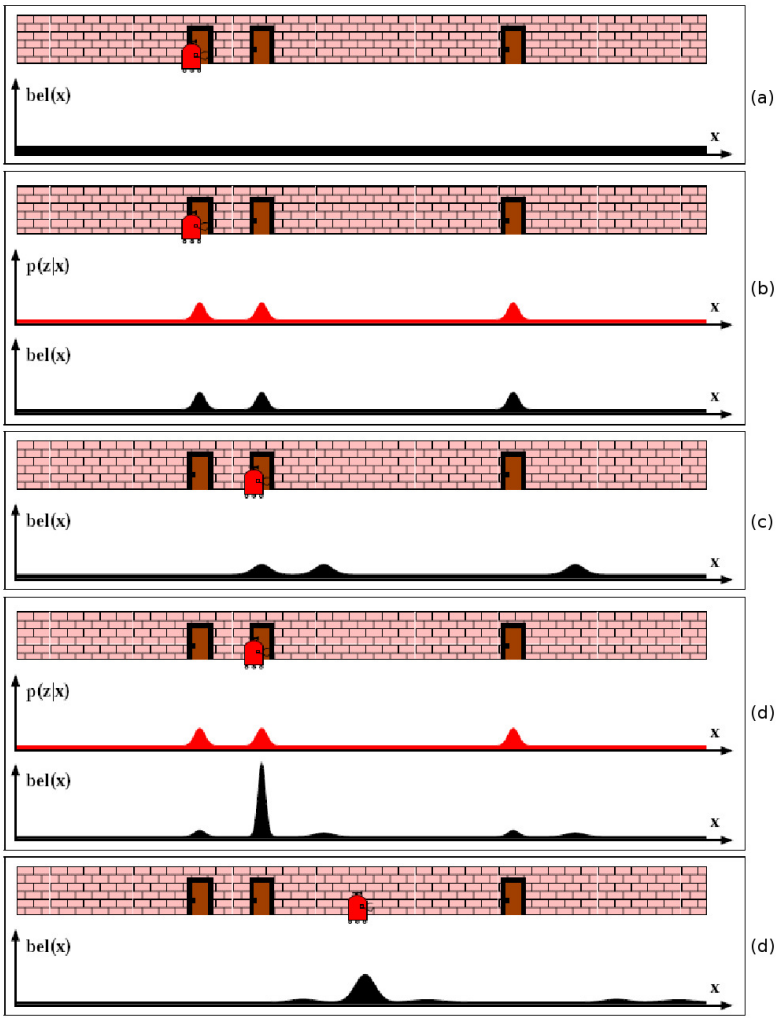


Figure 2.3: The changing belief of a 1DOF robot, as control and sensor data are integrated using Markov localization. The robot's initial pose belief is represented as a uniform distribution (a). At the next time step, the robot detects a door in (b), and updates the belief accordingly. The likelihood of detecting a door is reflected in $p(z|x)$, which represents the probability of observing a door given pose x. (c) Shows the belief after having moved forward some distance. Note that the peaks have smoothed out, due to uncertainty regarding the final motion resulting from the given motion command. In (d) the robot again senses a door, and upon integrating this new observation, the belief now strongly represents the correct pose. After further movement, the belief in (e) is again smoothing out. Figure reprinted from [7].

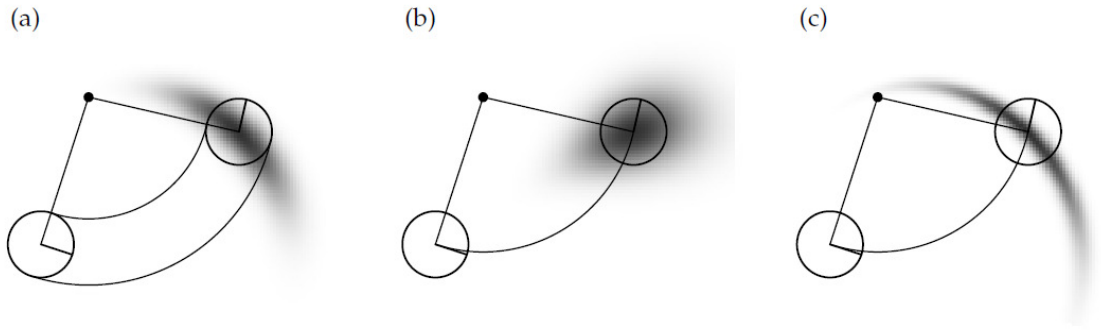


Figure 2.4: The motion model of a differential drive robot. The robot in (a) exhibits accurate forward motion, but considerable error due to rotational slip, typical of such a vehicle. (b) Shows very little rotational slippage, but reduced accuracy for translation. (c) Is similar to (a), but has even greater accuracy for translation, and reduced accuracy for rotation. Figure reprinted from [7].

the motion model of a differential drive vehicle.

The measurement probability $p(z_t|x_t)$ is used to improve the estimate using evidence observed by the robot. Given an observation z_t observed by the robot's sensors, the measurement probability estimates the likelihood of observing such a reading given the pose x_t. This is generally accomplished by comparing the real observation with a virtual one generated using an *a priori* map. As such, the measurement probability is sometimes stated as $p(z_t|x_t, m)$, where m (or m_t, if the map changes over time) is an *a priori* map given to the robot. Various similarity measures are possible, but two commonly used ones are the beam model for range finders [6] and the *a priori* likelihood map [7]. Both are formulated only for a single range data point, even though range *scanners* generally provide denser data using a scanning sensor. It is therefore commonly assumed that the probability for a data set z_t, consisting of i range points, z_t^i, is computed as a combination of the individual points according to $p(z_t|x_t) = \prod_i p(z_t^i|x_t)$.

In [6] a sensor model is developed which computes the measurement probability based only on the expected distance from the map. The probability function suggested is the union of several independent functions and has been shown to work well in practice. Experiments show a decrease in failures from 39.5% and 10.2% to 7.8% and 6.8%, respectively, when compared to other approaches [6]. The probability of observing a certain range is formulated as a mixture:

$$p(z_t|x_t) = \alpha_1\, p_{hit}(z_t|x_t) + \alpha_2\, p_{short}(z_t|x_t) + \alpha_3\, p_{max}(z_t|x_t) + \alpha_4\, p_{rand}(z_t|x_t)$$

$$
\begin{aligned}
\text{sp}_{hit}(z_t|x_t) &= \begin{cases} \eta * N(z_t; z_t^*, \sigma_{hit}^2) & \text{if } 0 \leq z_t \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \\
p_{short}(z_t|x_t) &= \begin{cases} \frac{\lambda_{short} \exp(-\lambda_{short}\ z_t))}{1-\exp(-\lambda_{short}\ z_t^*)} & \text{if } 0 \leq z_t \leq z_t^* \\ 0 & \text{otherwise} \end{cases} \\
p_{max}(z_t|x_t) &= \begin{cases} 1 & \text{if } z_t = z_{max} \\ 0 & \text{otherwise} \end{cases} \\
p_{rand}(z_t|x_t) &= \begin{cases} \frac{1}{z_{max}} & \text{if } 0 \leq z_t \leq z_{max} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

Figure 2.5: Four component PDFs for a range finder's beam model [6].

where z_t is the observed range and z_t^* is the expected range (for details see Figure 2.5 and 2.6). The measurement probability is defined as the composition of four distinct distributions. The first of these, $p_{hit}(z_t|x_t)$, is a Gaussian distribution centered on the range expected to be observed at x_t. This represents the fact that noise is present in the range sensor, preventing an exact, perfect reading. The next distribution, $p_{max}(z_t|x_t)$, represents the fact that it is possible for an object to simply be out of range for the sensor, in which case many sensors return a fixed maximum value. The probability of observing something nearer than expected is modeled by $p_{short}(z_t|x_t)$. This is motivated by the fact that while detecting something between the robot and the expected object is possible, anything passing *behind* the object would be shielded by the object itself and therefore not detected. Finally, $p_{rand}(z_t|x_t)$ handles random, unexpected readings, such as the sensor simply failing to detect an object.

An alternative method makes use of the likelihood field (shown in Figure 2.7) which can be precomputed and then used as a lookup table. This approach represents the likelihood of the robot's range finder detecting an object at some location on the map, determined relative to the robot's current pose. For example, given a pose estimate (x, y, θ), and an object detected at ϕ degrees relative to the robot's pose, at a distance r, then the likelihood field estimates directly the probability of the range finder reporting an object at (x', y'). Define (x', y') as:

$$
\begin{aligned}
x' &= x + cos(\theta + \phi)\ r \\
y' &= y + sin(\theta + \phi)\ r
\end{aligned}
$$

One advantage of this approach is that it does not require the computation of the expected sensor output. Instead, it simply projects the observed data onto the likelihood map. The obvious down side is that, for example, detecting an object at a distance of

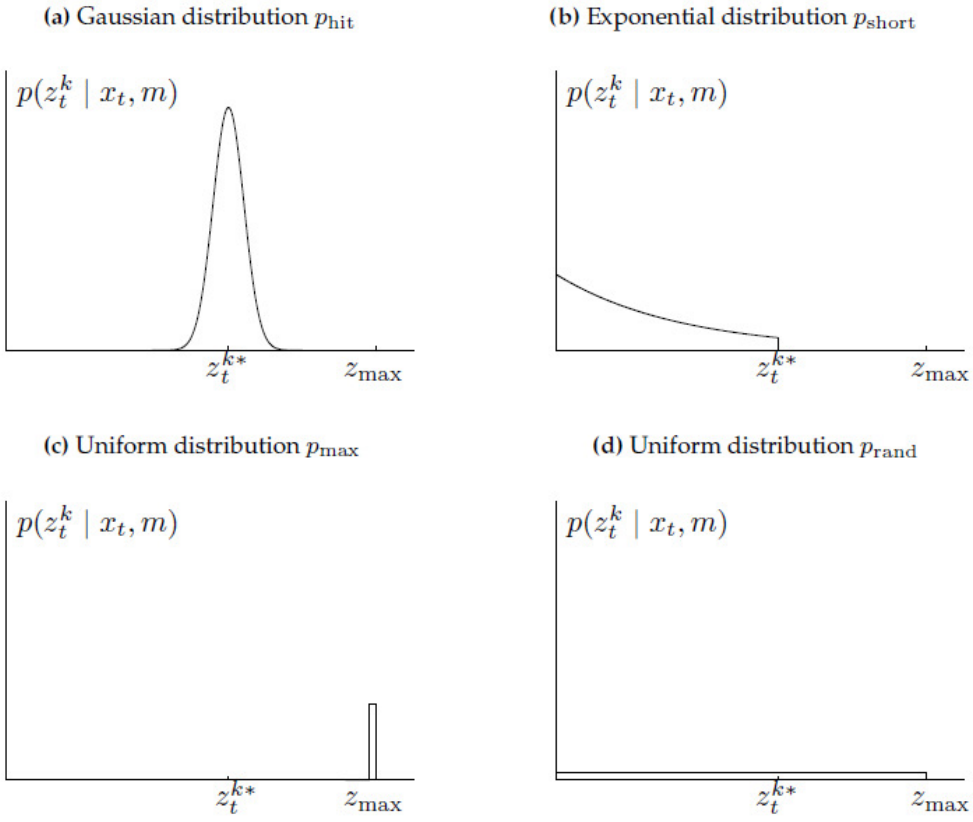
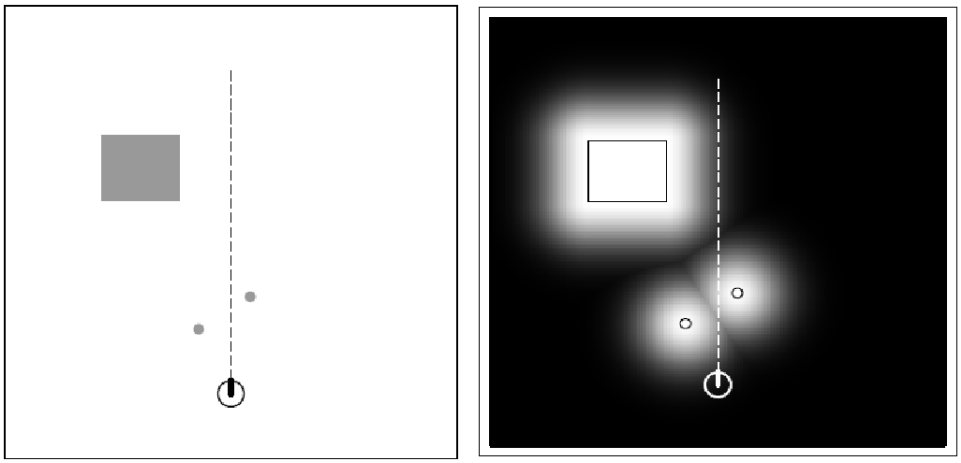


Figure 2.6: A visualization of the components distributions for the beam model. Prior to running the filter, a separate set of distributions is generated for each expected range z_t^*. Reprinted from [6].

2m ignores the possibility that an object *should* have been detected at a distance of 1m according to the map. Although a flaw, it does not appear to be very problematic: trials for this work show that over time such readings tend to naturally become inconsistent with the map, largely negating the problem.

Other methods for defining either the state transition probability or the measurement probability are also possible. In fact, depending on the sensors employed by the robot, entirely different approaches may be required. The robot may depend on vision, for example, or make use of an alternative representation of its environment. In either case, the Bayes' filter may still be applied, and only the specific probability distribution needs to be adjusted.



(a) (b)

Figure 2.7: Example likelihood field (b) for a simple environment (a). Reprinted from [7].

2.2 Practical estimation

Section 2.1 discussed Bayes filters and the specific application of the Bayes filter to pose estimation, called Markov localization. Markov localization may be implemented in a variety of ways, each with its own benefits and drawbacks. A common theme in each of these implementations is that they must all deal with representing the real (continuous) world on a discrete computer. Bayes' filter is likewise defined as a continuous function over the entire state space (the robot's world in Markov localization). Unfortunately, except in special cases such as those exploited for Kalman filters, the nature of the functions required for pose estimation generally prevents the computation of continuous solutions for the belief. In this section we discuss several existing approaches to the problem, beginning with a discrete grid approach.

2.2.1 Discrete grid localization

An approach popular in early implementations (e.g., [6, 16, 5, 34, 25, 62]) of Markov Localization is the discretization of the state space using a grid based representation. By dividing the state space into a grid, a probability can be maintained for each grid cell, where each cell is used to represent a potential pose. Using the simplest approach, each of these cells must be updated every time the robot moves and each time a sensor reading becomes available. Due to the discretized nature of the representation, the cell size limits the accuracy with which the robot can localize itself: there is a clear

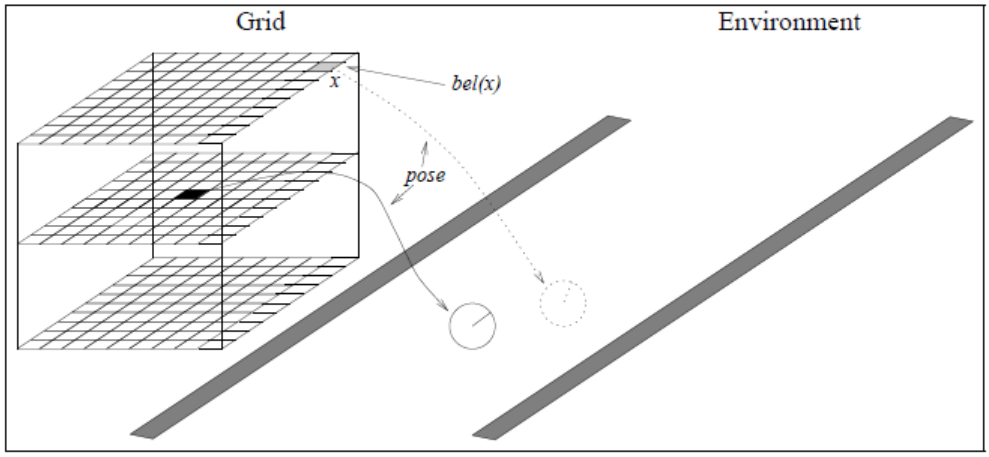


Figure 2.8: A discrete grid representation of a robots state space. Each layer is used to represent one parameter of the state x, y, θ. Reprinted from [7]

trade off between accuracy and computational/space requirements. Increased cell size requires less computation but likewise results in a less accurate estimate. On the other hand, dividing the space into smaller cells is more accurate, but also increases the time required to update the belief.

The basic approach for updating each cell (potential pose) x_i in the grid involves calculating the probability of the robot arriving at x_i from any cell in the space. The first step for each cell x_i is to compute $\int p(x_i|x_{t-1}, u_{t-1})dx_{t-1}$, based on the existing belief and the current motion command. Additionally, after recording a sensor reading, the grid is again updated by calculating the likelihood of observing that sensor reading at each cell. That is, for each x_i, update the probability according to $p(x_i)\ p(z|x_i)$. Since the sensor likelihood is independent of time, [6] shows that we may pre-compute a portion of the sensor likelihood function. Further computational improvements include selectively updating only portions of the grid, which reduces the computational requirements, although memory requirements are unaffected. See [6] for more details.

The Dynamic Markov approach, introduced in [16], attempts to deal with the problem of selecting a cell size and updating the many states. An octree [41] is used to represent the state space (since three parameters are required), and the size of the cells is varied dynamically according to the current belief. The grid is initially divided at a coarse resolution, until peaks develop in the belief, at which time sections of the grid are refined (divided) dynamically. To further improve efficiency, a *selective update strategy* is used to only update likely states, ignoring unlikely

states. A single probability is maintained which indicates the likelihood that the actual pose is not in one of the states being considered. When this probability becomes large, the belief may no longer represent the actual state. In this case, localization must be re-initialized as the belief now contains little useful information other than "I am lost."

Representing and computing the measurement and motion functions works reasonably well for applications such as pose estimation for 2D ground contact robots. Unfortunately, the approach becomes computationally unattractive when moving from 2D to 3D applications. To illustrate, simply representing the state space for a 30mx30m room using a naïve 2D grid representation requires roughly 7Mb of memory (assuming 15x15cm cells and 2° orientation resolution). The same environment in 3D (using the same assumptions, and a 3m ceiling) would require approximately 4.3Tb of memory. Clearly a sparse representation can be used, as in [16], but the sparse representation must still handle the much larger 3D space.

It is obvious then that uniform sampling of the state space is computationally expensive, even when only considering three pose parameters. The probabilities for many cells must be maintained, despite the fact that a majority of those cells represent unlikely poses. For this reason, even for 2D pose estimation, alternative approaches have been developed which focus computation more specifically. Particle filtering is currently the most popular method in the literature and is discussed in the next section.

2.2.2 Monte-Carlo localization

Particle filters have emerged as a very popular approach to the pose estimation problem (e.g., [50, 26, 33]), as they address both memory and computational requirements for 2D robots. Monte-Carlo Localization (MCL) is a *particle filtering* approach applied to robot localization (cf. [19] for particle filters in general, and [58, 7, 67] for the application of particle filters to robotics). Particle filters estimate the desired probability distribution using a sample based approach. When applied to pose estimation, this results in some number of particles being distributed in a map of the environment: each particle (denoted as $p_i = \{x_i, w_i\}$) represents a pose x_i and an associated measure of its correctness w_i, the importance weight. At each time step the particle set is updated (see Figure 2.9) using the control input and sensor data. The update process tends to increase the uncertainty when given a control input (due to uncertainty in the result of motion) and then reduces the uncertainty when integrating the sensor measurement.

As it may be very difficult (or impossible) to sample directly from the desired probability $p(x_t|u_{0:t-1}, z_{1:t})$, an alternative distribution is sampled instead. Importance

input: {control u_{t-1}, sensor measurement z_t, particle set P }

For i = 1 to |P| do
draw a random particle p from P according to $w_i \; \forall \; p_i \in$ P
generate the new pose $p' \sim$ p$(p'|u_{t-1}, p)$
w' = p$(z_t|p')$
P' = {P', { p', w' }}
end for

normalize all $w' \in$ P' such that $\sum_i w'_i = 1$
return P'

Figure 2.9: The basic MCL algorithm. After [24].

sampling [28] works by sampling from this alternative distribution, and then correcting the resulting belief by estimating a weight which reduces the difference between the sampled, alternative belief and the desired belief. As the complexity of importance sampling increases with time, a recursive approach is desirable. Sequential importance sampling (SIS) [10, 18] provides a recursive algorithm to estimate the required weight at time t, given the weight at $t-1$.

A key issue with particle filters is the tendency for most particles' weight to become too small to affect the belief of the robot. As a result, a small set of particles tend to falsely gain high probability while the remaining particles tend toward a negligible probability. To prevent this, a bootstrap filter [61] is used (also known as Sampling Importance Resampling) to address this issue. When the algorithm determines it is required, the bootstrap filter re-samples the particle cloud. Particles are eliminated or retained according to their weight; particles with low probability tend to be removed, and particles with high probability tend to be duplicated. Figure 2.10 outlines the low variance resampler [7].

The number of particles required to localize the robot effectively may be set as a constant at initialization, or adjusted while running. In introducing MCL, [4] decided to adapt the particle size by integrating the control and sensor data in one step. They sample until $\sum_i w_i$ is greater than a preset threshold η, motivated by the fact that particles in highly likely areas will have a large weight, requiring fewer samples. Particles placed in unlikely places have a low weight and contribute relatively little to the sum. On the other hand, a method known as KLD sampling (based on the Kullback-Leibler distance) [23]

input: {Particle set P_t, Particle size M}

$\overline{P_t} = \{\phi\}$ A new particle set
$r =$ rand$(0, M^{-1})$
$c = P_t^1.w$
$i = 1$
For $m = 1$ to M do
  $U = r + (m-1) * M^{-1}$
  while $U > c$
    $i = i + 1$
    $c = c + P_t^i.w$
  end while
  Add P_t^i to $\overline{P_t}$
end for

return $\overline{P_t}$

Figure 2.10: Low variance sampler. From [7]. Notationally P_t^i is the i^{th} particle of the set, and $P_t^i.w$ is the weight associated with that particle.

adapts the size of the particle set such that, with some probability $1-\delta$, the error between the true posterior and the sample-based approximation is less than ϵ. The algorithm works by tracking where particles are placed in the space and then sampling until the particle size n is reached. If a particle is placed into an empty "bin" in the space, then the particle size is recomputed according to the approximation

$$n \simeq \frac{k-1}{2\epsilon}\left\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}z_{1-\delta}\right\}^3 \tag{2.4}$$

where k is the number of non-empty bins in the space and $z_{1-\delta}$ is the upper $1-\delta$ quantile of the normal distribution with zero mean and unity variance. The result is that when the particle set is focused on a small region of the map, fewer particles are placed than when the particle set is distributed widely.

Robots must often operate in environments with people and other dynamic events, and even advanced sensors may still return incorrect data at times. In [8], adjustments to Markov localization are discussed for environments which contain a large amount of dynamic content. They utilize an *entropy filter* to ignore those sensor readings which

increase the entropy of the robot's belief: only sensor readings which confirm the robot's belief are incorporated within the particle filter. In addition, a *novelty filter* is used which eliminates sensor readings for obstacles which are 'novel', i.e. unexpected. If an individual reading, in a scan from the sensor, is shorter than expected with a probability greater than θ (set to 0.99 in [8]'s experiments), then that reading is ignored. Experimental result show standard localization failed in 27% of cases, while the addition of these filters reduces the failure rate to less then 2%. This filter is particularly useful in situations where portions of the sensor scan are frequently corrupted by mobile features, such as people. These temporary influences are clearly not part of the map, and as such their presence in the scan does not contribute positively to the estimated likelihood. Detecting, and removing them is therefore advantageous, as validated by experimental results.

2.2.3 Rao-Blackwellized Particle Filters

Although particle filters are effective when sampling a small number of parameters, even particle filters can become ineffective when the dimensionality of the state space increases. Rao-Blackwellization is an approach that can be used to improve the performance of a particle filter if the structure of the state-space model allows for some of the parameters to be marginalized out, conditioned on the remaining parameters [70]. Rao-Blackwellized particle filters are used to estimate the posterior using a parametric pdf to represent a subset of the parameters, conditioned on the remaining, particle sampled parameters. It is assumed that the parametric distribution can be computed relatively efficiently. Only the remaining parameters need to be sampled (in the reduced state space) which can be accomplished using particle filters [7, 17].

Formally, the desired belief is again represented as $bel(x_t) = p(x_t|z_{1:t}, u_{0:t})$ (see Section 2.1.1). If the state space can be divided as $X = \{P, R\}$, then using the chain rule of probability, we can rewrite $bel(x_t)$ as follows:

$$\begin{aligned} bel(x_t) &= p(x_t|z_{1:t}, u_{0:t}) \\ &= p(r_t, p_t|z_{1:t}, u_{0:t}) \\ &= p(r_t|p_t, z_{1:t}, u_{0:t})p(p_t|z_{1:t}, u_{0:t}) \end{aligned}$$

If we can efficiently compute $p(r_t|p_t, z_{1:t}, u_{0:t})$ analytically (e.g., using a Kalman filter, or some other parametric filter), then only $p(p_t|z_{1:t}, u_{0:t})$ needs to be sampled using a particle filter [28]. The sampling of this reduced space can generally be accomplished using far fewer particles compared to standard particle filters operating on the entire search space. In addition to sampling $p(p_t|z_{1:t}, u_{0:t})$, each particle must now also carry a parametric representation of $p(r_t|p_t, z_{1:t}, u_{0:t})$.

Rao-Blackwellization has been applied very successfully to various robotics areas. For example, FastSLAM [51] makes use of Rao-Blackwellization to reduce significantly the computational requirements of Simultaneous Localization and Mapping (SLAM). The FastSLAM algorithm first computes robot path estimates using a particle filter. Each particle then maintains a collection of landmark estimates computed analytically using Kalman filters, conditioned on the path estimate. Rao-Blackwellization has also been applied to tracking the attitude and position of an air-vehicle. Here the attitude parameters are sampled using particles, while the position is estimated using GPS position fixes and a parametric filter [70].

2.2.4 So where is the robot?

When a definitive solution for the robot's pose is needed, a single pose estimate must be determined from the PDF of the belief. Unfortunately, there is no straight-forward solution to this problem: at any given time the particle filter may represent a multi-modal distribution. In the context of pose estimation it is therefore often meaningless to take a simple mean or the minimum mean square error (MMSE) of the particle filter as a reasonable estimate of the true pose. For example, the estimate resulting from the mean of multiple peaks may be a pose with very low probability [20, 30].

The *maximum a posteriori* (MAP) estimate is better able to deal with the presence of multiple modes. MAP estimation can be used to determine an optimal solution over a time sequence [30], or only for the current state [3]. Simulated annealing or genetic algorithms have been applied to solving the global optimization problem, although they are not easily used for sequential MAP estimation [30]. In [9, 30, 44] the particle representation is considered as a randomized adaptive discrete grid approximation of the target distribution. As the transition and observation probabilities can be evaluated for any two consecutive states, the particle filter approximation can be considered a Hidden Markov Model [30]. The Viterbi algorithm [73] is then applicable to determining a MAP sequence for any time t, although the memory requirements grow with time. If a MAP estimate for only the past k states is sufficient, then the memory requirements are constant, although the computational complexity is high ($O(N^2)$). This cost is reduced to $O(NlogN)$ in [44] through various optimizations, making the approach more feasible.

2.3 Orientation and the Kalman Filter

The Kalman filter [64] can be used to estimate the states of a linear system which has been corrupted by certain types of noise (i.e., gaussian noise). Two key objectives of the Kalman filter are that the expected value reported by the filter should equal the expected

value of the true state and the error variance is minimized. Informally, the Kalman filter is a Bayes' filter within which the probability distributions are assumed to be Gaussian, the Markovian property holds, and plant and measurement processes are linear.

Assume that the system is described by a state x which, in the case considered here, cannot be measured directly; the Kalman filter estimates the state by measuring the system output y. A process model is then required which describes the expected response of the system. Given this process model (or state equation), the state x_{k+1} can then be estimated by the Kalman filter given the state estimate x_k and the control input u_k sent to the system. The effect of the control input on the system is modeled by Γ_k. Both the measurement (Λ_k) and the process models (Φ_k) are assumed to be corrupted by noise (C_v and C_w respectively), which must be uncorrelated and have zero mean. Once the process model generates an expected outcome, the estimate must then be corrected based on the recorded measurement y_{k+1}.

$$\tilde{x}_{k+1} = \Phi_k \vec{x}_k + \Gamma \vec{u}_k \tag{2.5}$$

$$\tilde{P}_{k+1} = \Phi_k \vec{P}_k \Phi_k^t + C_{v,k} \tag{2.6}$$

$$K_{k+1} = \tilde{P}_{k+1} \Lambda_{k+1}^t \left(\Lambda_{k+1} \tilde{P}_{k+1} \Lambda_{k+1}^t + C_{w,k+1} \right)^{-1} \tag{2.7}$$

$$x_{k+1} = \tilde{x}_{k+1} + K_{k+1} \left(y_{k+1} - \Lambda_{k+1} \tilde{x}_{k+1} \right) \tag{2.8}$$

$$P_{k+1} = \tilde{P}_{k+1} - K_{k+1} \Lambda_{k+1} \tilde{P}_{k+1} \tag{2.9}$$

Equations 2.5 and 2.6 estimate the next state without considering the latest measurement. A mixture matrix K (known as the Kalman gain) is computed in Equation 2.7, which weights the measurement and the process model. If the measurement noise is large, K will be small, and little weight will be given to the measurement. A small measurement noise results in K being large and hence more importance will be assigned to the measurement. Finally, Equations 2.8 and 2.9 apply this correction, resulting in the next filtered estimate of the state. There is insufficient space here to go into the full details of Kalman filtering. For more details including a complete derivation of the Kalman filter and its assumptions see [64, 75]. The original Kalman filter paper is [43].

2.3.1 Representing orientation

The estimation of orientation is a well studied problem in the literature [59, 12, 76, 56, 48] and is useful in a wide variety of applications. While orientation is commonly represented using the typical roll, pitch and yaw parameters, they suffer from a draw-back known as gimbal lock. Gimbal lock occurs when two of the axes become parallel, which effectively eliminates one sensing axis. Quaternions

provide an alternative representation which addresses this problem (see [71, 74] for an overview). Here orientations are represented as 4D vectors, removing the problem of gimbal lock and allowing for simple concatenation of successive rotations. Quaternions are a popular representation for orientation in a variety of areas, including robotics and 3D computer graphics.

Since the measurement model for an orientation system is not linear, the Kalman filter cannot be applied to 3D orientations (and quaternions) directly. However, alternative versions of the Kalman filter exist that are applicable to orientation. In the following section the quaternion based, Extended Kalman filter used in this work is described.

2.3.2 EKF for orientation estimation

Although Kalman filtering is excellent for the estimation of linear systems, estimating the orientation of a vehicle is not a linear problem. The *Extended* Kalman filter [64] solves this by extending the plant and measurement models in the Kalman filter. This extension is performed by linearizing the plant and measurement functions around the current estimate of the filter using the partial derivatives of the plant and measurement model. In this way, the problem can be considered linear in the region surrounding the current estimate. In this section we review the filter introduced in [59], making minor modifications to their notation for readability. Although the algorithm in this work treats orientation estimation as a black box, it is useful to gain insight into the working of such a filter. For further details of the EKF and its application to orientation estimation see [59, 48].

To begin with, define the orientation of a coordinate system $\mathcal{B}$ (i.e., the robot), as a rotation relative to a reference coordinate frame $\mathcal{N}$ (i.e., the world). Given a rotation quaternion $\vec{q} = [q_1, q_2, q_3, q_4]^T$, the transformation of a vector $\vec{x}^{3\text{x}1}$ from $\mathcal{N}$ to $\mathcal{B}$ can be defined as

$$\vec{x}^b = C_n^b[\vec{q}]\ \vec{x}^n$$

where

$$C_n^b[\vec{q}] = \frac{1}{\|\vec{q}\|} \begin{bmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\ 2(q_1q_2 - q_3q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2q_3 + q_4q_1) \\ 2(q_1q_3 + q_2q_4) & 2(q_2q_3 - q_4q_1) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{bmatrix} \quad (2.10)$$

and

$$\|\vec{q}\| = \sqrt{q_1^2 + q_2^2 + q_3^2 + q_4^2}.$$

Given the angular velocity $\vec{\omega} = \{p, q, r\}_B$ of a body relative to $\mathcal{N}$, then the angular motion of a rigid body obeys the differential equation

$$\frac{d}{dt}\vec{q} = \Omega[\vec{\omega}]\ \vec{q}$$

where

$$\Omega[\vec{\omega}] = \frac{1}{2}\begin{bmatrix} 0 & -r & q & p \\ r & 0 & -p & q \\ -q & p & 0 & r \\ -p & -q & -r & 0 \end{bmatrix}.$$

Assume that sensor readings are taken at an interval of T_s such that $\vec{\omega}$ is effectively constant between samples, and $\vec{\omega}\ T_s \to 0$, i.e., the overall change in orientation per time step is vanishingly small. Then the small angle approximation can be used, allowing the quaternion prior to sample k to be incrementally updated as follows (from pages 511 and 755 of [74]):

$$\begin{aligned} \vec{q}_{k+1} &= \exp\left(\Omega[\vec{\omega}_k]\ T_s\right)\ \vec{q}_k \\ &= \begin{bmatrix} c_w & r\ s_w & -q\ s_w & p\ s_w \\ -r\ s_w & c_w & p\ s_w & q\ s_w \\ q\ s_w & -p\ s_w & c_w & r\ s_w \\ -p\ s_w & -q\ s_w & -r\ s_w & c_w \end{bmatrix} \vec{q}_k \end{aligned}$$

where

$$s_w = \sin\left(\frac{T_s\sqrt{p^2+q^2+r^2}}{2}\right)$$

and

$$c_w = \cos\left(\frac{T_s\sqrt{p^2+q^2+r^2}}{2}\right).$$

In order to apply a Kalman filter, the processes being estimated need to be modeled so that predictions can be made. The work in [59] assumes that the orientation sensors incorporate a gyro, an accelerometer (i.e. the two typical components of an IMU) and a magnetometer. The output of these sensors varies according to the actual angular velocity $\vec{\omega}_{true}$, the total acceleration experienced by the device (due to gravity $\vec{g}$ and the device's

acceleration $\vec{a}_{body}$) and earth's magnetic field $\vec{h}$. Due to various corrupting influences, these values cannot be measured directly, and instead are modeled as [59]:

$$\begin{aligned}\vec{\omega} &= \vec{\omega}_{true} + \vec{b}_g + \vec{v}_g \\ \vec{a} &= C_n^b(\mathbf{q})(\vec{g} + \vec{a}_{body}) + \vec{b}_a + \vec{v}_a \\ \vec{m} &= C_n^b(\mathbf{q})\vec{h} + \vec{b}_m + \vec{v}_m.\end{aligned} \tag{2.11}$$

Where $\vec{b}_g, \vec{b}_a, \vec{b}_m$ are bias vectors and $\vec{v}_g, \vec{v}_a, \vec{v}_m$ are uncorrelated white Gaussian measurement noise with zero mean and covariance matrices $\Sigma_{g,a,m} = \sigma_{g,a,m}^2 \mathbf{I}^{3x3}$, respectively.

The Kalman filter makes use of two equations to compute expected values which can then be compared to the actual values. The *a priori* equation known as the *state transition equation* predicts what the next state should be, based on the best estimate of the last state, without any current sensor evidence. This filter combines the incremental rotation update from equation 2.11 and the bias vectors:

$$\begin{aligned}\vec{x}_{k+1} &= \Phi(T_s, \vec{\omega}_k) + \mathbf{w}_k \\ &= \begin{bmatrix} exp(\Omega_k T_s) & 0 & 0 \\ 0 & I^{3x3} & 0 \\ 0 & 0 & I^{3x3} \end{bmatrix} \begin{bmatrix} q_k \\ b_k^a \\ b_k^m \end{bmatrix} + \begin{bmatrix} \vec{w}_k^q \\ \vec{w}_k^a \\ \vec{w}_k^m \end{bmatrix}.\end{aligned} \tag{2.12}$$

Once the next state has been predicted, the estimate needs to be corrected using evidence. The *a posterior* estimate can be computed from the *a priori* estimate by comparing the observed and expected data. The *measurement model* is used to generate an estimate of what the sensor measurements *should* be according to the *a priori* estimate.

$$\begin{aligned}\vec{z}_{k+1} &= f(\vec{x}_{k+1}^-) + \vec{v}_{k+1} \\ &= \begin{bmatrix} C_n^b(q_{k+1}) & 0 \\ 0 & C_n^b(q_{k+1}) \end{bmatrix} \begin{bmatrix} \vec{h} \\ \vec{g} \end{bmatrix} + \begin{bmatrix} \vec{b}_{k+1}^a \\ \vec{b}_{k+1}^m \end{bmatrix} + \begin{bmatrix} \vec{v}_{k+1}^a \\ \vec{v}_{k+1}^m \end{bmatrix}\end{aligned} \tag{2.13}$$

Since the measurement model is not linear, as the Kalman filter requires, it needs to be locally linearized for the covariance calculations, by computing its Jacobian around the apriori estimate:

$$F_{k+1} = \frac{\partial}{\partial \vec{x}_{k+1}} \vec{z}_{k+1} \Big|_{\vec{x}_{k+1} = \vec{x}_{k+1}^-} \tag{2.14}$$

As the filter integrates new data readings, a covariance matrix P_k, indicating the certainty of the state estimate x_k, is maintained. In addition, temporary matrices are also created at each step; for the state transition, Q_k, and for the measurement model, R_{k+1}.

$$Q_k = \begin{bmatrix} (T_s/2)^2 \, \Xi_k \, \Sigma_g \, \Xi_k^T & 0 & 0 \\ 0 & \Sigma_a^k & 0 \\ 0 & 0 & \Sigma_m^k \end{bmatrix} \tag{2.15}$$

$$R_{k+1} = \begin{bmatrix} R_{k+1}^a & 0 \\ 0 & R_{k+1}^m \end{bmatrix} \tag{2.16}$$

$$\text{where } \Sigma_{a,m}^k = T_s \sigma_{w_a,w_m}^2 \mathbf{I}^{3x3}$$

$$R_{k+1}^{a,m} = \begin{cases} \sigma_{a,m}^2 \mathbf{I}^{3x3} & \text{conditional on motion, see [59]} \\ \infty & \text{otherwise} \end{cases}$$

$$\Xi_k = \begin{bmatrix} q_4 & -q_3 & q_2 \\ q_3 & q_4 & -q_1 \\ -q_2 & q_1 & q_4 \\ -q_1 & -q_2 & -q_3 \end{bmatrix}$$

Combining the developed state and measurement models, the Extended Kalman filter can now be used to estimate orientation as follows, provided an initial state $\vec{x}_k$ and covariance matrix P_k. Based on $\vec{x}_k$ and the angular velocity $\vec{\omega}$, the next state and error covariance matrix are estimated using the state transition model (*a priori*):

$$\vec{x}_{k+1}^- = \Phi(T_s, \vec{\omega}_k) \, \vec{x}_k \tag{2.17}$$

$$P_{k+1}^- = \Phi(T_s, \vec{\omega}_k) \, P_k \, \Phi(T_s, \vec{\omega}_k)^T + Q_k \tag{2.18}$$

The *Kalman gain* is computed next, functioning as a mixture matrix between the estimated state and covariance, and the measured data: the data known with more certainty is assigned more weight.

$$K_{k+1} = P_{k+1}^- F_{k+1}^T \, (F_{k+1} P_{k+1}^- F_{k+1}^T + R_{k+1})^{-1} \tag{2.19}$$

Using this gain, a correction is applied to the estimate and covariance (*a posteriori*).

$$\vec{x}_{k+1} = \vec{x}_{k+1}^- + K_{k+1}[\vec{z}_{k+1} - f(\vec{x}_{k+1}^-)] \tag{2.20}$$

$$P_{k+1} = P_{k+1}^- - K_{k+1} F_{k+1} P_{k+1}^- \tag{2.21}$$

2.4 The Normal Distribution Transform

The Normal Distribution Transform (NDT) [53] was originally designed for laser scan alignment for ground contact robots performing SLAM. The NDT's feature of interest here is the likelihood estimator that it generates for a given map. This likelihood represents the probability that a range measurement reports a point at any particular location in the map. The NDT is generated by dividing the map, or range scan, composed of a set of points x, into a two dimensional grid of size n, where the first cell is centered on the origin. Associated with each cell are the points $\{\vec{x}_i\}$ that are contained within the cell; a mean and covariance is computed for each cell using this set of points. A piecewise continuous function is thus defined over the entire map, using a normal distribution conditioned on the points in each cell.

More specifically, for the points in each cell, the NDT computes,

$$\text{the mean point } \vec{q} = \frac{1}{n}\sum_i \vec{x}_i \tag{2.22}$$

$$\text{and the covariance } \Sigma = \frac{1}{n}\sum_i (\vec{x}_i - \vec{q})(\vec{x}_i - \vec{q})^t. \tag{2.23}$$

This provides an efficient lookup table for the likelihood of the range finder reporting a data point anywhere in the map. Standard minimization techniques can then be applied to correct alignment errors between the reference scan on which the NDT is based and a newly obtained laser scan. Under the NDT, the likelihood of observing a point $\vec{x}$ in the map, located in cell (u,v) using a grid centered at $(0, 0)$ may be computed as a multinomial normal distribution:

$$p(\vec{x})_{(u,v)} = \frac{1}{2\pi\sqrt{|\Sigma|}} \exp\left(-\frac{(\vec{x}_i - \vec{q})^t \Sigma^{-1} (\vec{x}_i - \vec{q})}{2}\right). \tag{2.24}$$

Since the space has been discretized, this approach leads to discontinuities in the distribution, which [53] compensates for by using four partially overlapping grids, and defining the overall probability as

$$P_{NDT}(\vec{x}) = \frac{1}{4}\left(p(\vec{x})_{(u,v)} + p(\vec{x})_{(u+\frac{n}{2},v)} + p(\vec{x})_{(u,v+\frac{n}{2})} + p(\vec{x})_{(u+\frac{n}{2},v+\frac{n}{2})}\right). \tag{2.25}$$

The normal distribution can easily be defined on higher dimensions and the definition of the NDT may be likewise extended to higher dimensions. Such an extension is outlined in [46] and was shown to be useful for 3D scan registration. The algorithm is changed such that instead of defining a grid using 2D cells, it is defined using a hypergrid of 3D

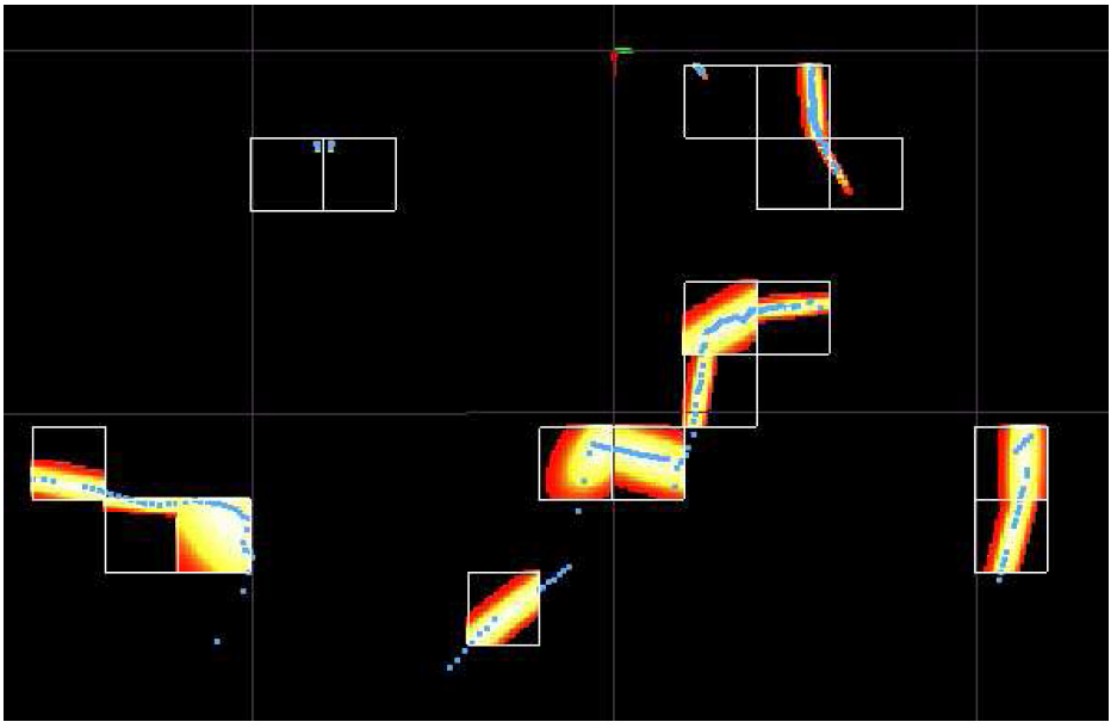


Figure 2.11: A 2D NDT map showing the cell divisions in a simple (sparse) map, indicated using blue dots. Brighter areas indicate a higher likelihood of having the range sensor return a "hit". Reprinted from [46].

cubes. Although the steps for initializing each cube are identical to equations 2.22 and 2.23, the grid is now centered on the 3D origin. The only other difference then, is that the likelihood of observing a point $\vec{x}$ in a given cell is defined as

$$p(\vec{x})_{(0,0,0)} = \frac{1}{(2\pi)^{1.5}\sqrt{|\Sigma|}} \exp\left(-\frac{(\vec{x_i} - \vec{q})^t \Sigma^{-1} (\vec{x_i} - \vec{q})}{2}\right) \tag{2.26}$$

The application of the NDT to pose estimation will be described in Section 3.5.1.

2.5 Summary

It is clear that pose estimation is an important capability for autonomous robots. Current solutions for global pose estimation have been shown to work well for robots with limited freedom of motion. Ground contact robots especially have been well studied, and various robust approaches have been developed for them. However, because of

their computational requirements, these solutions quickly become unattractive when considering vehicles operating with higher degrees of freedom in non-planar environments. On the other hand, the Kalman filter provides a direct, closed form estimate of the state, allowing for fast updates since all the matrices can generally be quickly computed. In comparison with particle filtering, the required time for each update is trivial; unfortunately, Kalman filtering is not normally applicable to pose estimation. The primary obstacle with the Kalman filter is its limitation to representing a unimodal distribution, which is incompatible with the multiple representations needed for position estimation. Although this limits the applicability of the Kalman filter, it finds wide and useful application in domains within which its constraints are met. For example, the EKF can be used to maintain an ongoing estimate of a robot's 3D pose from direct pose measurements. Although neither a pure Kalman filter nor a pure particle filter is appropriate for the problem of pose estimation for 6DOF pose tasks, hybrid approaches may prove effective. The approach taken in current solutions for estimating the range likelihood may be too slow when used in 3D situations. The NDT provides an efficient, alternative likelihood estimator that will be further explored in the following chapter.

In the next section we outline an approach for addressing some of the problems associated with global pose estimation for 6DOF autonomous agents. Specifically, a Rao-Blackwellized approach is developed for 6DOF autonomous agents that uses an Extended Kalman filter for orientation and a particle filter for the other parameters.

Chapter 3 Localization when orientation is known

Vehicles operating in a 3D environment must maintain a host of parameters specifying the vehicle's state. Instead of using a single monolithic system to estimate every component of the state, various sub-systems can be used to monitor individual parameters. This allows an individual sensor to be focused on the task it does well, instead of attempting to estimate all parameters. For example, many airplanes or helicopters have independent devices to monitor a variety of aircraft parameters. A variety of status displays (such as those shown in Figure 3.1) are made available to the pilot. The altitude, airspeed and vertical speed are computed using air pressure and air flow sensors on the aircraft. On the other hand, turn, heading, compass and attitude indicators use a combination of an inclinometer, gyroscope and magnetic compass. Similarly, in pose estimation it makes sense to decompose the various parameters into individual components, based on the available sensors, instead of attempting to estimate everything using a single approach. Taking advantage of hidden structure in the state can make an enormous impact on the computational requirements for estimating the overall state.

In this chapter, we describe an algorithm for the efficient estimation of the pose for a device operating in an unrestricted, three dimensional environment, under the assumption that orientation estimation can be achieved independently of position. Section 3.1 derives the filter, while Section 3.2 outlines the general process for using it. The subsequent sections provide a simple example (Section 3.3), and information regarding the various distributions that need to be computed (Sections 3.4 and 3.5). Finally, Section 3.6 provides an example of the filter in use on a real dataset.

3.1 Derivation of the filter

The purpose of this filter is to estimate a belief regarding the state (position and orientation) of a mobile robot as it varies over time. Let the state of the robot at time t be defined by $X_t = \{x_t, \theta_t\}$, where x_t denotes the *position* and θ_t denotes the *orientation* of the robot. Here the position and orientation are vectors encoding the 3D nature of these values. Further, let Y_t denote relevant evidence collected by the robot at time t, and $Y_{m:n}$

Figure 3.1: Instrument panel on a helicopter showing, among other things gyro, heading and altitude dials. Aircraft pilots rely on these instruments to maintain safe flight, when personal instincts may be unreliable.

the evidence between times m and n. In very general terms, the belief of the robot can then be stated as

$$Bel(X_t) \triangleq p(X_t|Y_{0:t}) \tag{3.1}$$

$$= p(x_t, \theta_t|Y_{0:t}) \tag{3.2}$$

As argued in Chapter 2, it is desirable to split this term into separate subcomponents that might be computed more efficiently than attempting to establish the full pdf directly. We can decompose this monolithic pdf using Rao Blackwellization. A typical application of Rao Blackwellization separating the position and orientation (see Chapter 2.2.3 or [17]) results in either

$$p(x_t, \theta_t|Y_{0:t}) = p(\theta_t|x_t, Y_{0:t})\ p(x_t|Y_{0:t}) \tag{3.3}$$

or

$$= p(x_t|\theta_t, Y_{0:t})\ p(\theta_t|Y_{0:t}) \tag{3.4}$$

depending on the assumption as to which can be established independently, position or orientation. Equation 3.3 would be appropriate if the position likelihood, $p(x_t|Y_{0:t})$, can be decoupled from prior knowledge of the device's orientation state. Although this is not

appropriate for the application considered here, it may be useful for other applications and is an interesting direction for future work. Of interest here is the manipulation given in Equation 3.4. For a device such as the AQUA robot, the orientation estimate in $p(\theta_t|x_t, Y_{0:t})$ is not generally affected by position (ignoring the effects of external noise sources such as local magnetic forces on orientation sensors). This suggests that a potentially effective mechanism for establishing $p(x_t, \theta_t|Y_{0,t})$ is to decouple θ_t and x_t using Equation 3.4. The estimate $p(\theta_t|Y_{0,t})$ can then be used when sampling $p(x_t|\theta_t, Y_{0:t})$. This is the basic approach taken here; the details of the approach are outlined in the following sections.

A recursive filter is desired that computes one portion of the state efficiently while using a sampling approach on the remaining state. Given evidence $Y_{0:t} = \{u_{0:t-1}, z_{1:t}, i_{1:t}\}$, let u_t be the control inputs for the robot at time t, z_t be the sensor readings used as evidence for the robot's position at time t, and let i_t be the IMU data used for orientation estimation at time t. Assume that control u_{t-1} is executed at state X_{t-1}, and that this commanded motion results in state X_t. The sensor data z_t and i_t are subsequently collected while the device is in state X_t. Note that neither z_0 nor i_0 are collected, since it is assumed that u_0 will be executed prior to the first data collection; similarly, there is no u_t.

Given some independent mechanism of estimating orientation, how can we reformulate the conditional pdf, $p(X_t|Y_{0:t})$, to exploit this? Consider the $Bel(X_t)$: after applying the theorem of total probability over θ_{t-1}, the belief (or *joint posterior distribution*) can be rewritten as

$$
\begin{aligned}
Bel(X_t) &\triangleq p(X_t|Y_{0:t}) \\
&= p(x_t, \theta_t|Y_{0:t}) && (3.5) \\
&= \frac{p(x_t, \theta_t, Y_{0:t})}{p(Y_{0:t})}. && (3.6)
\end{aligned}
$$

Applying the theorem of total probability over θ_{t-1} on the numerator

$$
= \frac{\int p(x_t, \theta_t, Y_{0:t}|\theta_{t-1})p(\theta_{t-1})\, d\theta_{t-1}}{p(Y_{0:t})} \qquad (3.7)
$$

and applying the chain rule of conditional probability $(p(a, b|c) = p(a|b, c)p(b|c))$

$$
= \frac{\int p(x_t, \theta_t|Y_{0:t}, \theta_{t-1})p(Y_{0:t}|\theta_{t-1})p(\theta_{t-1})\, d\theta_{t-1}}{p(Y_{0:t})}. \qquad (3.8)
$$

Observing that $p(a|b) = \frac{p(b|a)p(a)}{p(b)}$ and pulling the denominator into the integral,

$$
= \int p(x_t, \theta_t|\theta_{t-1}, Y_{0:t})\; p(\theta_{t-1}|Y_{0:t})\; d\theta_{t-1}. \qquad (3.9)
$$

Equation 3.9 may now be modified by applying the chain rule of probability ($p(a, b|c) = p(b|c)\ p(a|b, c)$) to expand $p(x_t, \theta_t|\theta_{t-1}, Y_{0:t})$ as follows:

$$\begin{aligned} p(x_t, \theta_t|\theta_{t-1}, Y_{0:t}) &= p(\theta_t|\theta_{t-1}, Y_{0:t})\ p(x_t|\theta_t, \theta_{t-1}, Y_{0:t}) \\ &\quad \text{expanding } Y_{0:t} \triangleq (u_{0:t-1}, i_{1:t}, z_{1:t}) \text{ in the second term} \\ &= p(\theta_t|\theta_{t-1}, Y_{0:t})\ p(x_t|\theta_t, \theta_{t-1}, u_{0:t-1}, i_{1:t}, z_{1:t}) \\ &= p(\theta_t|\theta_{t-1}, Y_{0:t})\ p(x_t|z_t, \theta_t, \theta_{t-1}, u_{0:t-1}, i_{1:t}, z_{1:t-1}). \end{aligned}$$

Writing $\alpha = (\theta_t, \theta_{t-1}, u_{0:t-1}, i_{1:t}, z_{1:t-1})$ this can be expressed as

$$p(x_t, \theta_t|\theta_{t-1}, Y_{0:t}) = p(\theta_t|\theta_{t-1}, Y_{0:t})\ p(x_t|z_t, \alpha). \tag{3.10}$$

Note that $p(x_t|z_t, \alpha)$ can be expanded as follows

$$\begin{aligned} p(x_t|z_t, \alpha) &= \frac{p(x_t, z_t, \alpha)}{p(z_t, \alpha)} \\ &= \frac{p(z_t|x_t, \alpha)p(x_t, \alpha)}{p(z_t, \alpha)} \\ &= \frac{p(z_t|x_t, \alpha)p(x_t|\alpha)p(\alpha)}{p(z_t|\alpha)p(\alpha)} \\ &= \frac{p(z_t|x_t, \alpha)p(x_t|\alpha)}{p(z_t|\alpha)}. \end{aligned}$$

Substituting into 3.10 obtains

$$p(x_t, \theta_t|\theta_{t-1}, Y_{0:t}) = p(\theta_t|\theta_{t-1}, Y_{0:t})\ \frac{p(z_t|x_t, \alpha)\ p(x_t|\alpha)}{p(z_t|\alpha)}. \tag{3.11}$$

Expanding α in $p(x_t|\alpha)$ allows 3.11 to be rewritten as

$$p(x_t, \theta_t|\theta_{t-1}, Y_{0:t}) = p(\theta_t|\theta_{t-1}, Y_{0:t})\ p(z_t|x_t, \alpha)\ p(z_t|\alpha)^{-1}\ p(x_t|\alpha) \tag{3.12}$$

$$\begin{aligned} &= p(\theta_t|\theta_{t-1}, Y_{0:t})\ p(z_t|x_t, \alpha)\ p(z_t|\alpha)^{-1} \\ &\quad p(x_t|i_t, \theta_t, \theta_{t-1}, u_{0:t-1}, i_{1:t-1}, z_{1:t-1}). \end{aligned} \tag{3.13}$$

Define $\beta = (\theta_t, \theta_{t-1}, u_{0:t-1}, i_{1:t-1}, z_{1:t-1})$. Note that $\alpha = (\beta, i_t)$. Then

$$p(x_t, \theta_t|\theta_{t-1}, Y_{0:t}) = \ p(\theta_t|\theta_{t-1}, Y_{0:t})\ p(z_t|x_t, \alpha)\ p(z_t|\alpha)^{-1}\ p(x_t|i_t, \beta). \tag{3.14}$$

Expand $p(x_t|i_t, \beta)$ using the same derivation as used to obtain 3.11 $\left(p(a|b,c) = \frac{p(b|a,c)p(a,c)}{p(b|c)}\right)$

$$= p(\theta_t|\theta_{t-1}, Y_{0:t})\ \ p(z_t|x_t, \alpha)\ p(z_t|\alpha)^{-1}\ \frac{p(i_t|x_t, \beta)\ p(x_t|\beta)}{p(i_t|\beta)}. \tag{3.15}$$

$$= p(\theta_t|\theta_{t-1}, Y_{0:t})\ p(z_t|x_t, \alpha)\ p(z_t|\alpha)^{-1}\ p(i_t|x_t, \beta)\ p(i_t|\beta)^{-1}\ p(x_t|\beta) \tag{3.16}$$

Finally, applying the theorem of total probability to $p(x_t|\beta)$ in terms of x_{t-1} results in the following equivalence:

$$p(x_t, \theta_t|\theta_{t-1}, Y_{0:t}) = p(\theta_t|\theta_{t-1}, Y_{0:t})\ p(z_t|x_t, \alpha)\ p(z_t|\alpha)^{-1}\ p(i_t|x_t, \beta)\ p(i_t|\beta)^{-1} \int p(x_t|x_{t-1}, \beta)\ p(x_{t-1}|\beta)dx_{t-1}. \tag{3.17}$$

Note that the derivation of 3.17 relies only on properties of conditional probability. Substituting Equation 3.17 into Equation 3.9 obtains

$$Bel(X_t) = \int p(\theta_t|\theta_{t-1}, Y_{0:t})\ p(z_t|x_t, \alpha)\ p(z_t|\alpha)^{-1}\ p(i_t|x_t, \beta)\ p(i_t|\beta)^{-1} \left[\int p(x_t|x_{t-1}, \beta)\ p(x_{t-1}|\beta)dx_{t-1}\right]\ p(\theta_{t-1}|Y_{0:t})\ d\theta_{t-1}. \tag{3.18}$$

It is possible to simplify several of these distributions under a Markov assumption (see Section 2.1.1). Based on the assumption that the state $\{x_t, \theta_t\}$ is complete at time t when considering the full pose, and that $\{\theta_t\}$ is complete when considering only the orientation, then all data preceding those states may be ignored in the distributions of equation 3.18. The measurement equations (and associated normalizers) can then be simplified as follows:

$$\begin{aligned} p(z_t|x_t, \alpha)\ p(z_t|\alpha)^{-1} &= p(z_t|x_t, z_{1:t-1}, \theta_t, \theta_{t-1}, u_{0:t-1}, i_{1:t}) \\ &\quad p(z_t|z_{1:t-1}, \theta_t, \theta_{t-1}, u_{0:t-1}, i_{1:t})^{-1} \\ &= p(z_t|x_t, \theta_t)\ p(z_t|\theta_t)^{-1} \end{aligned} \tag{3.19}$$

and

$$\begin{aligned} p(i_t|x_t, \beta)\ p(i_t|\beta)^{-1} &= p(i_t|x_t, z_{1:t-1}, \theta_t, \theta_{t-1}, u_{0:t-1}, i_{1:t-1}) \\ &\quad p(i_t|z_{1:t-1}, \theta_t, \theta_{t-1}, u_{0:t-1}, i_{1:t-1}) \\ &= p(i_t|x_t, \theta_t)\ p(i_t|\theta_t)^{-1}. \end{aligned} \tag{3.20}$$

Note in Equation 3.20 that a further simplification is possible if the assumption that i_t is independent of x_t holds. Such an assumption is not necessarily true, i.e., magnetic

sensors may behave differently near a ship's hull. Nonetheless, in certain situations it may reduce the required computational effort; this potential simplification is discussed in more detail later in this section.

Further, $p(x_t|x_{t-1}, \beta)$ contains the complete state $\{x_{t-1}, \theta_{t-1}\}$, allowing us to ignore most of the parameters (with the exception of u_{t-1} since it occurs after the complete state). Although the state θ_t occurs after the (assumed) complete state, we ignore it, since the orientation of a device at time t does not affect the likelihood of having *arrived* at x_{t-1}. Only θ_{t-1} is needed for a complete state in $p(\theta_t|\theta_{t-1}, Y_{0:t})$, since we assume that orientation may be estimated independently of x_{t-1}. Therefore only the data $\{u_{t-1}, i_t, z_t\}$ that occurs after the complete state must still be considered. Any evidence related to the device's position may also be eliminated, again based on the assumption that orientation is not affected by position. As a result we may also ignore z_t, since by definition it contains no information related to the device's orientation. This then results in the following simplifications:

$$\begin{aligned} p(x_t|x_{t-1}, \beta) &= p(x_t|x_{t-1}, u_{0:t-1}, z_{1:t-1}, i_{1:t-1}, \theta_t, \theta_{t-1}) \\ &= p(x_t|x_{t-1}, \theta_{t-1}, u_{t-1}) \end{aligned} \tag{3.21}$$

and

$$\begin{aligned} p(\theta_t|\theta_{t-1}, Y_{0:t}) &= p(\theta_t|\theta_{t-1}, u_{0:t-1}, z_{1:t}, i_{1:t}) \\ &= p(\theta_t|\theta_{t-1}, u_{t-1}, i_t). \end{aligned} \tag{3.22}$$

Substituting these equations back into equation 3.18 results in

$$\begin{aligned} Bel(X_t) = \int & p(\theta_t|\theta_{t-1}, u_{t-1}, i_t)\; p(z_t|x_t, \theta_t)\; p(z_t|\theta_t)^{-1}\; p(i_t|x_t, \theta_t)\; p(i_t|\theta_t)^{-1} \\ & \int p(x_t|x_{t-1}, \theta_{t-1}, u_{t-1}) \\ & p(x_{t-1}|\theta_t, \theta_{t-1}, u_{0:t-1}, i_{1:t-1}, z_{1:t-1})\; dx_{t-1}\; p(\theta_{t-1}|Y_{0:t})\; d\theta_{t-1}. \end{aligned} \tag{3.23}$$

The terms $p(z_t|x_t, \theta_t),\ p(z_t|\theta_t)^{-1},\ p(i_t|x_t, \theta_t)$ and $p(i_t|\theta_t)^{-1}$ are constant with respect to the integral $d\theta_{t-1}$ and may therefore be moved out of the integral. Finally, moving $p(\theta_{t-1}|Y_{0:t})$ into the inner integral results in:

$$\begin{aligned} Bel(X_t) = & p(z_t|x_t, \theta_t)\; p(z_t|\theta_t)^{-1}\; p(i_t|x_t, \theta_t)\; p(i_t|\theta_t)^{-1} \\ & \int p(\theta_t|\theta_{t-1}, u_{t-1}, i_t) \int p(x_t|x_{t-1}, \theta_{t-1}, u_{t-1}) \\ & p(x_{t-1}|\theta_t, \theta_{t-1}, u_{0:t-1}, z_{1:t-1}, i_{1:t-1})\; p(\theta_{t-1}|Y_{0:t})\; dx_{t-1}\; d\theta_{t-1}. \end{aligned} \tag{3.24}$$

Note that the u_{t-1} term in $p(x_{t-1}|\theta_{t-1}, u_{0:t-1}, z_{1:t-1}, i_{1:t-1})$ is executed after the state x_{t-1}, while the state θ_t occurs after x_{t-1}, allowing both to be safely removed. Also

observe that $u_{0:t-2}, z_{1:t-1}, i_{1:t-1} \triangleq Y_{t-1}$. We may then simplify the last two terms of equation 3.24 as follows:

$$\begin{aligned} & p(x_{t-1}|\theta_{t-1}, u_{0:t-2}, z_{1:t-1}, i_{1:t-1})\ p(\theta_{t-1}|Y_{0:t}) \\ =& p(x_{t-1}|\theta_{t-1}, Y_{t-1}) p(\theta_{t-1}|Y_{t-1}) \\ =& p(x_{t-1}, \theta_{t-1}|Y_{t-1}) \\ =& Bel(X_{t-1}). \end{aligned} \tag{3.25}$$

Substituting this equivalence back in results in the desired recursive filter:

$$\begin{aligned} Bel(X_t) = p(z_t|\theta_t)^{-1}\ p(i_t|\theta_t)^{-1}\ p(z_t|x_t, \theta_t)\ p(i_t|x_t, \theta_t) \\ \int p(\theta_t|\theta_{t-1}, u_{t-1}, i_t) \int p(x_t|x_{t-1}, \theta_{t-1}, u_{t-1}) \\ Bel(X_{t-1}) dx_{t-1}\ d\theta_{t-1}. \end{aligned} \tag{3.26}$$

As mentioned earlier (Equation 3.20), a simplification could be introduced if the assumption $p(i_t|x_t, \theta_t) = p(i_t|\theta_t)$ is valid, resulting in $p(i_t|\theta_t)\ p(i_t|\theta_t)^{-1} = 1$. This situation may arise if the sensors employed are able to purely measure orientation-relevant properties, while never being affected by where the measurements are taken. The experiments discussed in Chapter 4 make this assumption. For convenience, define the normalizing constant

$$\eta = \begin{cases} p(z_t|\theta_t)^{-1} & \text{if } p(i_t|x_t, \theta_t) = p(i_t|\theta_t) \\ p(z_t|\theta_t)^{-1}\ p(i_t|\theta_t)^{-1} & \text{otherwise} \end{cases}$$

The belief of X_{t-1} is propagated forward, prior to measurement, using the conditional probabilities $p(x_t|x_{t-1}, \theta_{t-1}, u_{t-1})$ and $p(\theta_t|\theta_{t-1}, u_{t-1}, i_t)$. The measurement likelihoods represented by $p(z_t|x_t, \theta_t)$ and $p(i_t|x_t, \theta_t)$ are then used to assign likelihood weights to the predicted state, using the collected measurement data.

Although this filter may be implemented any number of ways, here we implement the orientation likelihood $p(\theta_t|\theta_{t-1}, u_{t-1}, i_t)$ using an EKF (from [59], see Section 2.3.2) and the position likelihood $p(x_t|x_{t-1}, \theta_{t-1}, u_{t-1})$ using a particle filter (see Section 2.2.2). Furthermore observe that only a single estimation process is required for $p(\theta_t|\theta_{t-1}, u_{t-1}, i_t)$ resulting in significant computational savings.

3.1.1 The normalizing term η

The filter as presented in Equation 3.26 contains the two terms $p(z_t|\theta_t)^{-1}$ and $p(i_t|\theta_t)^{-1}$. The normalizer $\eta = p(z_t|\theta_t)^{-1} p(i_t|\theta_t)^{-1}$ can be computed explicitly for the example shown in Section 3.3. Although easily computed for such a case, their computation in

general is complex. As a result, an approximation is generally used [7], which greatly simplifies the problem. By approximating the terms $p(z_t|\theta_t)^{-1}$ and $p(i_t|\theta_t)^{-1}$ as constant normalizers, they may be ignored entirely in practice. Instead, the sum of the computed weights (over a portion of the space, i.e., all particles in the case of a particle filter) is assumed to sufficiently approximate the normalizer. This is based on the assumption that the sum of a likelihood distribution should sum to one. Table 3.2 shows the result of using this approximation, compared to the actual result, shown in Table 3.1.

3.2 Algorithm process

The previous section developed a recursive approximation for the robot's belief, which may be restated as:

$$Bel(X_t) = \eta\, p_{range}\, [p_{imu}] \int model_{\text{Orientation}} \int model_{\text{Motion}}\, Bel(X_{t-1})\, dx_{t-1}\, d\theta_{t-1}.$$

This filter can be implemented using a particle filter to estimate the position of the robot, and a Kalman filter to estimate the orientation (although a different estimator could be used instead). This hybrid implementation takes advantage of the efficiency of the Kalman filter, while using a particle filter to sample the more difficult to estimate position state, for which no direct sensor readings are available. The implementation details are outlined below. Define each particle as the tuple $P_t = \{\theta_t, x_t, w_t\}$, where θ_t is the orientation of the device (as estimated by the Kalman filter). The position x_t at state X_t is known with certainty w_t.

Step 0. Filter initialization
Place N particles over the *position* space $\{x, y, z\}$, either uniformly or in some focused manner. At this time also initialize the Extended Kalman filter that estimates $\vec{\theta}$ and any required likelihood structures. The implementation for this work uses a single Extended Kalman filter, avoiding the computational expense of updating a Extended Kalman filter for each particle. This choice is further motivated by the fact that the orientation is assumed independent from the position, with individual Kalman filters theoretically converging to the same orientation estimate.

The filter can now be updated at time t as follows:
For Loop

Step 1. Update the Extended Kalman Filter.
Using IMU data i_t, and motion estimate u_{t-1} (provided by the egomotion estimation process [37]) create a best estimate θ_t of the orientation for the current state at time t. The Kalman filter should be allowed to converge to a reasonably

certain estimate of the device's orientation, before moving onto the next step. This is motivated by the fact that computation of the range likelihood and motion model is essentially meaningless without a reasonable estimate of the orientation.

Step 2. Particle filter
For each particle p_i, estimate the next position state x_t according to the control input u_{t-1} and the motion model. See Section 3.4 for details on the motion model. Using the motion model, the control u_{t-1} is converted into an estimate of the actual resulting motion, $\{\Delta_\theta, \Delta_x\}$, resolved in the robot orientation frame defined by θ_{t-1}. This provides an *a priori* best estimate of the next state, and is added to the current estimate p_i, resulting in the updated particle position.

Step 3. Weighting
For each particle in the particle set, compute a weight update describing the likelihood that the particle describes the true pose of the robot. For each particle i, the weight is updated according to:

$$w_t^i = p(z_t|x_t^i, \theta_t^i)\ p(i_t|x_t^i, \theta_t^i)\ w_{t-1}^i.$$

Finally, the weights of the particle set are normalized.

$$w_t^i \leftarrow w_t^i \sum_{j=0}^{N} \frac{1}{w_t^j}.$$

Step 4. Re-sampling
Especially when the filter is initialized using a uniform distribution, the majority of particle weights will tend towards zero. This can lead to filter failure, but can typically be prevented by resampling the particles occasionally. For information on resampling refer to Section 2.2.2.

End For

It is important to efficiently compute the likelihoods at each update, since both the map and sensors utilize 3D data. Computing the distributions defined in Section 3.1 is discussed in the following sections.

Section 2.3.2	Orientation update	$p(\theta_t\|\theta_{t-1}, u_{t-1}, i_t)$
Section 3.4	Motion model	$p(x_t\|x_{t-1}, \theta_{t-1}, u_{t-1})$
Section 3.5	Range likelihood	$p(z_t\|x_t, \theta_t)$

However, a simple example to illustrate the overall technique is provided first, considering the dimensionality of the overall problem.

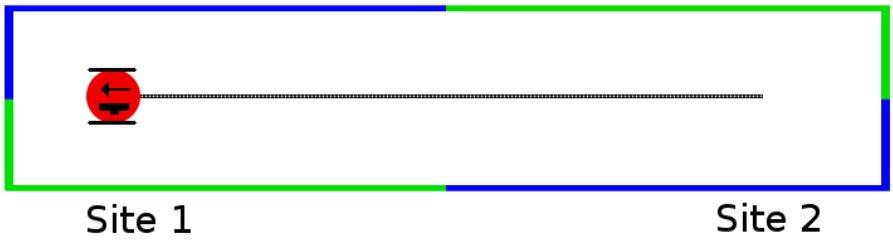


Figure 3.2: The robot's environment for this example. The walls are painted in such a way that, lacking IMU information, the color of the wall is insufficient to determine the position of the robot. The diagram illustrates the final state of the robot in this example: it is located at Site 1, facing south (note the camera on the robot). Translation of the robot is limited to moving in the direction of the arrow.

3.3 Simple example

This section provides a simple example of the filter applied to an estimation task. Consider a mobile robot placed in a hallway; the robot is able to turn around and face opposite its current direction (i.e., turn to face either North or South). The robot may also either move to the other end of the hallway, or it may remain at its current location. The west end of the hallway (Site 1) has been painted blue/green, while the east end (Site 2) is green/blue (See Figure 3.2). The state $X_t = \{x_t, \theta_t\}$ is to be estimated. Define x_t as the position of the robot at time t, limited to either end of the hallway, i.e., $x_t \in (\text{Site 1}, \text{Site 2})$. In addition, let θ_t be the direction faced by the robot, i.e., $\theta_t \in (\text{North}, \text{South})$. The robot is equipped with sensors $Y = \{Z, I\}$, where Z senses the colour of the hallway, and I senses the direction the robot is facing. Controls to the robot consist of the pair $u_t \in ((\text{Turn}, \text{Stay}), (\text{Move}, \text{Stay}))$. As an arbitrary choice, the robot is only able to translate in the direction it is facing (i.e., in the direction of the arrow on the robot), due to the mechanics selected for this robot. As a result, when facing south at Site 1, the robot is likely to remain where is it when given a $move$ command. Table 3.3 details the plant and measurement likelihoods for each of the four distributions that are required. The robot's environment is illustrated in Figure 3.2.

Initially, no information regarding the state of the system is known; therefore the belief is initialized to be uniform over all possible robot states:

$$Bel(\text{Site 1}, \text{North})_0 = Bel(\text{Site 1}, \text{South})_0 =$$
$$Bel(\text{Site 2}, \text{North})_0 = Bel(\text{Site 2}, \text{South})_0 = 0.25.$$

i_t	x_t	$\theta_t = N_{orth}$	$\theta_t = S_{outh}$
North	Site 1	0.9	0.1
North	Site 2	0.9	0.1
South	Site 1	0.1	0.9
South	Site 2	0.1	0.9

(a) $p(i_t|x_t, \theta_t)$

z_t	x_t	$\theta_t = N_{orth}$	$\theta_t = S_{outh}$
Blue	Site 1	0.85	0.15
Green	Site 1	0.15	0.85
Blue	Site 2	0.15	0.85
Green	Site 2	0.85	0.15

(b) $p(z_t|x_t, \theta_t)$

θ_t	θ_{t-1}	u_{t-1}	$i_t = N_{orth}$	$i_t = S_{outh}$
North	North	Turn	0.8	0.1
South	North	Turn	0.2	0.9
North	South	Turn	0.9	0.2
South	South	Turn	0.1	0.8
North	North	Stay	1.0	0.5
South	North	Stay	0.0	0.5
North	South	Stay	0.5	0.0
South	South	Stay	0.5	1.0

(c) $p(\theta_t|u_{t-1}, \theta_{t-1}, i_t)$

x_t	x_{t-1}	u_{t-1}	$\theta_{t-1} = N_{orth}$	$\theta_{t-1} = S_{outh}$
Site 1	Site 1	Move	0.1	.9
Site 2	Site 1	Move	0.9	0.1
Site 1	Site 2	Move	0.1	0.9
Site 2	Site 2	Move	0.9	0.1
Site 1	Site 1	Stay	1	1
Site 2	Site 1	Stay	0	0
Site 1	Site 2	Stay	0	0
Site 2	Site 2	Stay	1	1

(d) $p(x_t|x_{t-1}, u_{t-1}, \theta_{t-1})$

Figure 3.3: The measurement probabilities (a,b) and models of the result for the robot's turning and moving capabilities (c,d). The measurement model in (a) shows the IMU indicates the correct orientation 90% of the time, independent of the position x_t. The camera sensor (b) detects the correct color in 85% of measurements, while misreading the color (i.e., due to lighting) 15% of the time. In (c) the result of rotation commands is modeled; when the measured orientation i_t conflicts with the final state, the measurement is given more weight. Finally, moving from one end of the hallway to the other occurs with 90% likelihood if the arrow in Figure 3.2 is aimed down the hallway. The robot remains at the current site 90% of the time if the arrow is aimed at the nearby wall. The "Stay" command always results in remaining in the current state.

Using the filter defined in Section 3.1, the posterior belief at time t denoted $\overline{Bel}_t$ is estimated as

$$\overline{Bel}(x_t, \theta_t) = \int p(\theta_t|\theta_{t-1}, u_{t-1}, i_t) \int p(x_t|x_{t-1}, \theta_{t-1}, u_{t-1})\ Bel(x_{t-1}, \theta_{t-1})\ dx_{t-1}\ d\theta_{t-1}.$$

Given the discrete nature of our example, this may be restated as:

$$\overline{Bel}(x_t, \theta_t) = \sum_{\theta_{t-1}} p(\theta_t|\theta_{t-1}, u_{t-1}, i_t) \sum_{x_{t-1}} p(x_t|x_{t-1}, \theta_{t-1}, u_{t-1})\ Bel(x_{t-1}, \theta_{t-1}).$$

The measurement update incorporates the sensor measurements and is computed as:

$$Bel(x_t, \theta_t) = \eta\ p(z_t|x_t, \theta_t)\ [p(i_t|x_t, \theta_t)]\ \overline{Bel}(x_t, \theta_t). \tag{3.27}$$

Based on these equations, we may then proceed with estimating the state of our system.

Given the control $u_0 = \{\text{Stay, Stay}\}$ and measurements $z_1 = \{\text{Green}\}$ and $i_1 = \{\text{North}\}$, the estimated belief may be evaluated according to

$$\begin{aligned}\overline{Bel}(x_1, \theta_1) = p(\theta_1|\text{North, Stay}, North) * [&p(x_1|\text{Site 1, North, Stay})\ Bel(\text{Site 1, North})_0 + \\ &p(x_1|\text{Site 1, South, Stay})\ Bel(\text{Site 1, South})_0] + \\ p(\theta_1|\text{South, Stay}, North) * [&p(x_1|\text{Site 2, North, Stay})\ Bel(\text{Site 2, North})_0 + \\ &p(x_1|\text{Site 2, South, Stay})\ Bel(\text{Site 2, South})_0]\end{aligned}$$

Which, for each of the four cases, evaluates to:

$$\begin{aligned}\overline{Bel}(x = \text{Site 1}, \theta = \text{North})_1 &= 0.375 \\ \overline{Bel}(x = \text{Site 1}, \theta = \text{South})_1 &= 0.125 \\ \overline{Bel}(x = \text{Site 2}, \theta = \text{North})_1 &= 0.375 \\ \overline{Bel}(x = \text{Site 2}, \theta = \text{South})_1 &= 0.125\end{aligned}$$

Substituting the measurements z_1 and i_1 into equation 3.27 results in

$$\begin{aligned}Bel(x = \text{Site 1}, \theta = \text{North})_1 &= \eta\ p(\text{Green}|S1, N)\ p(\text{North}|S1, N)\ \overline{Bel}(S1, N)_1 \\ &= \eta\ 0.15 * 0.9 * 0.375 \approx \eta\ 0.028 \\ Bel(x = \text{Site 1}, \theta = \text{South})_1 &= \eta\ 0.85 * 0.1 * 0.125 \approx \eta\ 0.053 \\ Bel(x = \text{Site 2}, \theta = \text{North})_1 &= \eta\ 0.85 * 0.9 * 0.375 \approx \eta\ 0.159 \\ Bel(x = \text{Site 2}, \theta = \text{South})_1 &= \eta\ 0.15 * 0.1 * 0.125 \approx \eta\ 0.009 \\ \text{where the normalizer } \eta &\approx [0.028 + 0.053 + 0.159 + 0.009]^{-1} \approx 4.00\end{aligned}$$

The belief at t = 1 is then represented as

$$\begin{aligned}Bel(x = \text{Site 1}, \theta = \text{North})_1 &\approx 0.113 \\ Bel(x = \text{Site 1}, \theta = \text{South})_1 &\approx 0.213 \\ Bel(x = \text{Site 2}, \theta = \text{North})_1 &\approx 0.638 \\ Bel(x = \text{Site 2}, \theta = \text{South})_1 &\approx 0.038\end{aligned}$$

Given the control $u_1 = \{\text{Turn, Stay}\}$ and measurements $z_2 = \{\text{Blue}\}$ and $i_2 = \{\text{South}\}$, the estimated belief may be evaluated according to

$$\begin{aligned}\overline{Bel}(x_2, \theta_2) = p(\theta_2|\text{North, Turn}, South) * [p(x_2|\text{Site 1, North, Stay})\ Bel(\text{Site 1, North})_1 + \\ p(x_2|\text{Site 1, South, Stay})\ Bel(\text{Site 1, South})_1] + \\ p(\theta_2|\text{South, Turn}, South) * [p(x_2|\text{Site 2, North, Stay})\ Bel(\text{Site 2, North})_1 + \\ p(x_2|\text{Site 2, South, Stay})\ Bel(\text{Site 2, South})_1]\end{aligned}$$

Evaluating for each of the four cases results in

$$\begin{aligned}\overline{Bel}(x = \text{Site 1}, \theta = \text{North})_2 &\approx 0.054 \\ \overline{Bel}(x = \text{Site 1}, \theta = \text{South})_2 &\approx 0.271 \\ \overline{Bel}(x = \text{Site 2}, \theta = \text{North})_2 &\approx 0.071 \\ \overline{Bel}(x = \text{Site 2}, \theta = \text{South})_2 &\approx 0.604\end{aligned}$$

Substituting the measurements z_2 and i_2 into equation 3.27 results in

$$\begin{aligned}Bel(x = \text{Site 1}, \theta = \text{North})_2 &= \eta\ p(\text{Blue}|S1, N)\ p(\text{South}|S1, N)\ \overline{Bel}(S1, N)_2 \\ &\approx \eta\ 0.85 * 0.1 * 0.054 \approx \eta\ 0.023 \\ Bel(x = \text{Site 1}, \theta = \text{South})_2 &\approx \eta\ 0.15 * 0.9 * 0.271 \approx \eta\ 0.020 \\ Bel(x = \text{Site 2}, \theta = \text{North})_2 &\approx \eta\ 0.15 * 0.1 * 0.071 \approx \eta\ 0.005 \\ Bel(x = \text{Site 2}, \theta = \text{South})_2 &\approx \eta\ 0.85 * 0.9 * 0.604 \approx \eta\ 0.257 \\ \text{where the normalizer } \eta &\approx 3.277\end{aligned}$$

The belief at t = 2 then evaluates to

$$\begin{aligned}Bel(x = \text{Site 1}, \theta = \text{North})_2 &\approx 0.075 \\ Bel(x = \text{Site 1}, \theta = \text{South})_2 &\approx 0.067 \\ Bel(x = \text{Site 2}, \theta = \text{North})_2 &\approx 0.018 \\ Bel(x = \text{Site 2}, \theta = \text{South})_2 &\approx 0.841\end{aligned}$$

Given the control $u_2 = \{\text{Stay, Move}\}$ and measurements $z_3 = \{\text{Green}\}$ and $i_3 = \{\text{South}\}$, the estimated belief may be evaluated according to

$$\begin{aligned}\overline{Bel}(x_3, \theta_3) = p(\theta_3|\text{North, Stay}, South) * [&p(x_3|\text{Site 1, North, Move})\ Bel(\text{Site 1, North})_2 + \\ &p(x_3|\text{Site 1, South, Move})\ Bel(\text{Site 1, South})_2] + \\ p(\theta_3|\text{South, Stay}, South) * [&p(x_3|\text{Site 2, North, Move})\ Bel(\text{Site 2, North})_2 + \\ &p(x_3|\text{Site 2, South, Move})\ Bel(\text{Site 2, South})_2]\end{aligned}$$

Evaluating for each of the four cases results in

$$\begin{aligned}\overline{Bel}(x = \text{Site 1}, \theta = \text{North})_3 &\approx 0.005 \\ \overline{Bel}(x = \text{Site 1}, \theta = \text{South})_3 &\approx 0.821 \\ \overline{Bel}(x = \text{Site 2}, \theta = \text{North})_3 &\approx 0.042 \\ \overline{Bel}(x = \text{Site 2}, \theta = \text{South})_3 &\approx 0.132\end{aligned}$$

Substituting the measurements z_3 and i_3 into equation 3.27 results in

$$\begin{aligned}Bel(x = \text{Site 1}, \theta = \text{North})_3 &= \eta\ p(\text{Green}|S1, N)\ p(\text{South}|S1, N)\ \overline{Bel}(S1, N)_3 \\ &\approx \eta\ 0.15 * 0.1 * 0.005 \approx \eta\ 0.35E^{-3} \\ Bel(x = \text{Site 1}, \theta = \text{South})_3 &\approx \eta\ 0.85 * 0.9 * 0.821 \approx \eta\ 0.349 \\ Bel(x = \text{Site 2}, \theta = \text{North})_3 &\approx \eta\ 0.85 * 0.1 * 0.042 \approx \eta\ 0.018 \\ Bel(x = \text{Site 2}, \theta = \text{South})_3 &\approx \eta\ 0.15 * 0.9 * 0.132 \approx \eta\ 0.010 \\ &\text{where the normalizer } \eta \approx 2.652\end{aligned}$$

The belief at t = 3 then evaluates to

$$\begin{aligned}Bel(x = \text{Site 1}, \theta = \text{North})_3 &\approx 0.919E^{-3} \\ Bel(x = \text{Site 1}, \theta = \text{South})_3 &\approx 0.926 \\ Bel(x = \text{Site 2}, \theta = \text{North})_3 &\approx 0.047 \\ Bel(x = \text{Site 2}, \theta = \text{South})_3 &\approx 0.026\end{aligned}$$

Table 3.1 summarizes the result of the filter over the course of three time steps. Initially the belief is uniform, but incorporating the controls results in the estimate of the

	$Bel()_0$	$\overline{Bel}()_1$	$Bel()_1$	$\overline{Bel}()_2$	$Bel()_2$	$\overline{Bel}()_3$	$Bel()_3$
Bel(Site 1, North)	0.25	0.375	0.113	0.054	0.075	0.005	0.000
Bel(Site 1, South)	0.25	0.125	0.213	0.271	0.067	**0.821**	**0.926**
Bel(Site 2, North)	0.25	0.375	**0.638**	0.071	0.018	0.042	0.047
Bel(Site 2, South)	0.25	0.125	0.038	**0.604**	**0.841**	0.132	0.026
Control	{Stay,	Stay}	{Turn,	Stay}	{Stay,	Move}	
Sensor $\{z_t, i_t\}$	{Green,	North}	{Blue,	South}	{Green,	South}	

Table 3.1: Evolving belief regarding the where the robot is located, and which direction it is facing, with the most likely state shown in bold. After 3 iterations, the filter believes with 0.926 probability that it is facing South at Site 1. Note that, regardless of whether $p(i_t|x_t, \theta_t)$ and $p(i_t|\theta_t)^{-1}$ are computed, the results are identical.

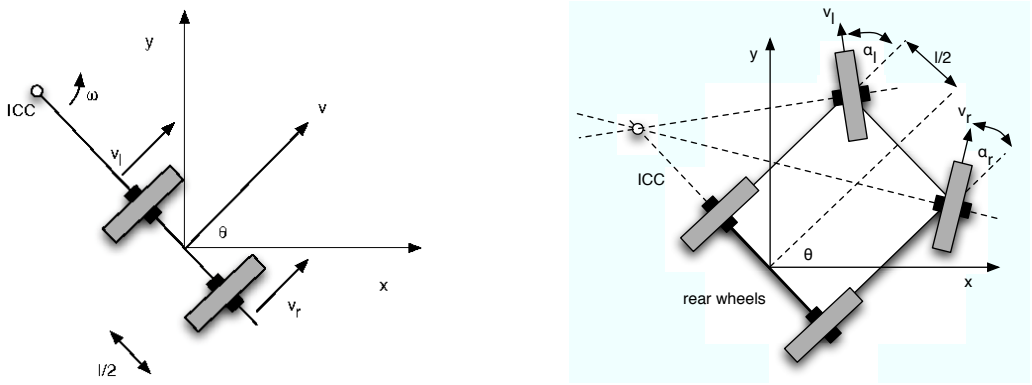
	$Bel()_0$	$\overline{Bel}()_1$	$Bel()_1$	$\overline{Bel}()_2$	$Bel()_2$	$\overline{Bel}()_3$	$Bel()_3$
Bel(Site 1, North)	0.25	0.375	0.145	0.021	0.002	0.000	0.000
Bel(Site 1, South)	0.25	0.125	0.030	0.154	0.035	**0.896**	**0.981**
Bel(Site 2, North)	0.25	0.375	**0.820**	0.083	0.002	0.002	0.000
Bel(Site 2, South)	0.25	0.125	0.005	**0.742**	**0.960**	0.102	0.020
Control	{Stay,	Stay}	{Turn,	Stay}	{Stay,	Move}	
Sensor $\{z_t, i_t\}$	{Green,	North}	{Blue,	South}	{Green,	South}	

Table 3.2: In the previous example (summarized in Table 3.1), the normalizer $\eta = p(z_t|\theta_t)^{-1}p(i_t|\theta_t)^{-1}$ was computed explicitly. The same experiment is presented here, but the normalizer η is now approximated(as discussed in Section 3.1.1). The final certainty is now 98.1%, compared to 92.6% in the previous example.

belief at $t = 1$ showing that the robot is more likely to be facing north. The robot correctly updates the belief after observing a green wall, and receiving a north reading from the IMU at time $t = 1$. The belief $Bel()_1$ is 75% certain that it is facing north, while most likely being at Site 2. After being commanded to turn around, and incorporating measurements, the belief changes so that with 84% certainty the robot is facing south at Site 2. Finally, moving the robot while not changing the direction results in the final belief, at timestep $t = 3$, that the robot is located at Site 1 and facing south with 92.6% certainty.

3.4 Motion model

The motion model is used to update the most recent estimate of the robot's position, taking commanded motion into account. Robot motion is inherently uncertain; as such,



(a) (b)

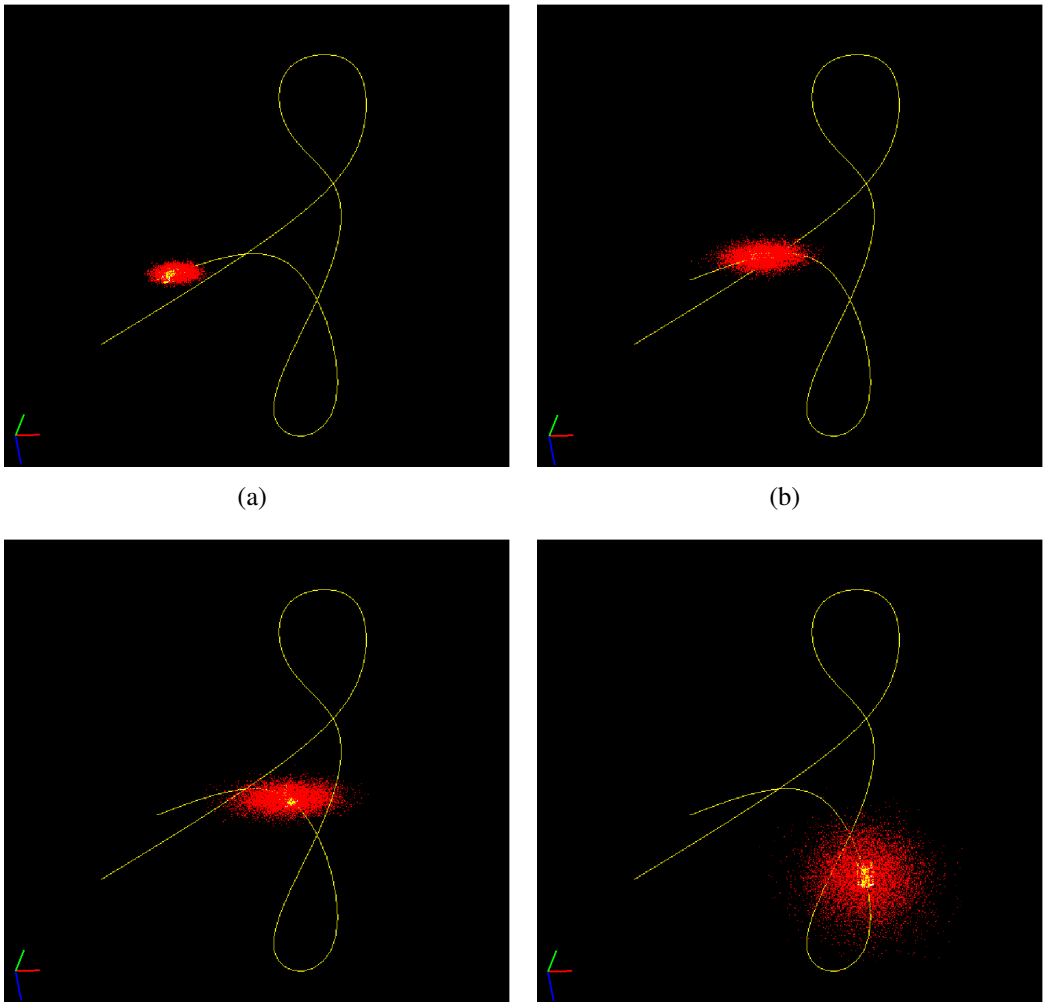
Figure 3.4: Common kinematic models for ground based vehicles, used for estimating a vehicle's motion, include (a) differential drive and (b) Ackerman steering. For full details refer to [22].

attempts to model the result of commanded motion must consider this uncertainty. The motion model takes the control input and estimates the likely outcome of executing that action, as shown in Figure 3.4.

Depending on the motion and control characteristics of the robot, the input could contain any number of parameters. The parameters may be limited to rotation and translation parameters, depending on the kinematics of the vehicle, such as differential drive or Ackerman steering for ground vehicles (see Figure 3.4). Kinematics approaches attempt to determine the outcome of commanded motion, based, for example, on the commanded number of revolutions for each wheel. Ground contact vehicles commonly use odometry, which attempts to record the result of commanded motion. The result from odometry, plus an appropriate noise model to model wheel slippage, may be sufficient to use as a motion model in such a case.

Vehicles operating in 3D environments are not necessarily able to collect odometry in the same sense. Lacking a surface to track, such vehicles must rely on fins, thrusters and control surfaces to effect motion. The indirect data of such locomotion requires the robot to rely more heavily on sensors to monitor its egomotion. For example, [37] describes a method using vision techniques coupled with an IMU.

Given a reasonably accurate estimate of the robot's motion (in the robot reference frame) using egomotion estimation, a very simple motion model can be used. The



(a) (b)

(c) (d)

Figure 3.5: The result of applying a motion model in 3D without corrective inputs. Figure (a) shows an initial position distribution, followed by the robot swimming forward. As the robot moves forward in Figure (b) and (c) the positional uncertainty increases. Turning a corner in (d), the robot's position has become quite uncertain along two axis. The motion model in this instance applies independent Gaussian noise proportional to the egomotion estimate along each axis.

motion model must still consider variations of the estimate, as the motion estimate itself may be prone to noise. Given the lack of a more detailed error model, a straightforward approach is to simply apply an additive Gaussian noise to the estimate. Of course, if a more refined error model is available, it could be used instead. For each particle an independent motion error is generated from the noise distribution for the estimated robot motion. In the case that particles are clustered, essentially describing the same pose, this added random noise allows them to expand into a larger surrounding space. Spatially separating the particles serves several purposes, the most obvious being that multiple particles describing the same (x, y, z) position is wasteful. In addition, adding a sufficiently large error to the motion estimate (for some of the particles) may push a particle into a better, nearby region, from where a resampling would result in potentially seeding the overall filter. Particles that wander too far from the optimal solution are likely to be killed off during the resampling if they land in unlikely locations.

This work uses a simple additive Gaussian noise as the motion model, where the variance of the Gaussian is set proportionally to the robot's estimated displacement, as recovered from the egomotion estimation. Although this approach overestimates the motion error, it allows particles to spread sufficiently at any update step in order to evolve into a better estimate.

3.4.1 Restricting the motion model

Although this work focuses on unconstrained 6DOF vehicles, the algorithm developed here may nonetheless be applied to vehicles with more robust odometry models, or more restricted motion capabilities. For example, an outdoor ground vehicle clearly cannot take off into the air, but nonetheless still operates in a non planar world (i.e., 2D pose estimation solutions may not be effective due to variations of the ground surface). The use of 6DOF pose estimation would be useful in these instances; it would, however, be fruitless to search for such robots flying far away from the ground. Given such a vehicle, the algorithm could simply make use of a less general motion model. This would prevent the estimate from suggesting poses that the sensor is not capable of achieving, i.e., the particles would remain at a constant height from the ground. This height depends on where the sensors for the vehicle are mounted; since the robot cannot leave the ground, the sensor's distance from the ground will likewise be relatively constant (where some variance in the height is expected, due for example to rough ground or the movement characteristics of the the robot). The particle filter should also consider this restriction when initializing the particle filter; the reduced search space can be taken into account when placing particles, limiting particle placement to locations within a certain range off the ground surface. For example, if the sensor is mounted 2 meters off the ground,

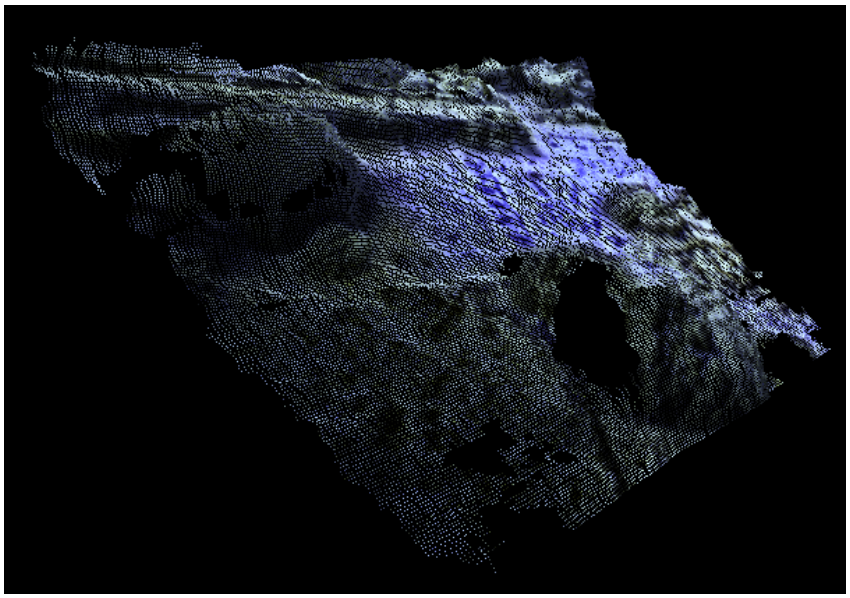


Figure 3.6: A sample 3D range cloud frame of a ship wreck, generated using the AQUA sensor. [38]

then particles could be restricted to a region of 2m $\pm$ ϵ (where ϵ represents the expected variance of the sensor's distance from the ground).

3.5 Range likelihood

A key sensing ability common to many robots is the ability to sense distance in some manner. Sonar and laser range sensors are common, and vision systems [38, 63] are becoming popular. Both vision and laser range finders are able to provide dense, detailed 3D information about regions visible to the robot. This range likelihood is used for each particle to estimate the likelihood of observing the output of such a sensor at the pose $\{x_i, \theta_i\}$, for the ith particle. The likelihood must be estimated efficiently, since the particles must span a much larger space (requiring more particles) compared to traditional 2D particle filters, and the range data contains far more data points. Typical particle filter methods attempt to generate an *expected* range scan, based on a given particle and a known map (outlined in Section 2.2.2). In 3D this amounts to ray-casting a scene, which is a well studied subject in computer graphics (see [35, 36]). Although efficient algorithms exist for 3D ray casting, current algorithms are insufficient to the task required for raycasting a frame for each particle of the particle filter. To see this, a critical bottleneck in the estimation of the range likelihood is the time required to generate a psuedo-scan for

each particle. For example, suppose that the range data z_t is composed of k independent data points z_t^k. Then the total likelihood is computed as:

$$p(z_t|x_t, \theta_t) = \prod_k p(z_t^k|x_t, \theta_t).$$

Given a particle $p_i = (x_i, \theta_i, w_i)$, the transformation $T^{x,\theta}$ may then be defined from the range finder's reference frame $\mathcal{N}$ to the world reference frame $\mathcal{B}$ defined by the map ($C_n^b[\theta_i]$ is specified in equation 2.10), as:

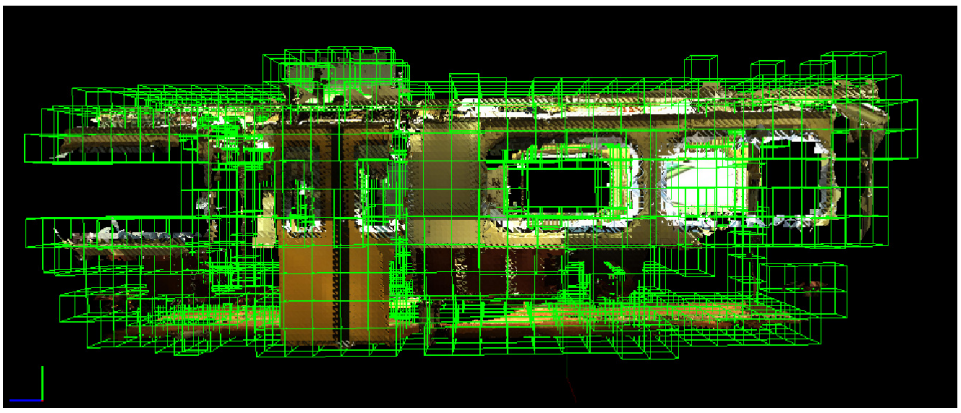
$$T^{(p_i)}(\vec{z}) = \vec{x_i} + C_n^b[\theta_i]\ \vec{z} \tag{3.28}$$

Then for each vector $T^{x,\theta}(z^k)$ originating at x_i, a raycast must be performed in the map. Even sparse sampling (every tenth pixel along both axis, i.e., 1% of the image) of a 320x240 pixel range frame would require 768 ray casts, resulting in nearly 40 million rays to be cast for a particle filter consisting of a modest 50,000 particles. Each ray requires multiple evaluations, continuing until an object is struck, or the maximum sensor range is reached.

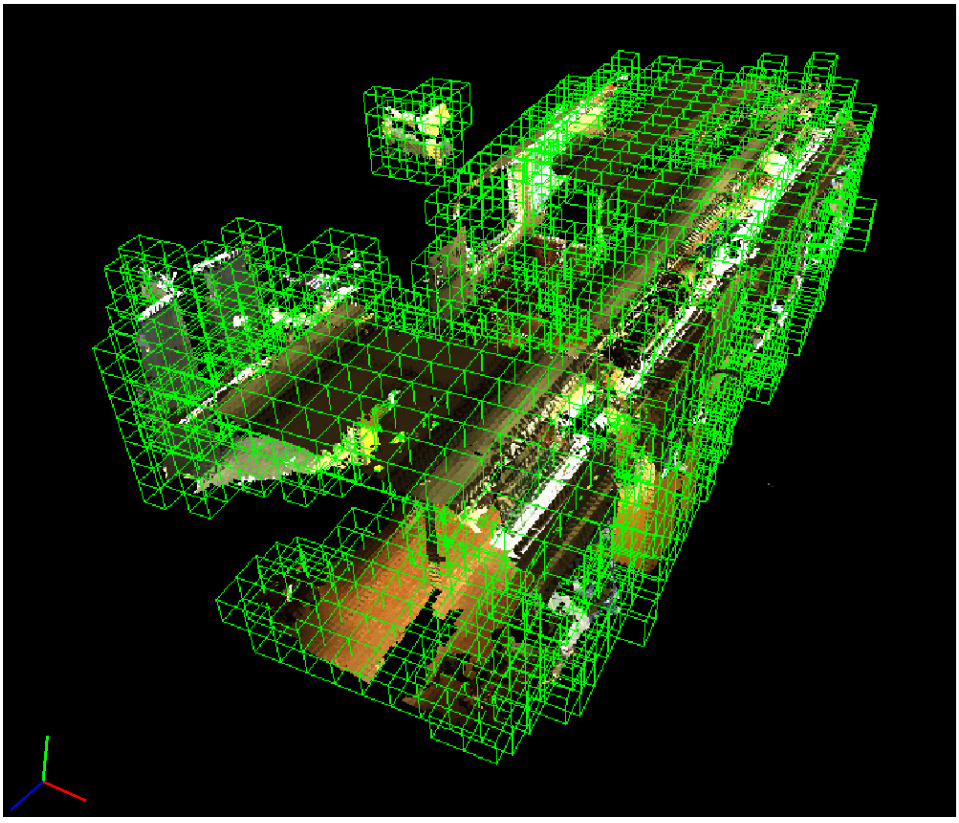
Rather than following the ray casting approach, an alternative that bypasses the raycast process is desired. One approach is the use of a lookup table that provides direct, precomputed values for the likelihood. The following section describes such an approach, based on results from the laser scan alignment literature [53].

3.5.1 Application of the NDT to pose estimation

As discussed in Section 2.4, the NDT has been shown to be effective in a number of scan alignment and map building applications. Here we use the NDT to estimate the likelihood of a range scan, without the use of minimization techniques. Note that recently this approach was independently proposed by [65]. Using the NDT in this manner provides a piece-wise PDF defined over the entire position space, which may be largely precomputed off-line. Every point detected by the robot, using its sensors, may be looked up directly (after transformation to the world coordinate frame), resulting in the estimated likelihood of observing that point. However, there is one clear shortcoming with using this method for pose estimation; certain classes of information are not well represented using this approach. To illustrate this, suppose two parallel walls, 1m apart, are present in an environment, and the robot detects a wall. Ray-casting techniques would realize that only the closest wall could be observed by the robot (see Figure 3.8 (a)), and would assign very low or zero likelihood to the more distant wall. On the other hand, since the NDT would directly look up the likelihood, it would not notice the nearer wall is present

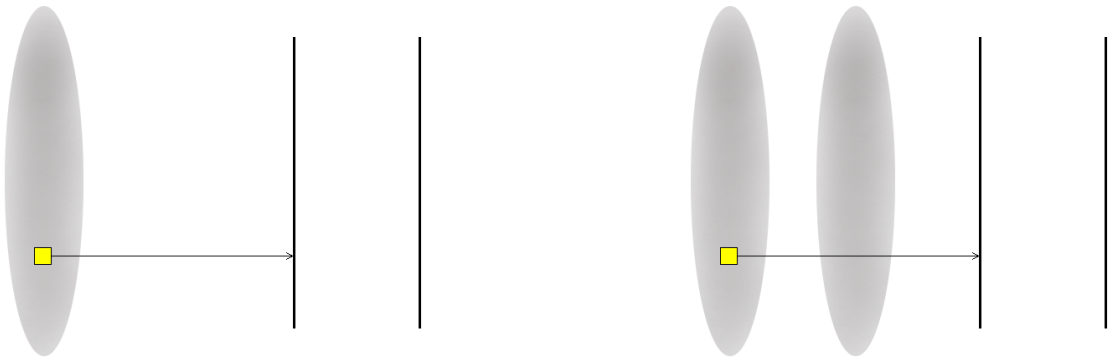


(a) Side view



(b) Birds-eye view

Figure 3.7: Octree structure used to store model data and likelihood lookup. Model comes from the C2SM project [68].



(a) Result using ray-casting techniques

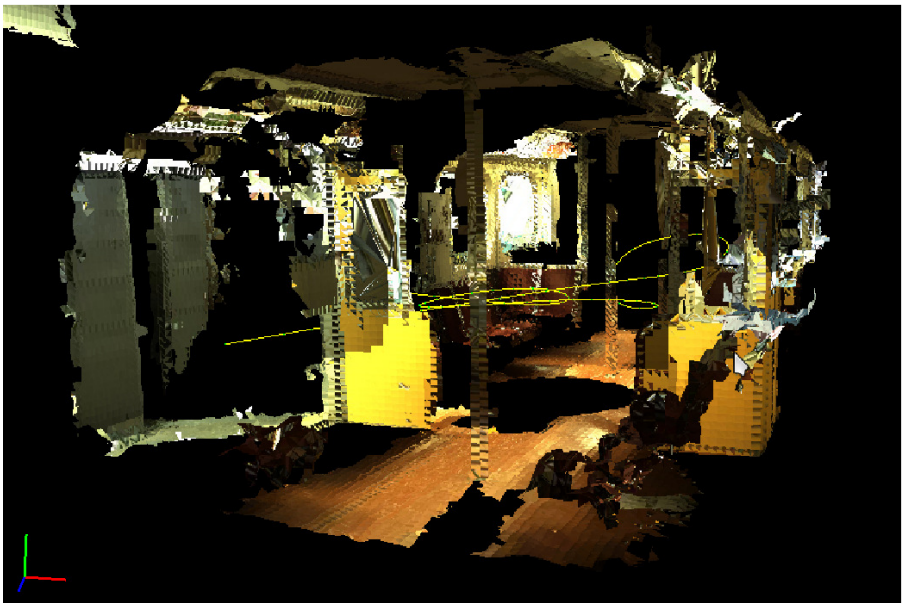
(b) Result using the NDT technique

Figure 3.8: The result of estimating the likelihood for detecting a range of 3m, using ray-casting techniques is shown in (a). The rightmost wall's presence is ignored, since it is not detectable from any pose on the robot side of the left wall. On the other hand, (b) shows two regions where the robot may be based on the NDT, resulting in one cloud for each wall.

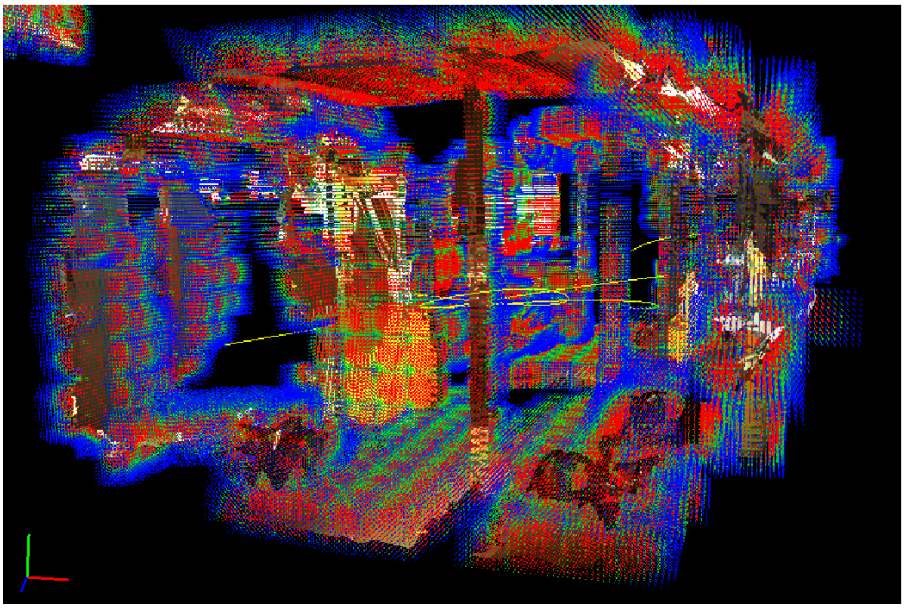
(see Figure 3.8 (b)). As such, both walls would be assigned equal probability. While this wrongly assigns importance to such map regions, experiments show that over time such inconsistencies tend to cancel each other out. This results in reducing the likelihood of particles placed incorrectly (to the point of being insignificant), making this incorrect importance manageable.

3.5.2 Evaluation of the range likelihood

Due to the number of particles required for effectively sampling the position space, the range likelihood must be evaluated as efficiently as possible. In Section 3.5.1 an approach for estimating the likelihood of a single range reading was outlined. In this section, the approach is optimized for multiple range readings in a scan. For every particle, the range data in a scan must be evaluated in the reference frame defined by a particle. Given a range scan z_t composed of an $m \times n$ 2D grid of 3D range points $z_t^{m,n}$, the likelihood of



(a) Subway model on which an NDT is defined.



(b) 3D NDT, red indicates high likelihood, while blue denotes low probability.

Figure 3.9: The 3D NDT used to define a likelihood on a model. The NDT is used to define the likelihood of the range sensor indicating a "hit" at any location in the map. Model data taken from the C2SM project [68].

the entire scan can then be estimated as

$$
\begin{aligned}
p(z_t|x_t,\theta_t) &= \prod_{m,n} p(z_t^{m,n}|x_t,\theta_t) \\
&\approx \prod_{m,n} P_{NDT}[T^{(p_i)}(z_t^{m,n})] \\
\text{Let } z^{\sim} &= T^{(p_i)}(z_t^{m,n}) \\
&= \prod_{m,n} \frac{1}{(2\pi)^{1.5}\sqrt{|\Sigma|}} exp\left(-\frac{(z^{\sim}-\vec{q})^t\Sigma^{-1}(z^{\sim}-\vec{q})}{2}\right). \\
\text{Let } \sigma(z^{\sim}) &= -\frac{(z^{\sim}-\vec{q})^t\Sigma^{-1}(z^{\sim}-\vec{q})}{2} \\
\text{and } C_i &= \frac{1}{(2\pi)^{1.5}\sqrt{|\Sigma|}}, \text{ constant per cell}
\end{aligned}
$$

$$
\begin{aligned}
&= \prod_{m,n} C_i \ \exp\sigma(z^{\sim}) \\
&= C_0 \ \exp\sigma(z_0^{\sim}) \ ... \ C_{mn} \ \exp\sigma(z_{mn}^{\sim}) \\
&= (C_0 \ \times ... \times \ C_{mn}) \ \exp(\sigma(z_0^{\sim}) + ... + \sigma(z_{mn}^{\sim})) \\
&= \exp(\ln(C_{0,0}) + ... + \ln(C_{m,n}) + \sigma(z_{0,0}^{\sim}) + ... + \sigma(z_{m,n}^{\sim})) \\
&= \exp(\sum_i \ln(C_i) + \sum_i \sigma(z_i^{\sim})). \qquad (3.29)
\end{aligned}
$$

The representation in this work stores $\ln(C)$ instead of C, since C is constant for each cell, and can therefore be precomputed. The likelihood may now be computed using only a single exponential call, which results in a considerable savings in computational cost. For a range scan with 100 range points, the constant C_i and the computed value z_i for each point are looked up and added together. The final likelihood for the range scan is then computed by taking the exponential of that value.

In practice, when $p(z_t|x_t,\theta_t)$ is computed for each particle, the weights for likely poses are extremely large, while unlikely poses have vanishingly small weights. This can result in weights that are skewed such that resampling will never select any less likely particles. This generally leads to a single particle surviving the resampling, causing filter failure. Instead, in this work the weights are evaluated on a logarithmic scale as $\ln(p(z_t|x_t,\theta_t))$. This amounts to not computing the exponential in Equation 3.29 in the first place, i.e.,

$$p(z_t|x_t,\theta_t) = \sum_i \ln(C_i) + \sum_i \sigma(z_i^{\sim}).$$

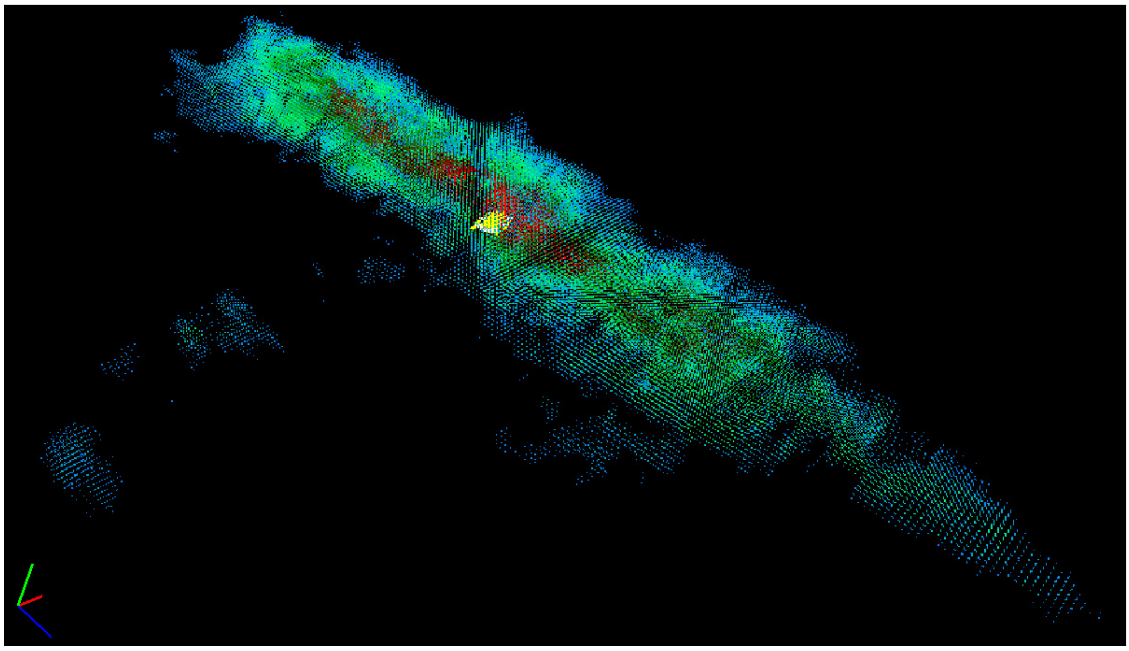


Figure 3.10: A sampled instance of $p(z_t|x_t, \theta_t)$ for a specific range image z_t, over all x_t, given an approximation of θ_t, on a subway model from [68]. The likelihood is indicated, from high to low as red, green and blue; black indicates zero likelihood. This image was generated by evaluating $p(z_t|x_t, \theta_t)$ at 2cm intervals, using the orientation estimate from the Kalman filter. When particles are used instead, the noisy motion model used ensures that even particles placed near the edges of this distribution will quickly converge to the high likelihood center (indicated in red).

Finally, depending on the available computational resources, the range scan may be sampled at reduced resolution; this work uses 10% of the available range points on each axis to balance computational requirements and data loss. Clearly, increasing the number of range points used should improve the accuracy of the estimate, and conversely, the limited sampling used here likely results in slower convergence. Exploring the use of a filter that varies the number of samples dynamically (depending on the number of particles in use or on some computational time constraint) would be of interest in future work.

3.6 Filter applied to a 3D dataset

This section demonstrates the result of applying the filter to a sequence of simulated sensor measurements collected from a dataset for the C2SM project [68]. The filter has no prior knowledge of the robot's pose, as demonstrated by the uniform distribution

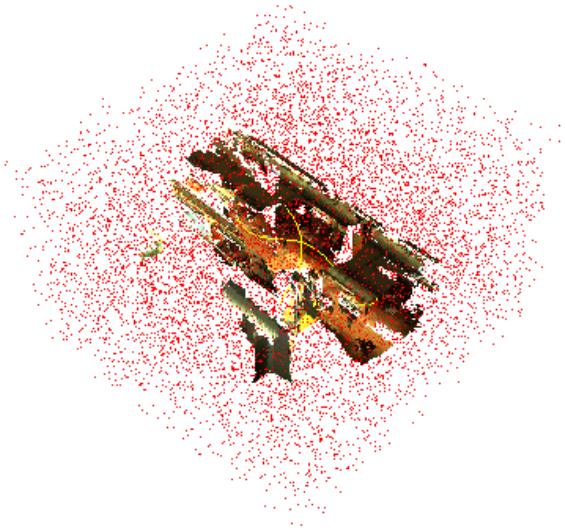
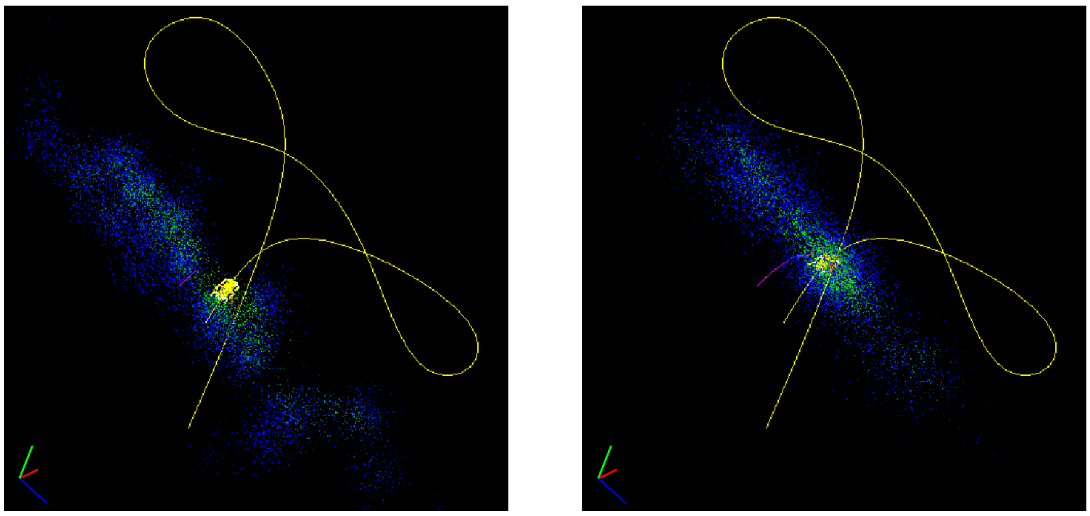


Figure 3.11: A uniform distribution of particles across the position search space. This example uses 10,000 particles for the search space.

of particles in the position space shown in Figure 3.11. Note that only position is encoded in the following figures. The EKF for orientation estimation is initialized using experimentally obtained values for $\sigma_{g,a,m}$, taken from [59]), shown in Table 3.3. After incorporating the sensor measurements, the majority of particles' weights are zero, as most of the search space has negligible likelihood. Resampling the particle filter results in Figure 3.12(a). Note that, again, areas of high likelihood are coloured red, then green, while low likelihood is indicated by blue; areas with zero likelihood are indicated by black, i.e., the absence of particles. In all figures, the yellow line indicates the ground truth robot path[3]. As expected, the particles initially form a stream along the length of the subway car, since the observed portion of the subway car is not unique. Upon incorporating another set of measurements, the filter begins to converge on the correct pose estimate, as shown in figure 3.12(b), indicated by the the green particles clustered near the robot.

Figure 3.13(a) shows the result of resampling after time $t = 7$ seconds. Although some particles remain away from the expected pose, most have clustered towards the correct result. Figures 3.13(b-d) show that the global pose estimate has converged to

[3] The plotted "true robot path" is based on the egomotion estimate in the case of AQUA sensor data, or simulated egomotion data in the other datasets. Error plots are relative to this ground truth path; although the true path would be preferable, it is difficult to obtain for devices such as the AQUA underwater robot.



(a) Belief after a single measurement.

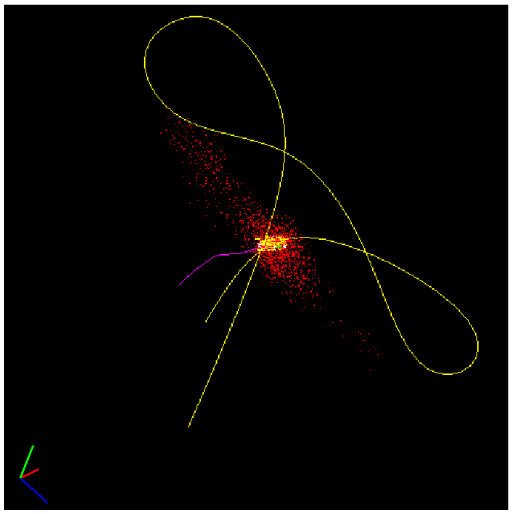
(b) Belief after a second update pass.

Figure 3.12: (a) shows the result of incorporating one measurement and resampling, while (b) has incorporated a second measurement. In (b), the particles have begun clustering around the true robot pose. The yellow path indicates the ground truth regarding the robot's path, while the purple line forming in (b) indicates the particle cloud mean estimate approximating the ground truth.

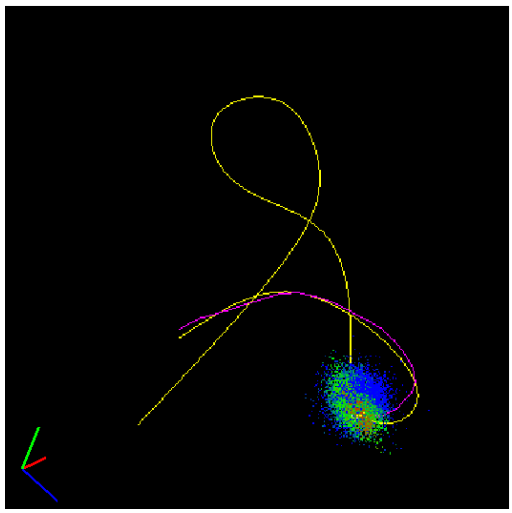
σ_g^2	σ_a^2	σ_m^2	$\sigma_{w_a}^2$	$\sigma_{w_m}^2$
0.1 rad/s	0.1 m/s^2	2 $mGauss$	0.5 m/s^2	50 $mGauss$

Table 3.3: Experimental filter parameter values, used for IMU simulations in this work. Taken from [59].

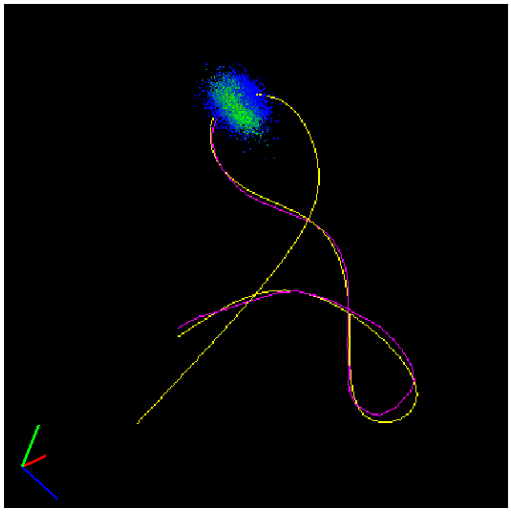
a single estimate. The visible color layers of the particle cloud in (b-d) is a result of available measurements. In Figure 3.13(b), the range sensor is able to provide a good restriction for the elevation of the device, although the lateral position is less accurately known. Figure 3.13(d) on the other hand contains sufficient information for a left to right position fix, but has reduced accuracy for the distance. The final error in estimation for this example is $\sim$15cm between the particle cloud's weighted average and the known correct result. Figure 3.14 shows the localization error, plotted as a function of time.



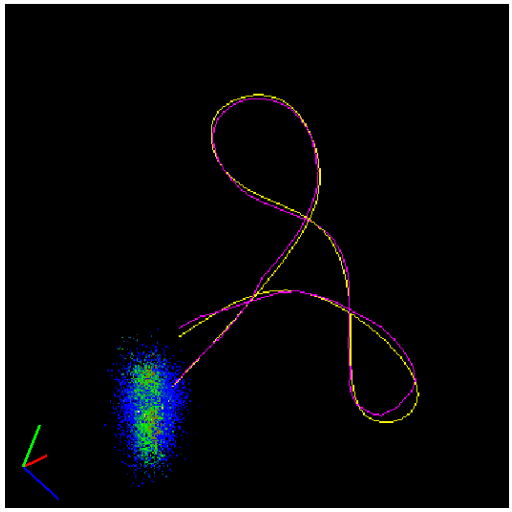
(a) At time $t = 7$.



(b) At time $t = 29$.



(c) At time $t = 57$



(d) At time $t = 95$.

Figure 3.13: Progressive evolution of the pose estimate. See legend of Figure 3.12 for details.

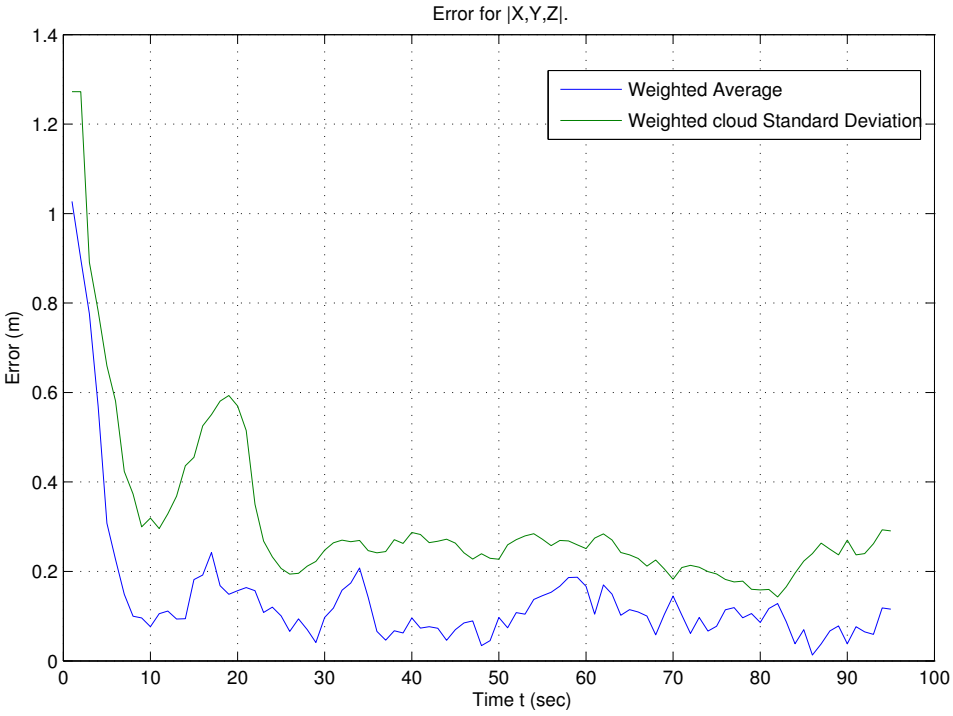


Figure 3.14: Error plot for localization shown in Figures 3.12 and 3.13. The plot ignores data at time $t = 0$, since it would amount to the distance between the map center and the correct robot pose. The error shown is the distance between the robot pose and the particle cloud mean. The cloud standard deviation shows the "spread" of the particle cloud, where more variance would indicate less certainty. Sudden spikes in the plot are attributed to missing data in the range sensor. This results in little information correcting the belief, leading to the motion model expanding the size of the particle cloud. When the range sensor is able to provide more detail again, the estimate converges again.

3.7 Summary

This section introduced a filter which is able to estimate the 6DOF pose of an unconstrained vehicle operating in a 3D environment. Using this filter, a simple example is provided to show how the filter may be applied to a simple, discrete estimation task. Following the derivation, we develop an efficient approach for estimating the 3D likelihoods required for implementing the filter. Finally, an example of the filter applied to a real dataset is covered. The next section covers results based on various datasets, both virtual and real.

Chapter 4 Simulation results

The filter developed in the previous chapter has been applied to several datasets to evaluate its performance; the results of those experiments are discussed in this chapter. The datasets used here vary from game models (for controlled testing) to real world datasets including a LIDAR[4] map and vision-based datasets from the AQUA and C2SM projects. The datasets used for testing the pose estimation algorithm are plotted in Figure 4.1. In order to provide a controlled evaluation of the approach, the algorithm is evaluated off-line on collected datasets while motion through these datasets is simulated (with the exception of the barge fragment). This is a reasonable approach given the complexity of collecting the data and the difficulty in obtaining ground truth estimates of the motion of a 6DOF vehicle. The use of simulated data for the robot's motion through the space requires simulation of the sensor and agent (and their error models). Details of these error models are provided below.

This chapter is laid out as follows; Section 4.1 describes the experimental conditions used. Section 4.2 tests the localization filter using a constant 10,000 particles, demonstrating the basic functionality of the filter. Large maps initially require a greater number of particles (compared to smaller maps) in order to cover the position space sufficiently;

[4]This LIDAR dataset kindly provided by John Meneely at Queens University (Belfast, N.Ireland), and meshed using the ball-pivot algorithm [2] implemented by Meshlab [13].

Map	Vertices	Polygons	Size (m^3)
AQUA project [38] barge model.	108612	36204	$3.6 \times 2.0 \times 2.2$.
A detailed subway car model [68].	159174	53058	$7.7 \times 4.3 \times 7.3$.
Small game map from YoFrankie [57].	55701	18567	$11.0 \times 3.8 \times 10.6$.
Four subway cars from [68].	619188	206396	$27.4 \times 3.4 \times 26.1$.
Large game map from YoFrankie [57].	157692	52564	$30.4 \times 14.2 \times 28.7$.
A LIDAR dataset of a kiln.	457464	152488	$53.1 \times 15.4 \times 75.3$.

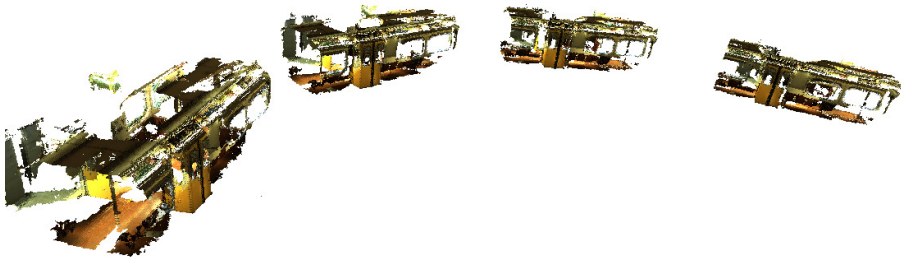
Table 4.1: Various environment models used for experiments.

for this reason, the KLD algorithm [23] is applied to the particle component of the filter. The KLD algorithm varies the number of particles as needed, depending on the distribution of particles in the search space (for details refer to Section 2.2.2). The results of using the KLD algorithm are discussed in Section 4.3. Finally, Section 4.4 evaluates the performance of the filter compared to a naïve implementation of a 6DOF particle filter (which estimates all six parameters using a particle filter).

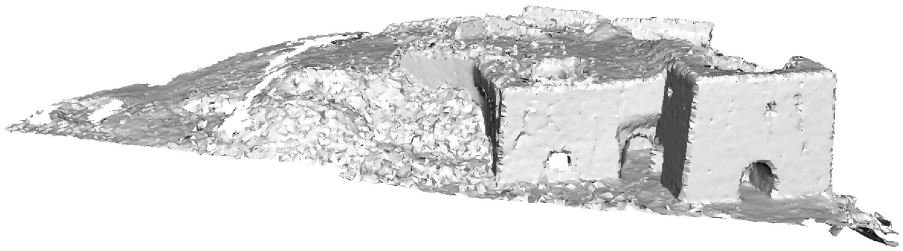
4.1 Experiment conditions

The environment maps used for experiments reported in this chapter are real (with the exception of the two game maps), and were collected using various vision and laser technologies. With the exception of the barge fragment, which was collected using the AQUASensor, all range data (hypothetically) observed by the robot using its stereo sensor was simulated to emulate the AQUASensor. The characteristics of the AQUASensor were duplicated based on real data from the AQUASensor; comparison of real and simulated data for the barge fragment showed very similar data. Additional noise and error for range image z_t (composed of range points z_t^i) is simulated by adding random, uniformly distributed noise in the range $[-0.1\,|z_t^i|, 0.1\,|z_t^i|]$ to each range point in the scan. As an additional corruption, randomly positioned occlusions are included in each range image, with each occlusion covering 10% of the width and height of the vision frame. Three occlusions are randomly included at each time step, with probability $0.75, 0.50$ and 0.25, respectively; occlusions are positioned somewhere between the camera and the observed distance. These occlusions are included to simulate random failures for portions of the frame, caused for example, by fish.

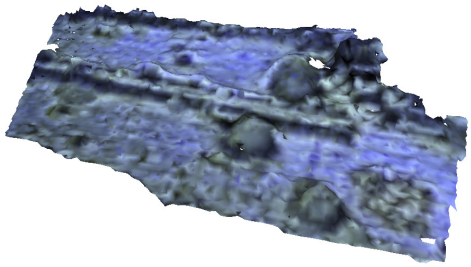
The motion of the robot through the models is simulated to emulate the speed of the AQUA robot. As such, the robot's speed averages 30cm/second, ensuring that sufficient overlap occurs between successive video frames. Since most of the robot motion is simulated (again, with the exception of the barge fragment), the egomotion must also be simulated. A known ground truth (generated in the map using a 3D modeling package) is provided to the simulator, outlining the robot's path through the environment. Based on this path, changes in position and orientation (relative to the robot's previous position along the path) are computed in the robot's coordinate frame, resulting in the control $\bar{u}_t$. The control $\bar{u}_t$ is then converted to the simulated, egomotion-estimated control u_t by adding randomly distributed noise in the range $[-0.2\,|\bar{u}_t|, 0.2\,|\bar{u}_t|\,]$, where $|\bar{u}_t|$ is the magnitude of the vector. Finally, all IMU data for the experiments is simulated by using the IMU model specified in [59]. Gaussian noise is added with parameters according to the experiments in [59], listed in Table 3.3.



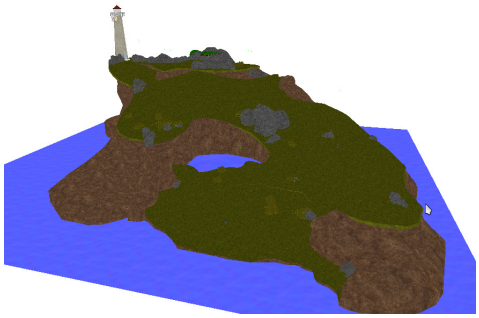
(a) Four copies of (f) (from [68]) in a simulated subway car setup



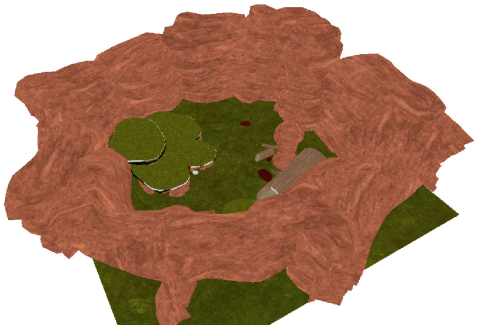
(b) A LIDAR based model of a kiln in Scotland [49]



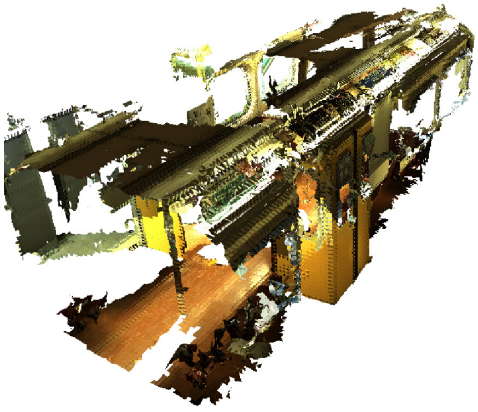
(c) Barge fragment from AQUA project [38]



(d) A game world from YoFrankie [57]



(e) A game world from YoFrankie [57]



(f) Subway model from C2SM project [68]

Figure 4.1: The environment models used to evaluate the localization algorithm.

In order to simplify the comparison of results across the different models, the experiments in this chapter were all performed with a common set of parameters.

- Each experiment is performed ten times, with all sensor noise re-initialized randomly for each experiment. The robot's correct path remains constant; this is to account for the stochastic nature of the localization algorithm.

- The particle filter for each experiment is initialized randomly using a uniform distribution, with particles covering a volume 2m larger (on each axis) than the model's volume (i.e., the subway model is initialized with a volume of $[7.7+2] \times [4.3+2] \times [7.3+2]m$). This ensures that the robot's position is included in the initial distribution.

- The motion model for the particle filter adds additive noise independently to each axis from a normal distribution with the variance set to $0.5\ |u_t|$. This results in tightly clustered particles expanding into a "bubble" around the estimated motion, i.e., $u_t \pm 0.5\ |u_t|$. Such a liberal model covers potential errors in the ego motion and also allows particles to move out of local maxima.

- The sampling rate for the IMU (and the corresponding extended Kalman filter update) is 100hz; by comparison, the inertiaCube3 [39] has an update rate of 180hz. Due to the greater computational cost associated with updating the particle filter, it is updated at a reduced rate in these experiments, once per second. This rate can certainly be increased, depending on the number of particles employed by the filter. Since the algorithm is run off-line, this update process results in the Kalman filter being updated 100 times, followed by a particle filter update.

- The unit t in the graphs refers to the number of seconds since the robot started moving. At each second, data is collected and time stamped; the particle filter component is then updated with this data.

- To allow the Kalman filter estimating the orientation of the vehicle to converge, it is assumed that the robot is stationary for 10 seconds prior to moving. This allows the sensors to settle, allowing the Kalman filter to converge to a reasonable initial estimate. The position estimate is not computed during this time, as it would be meaningless without a reasonable orientation estimate. These 10 seconds occur prior to time $t = 1$ in the figures.

- Since the range scan is not necessarily complete, i.e., portions may not contain any data, the filter only uses range data in the range (0, Max_sensor_range]. The number of range samples used at each update step is indicated in the figures.

- All graphs represent results starting at time $t = 1$, after integrating the control u_0, sensor measurements z_1, i_1 and resampling once. The data prior to the belief at $t = 1$ (i.e., the initial random particle filter distributed over the space uniformly) would simply clutter the graphs and is therefore omitted.

- The particle cloud mean μ_t is computed according to

$$\mu_t = \sum_i pos_i \; w_i,$$

  where pos_i is the position vector and w_i is the scalar weight associated with the ith particle.

This chapter focuses on the position estimate results, on the assumption that the orientation is reasonably estimated by some other filter (i.e., the choice of orientation estimator is unimportant, as long as it is able to reasonably estimate the device's orientation). If the orientation estimate is faulty, the position estimate is very unlikely to converge correctly.

4.2 A constant 10,000 particles

This section describes experiments for validating the algorithm. In order to avoid tailoring the localization process to the environment, a baseline evaluation was performed with a constant (10,000) particles used for localization. Examples of existing particle filters used for 2D pose estimation sometimes start with 40,000 particles (e.g., [23]). This suggests a reasonable number of particles to use; instead only a quarter were used, as both an added challenge and, initially, to reduce the time required for running experiments on the datasets. The smaller models (the subway, mini-level and barge models) tend to converge rapidly to the correct position estimate given this sampling. This is expected, as they are small and do not require a large number of particles to cover the space sufficiently, compared to the larger models. For example, Figure 4.2 (right column) shows that the filter converges almost immediately to the correct position estimate. Further tests shows that, for example, the small game map converges reliably using 750 particles, see Figure 4.24. Although particles are initially distributed uniformly throughout the model, the evaluation of the likelihood $p(z_t|x_t, \theta_t)$ is zero for large portions of the map. This forces the particles into a relatively small subset of the map during the first resampling of the particle cloud, resulting in the observed rapid convergence (see Figure 4.3 for a graphical illustration).

The larger models (lighthouse, kiln and large subway models), illustrate the result of using fewer, or on occasion too few, particles, i.e., under-sampling the position space. Although the filter sometimes converges, it does not do so consistently, due to a lack of particles (see Figure 4.2, left column). Although a particle may be placed with some luck in a location where $p(z_t|x_t, \theta_t)$ evaluates to a high likelihood, this is not always the case. Too many particles may end up in locations with near zero likelihood. For example, the large subway model (see Figure 4.8) sometimes converges incorrectly, particularly along the x and z axis. The y axis indicates height in this case, and since all four subway cars are at the same height, converging to any of the subway cars will results in the same height estimate. The lighthouse model (Figure 4.4) on the other hand, has a large variance along all three axis, i.e., the map is very varied in its height. This tends to result in particles converging incorrectly, which eventually results in a sum likelihood of zero over all particles. Although detecting failure is an independent area of study (known as the kidnapped robot problem), in this work it is trivially (and not robustly), detected by noting when the normalizer equals zero. When the particle filter fails, it is automatically reinitialized as a uniform distribution. This pushes the particle cloud deviation to a maximum, which is evident in the figures. This reinitialization results in the filter converging on the 3rd try, shown in figure 4.2. Figures 4.4 to 4.11 depict the results with 10,000 particles for the various datasets.

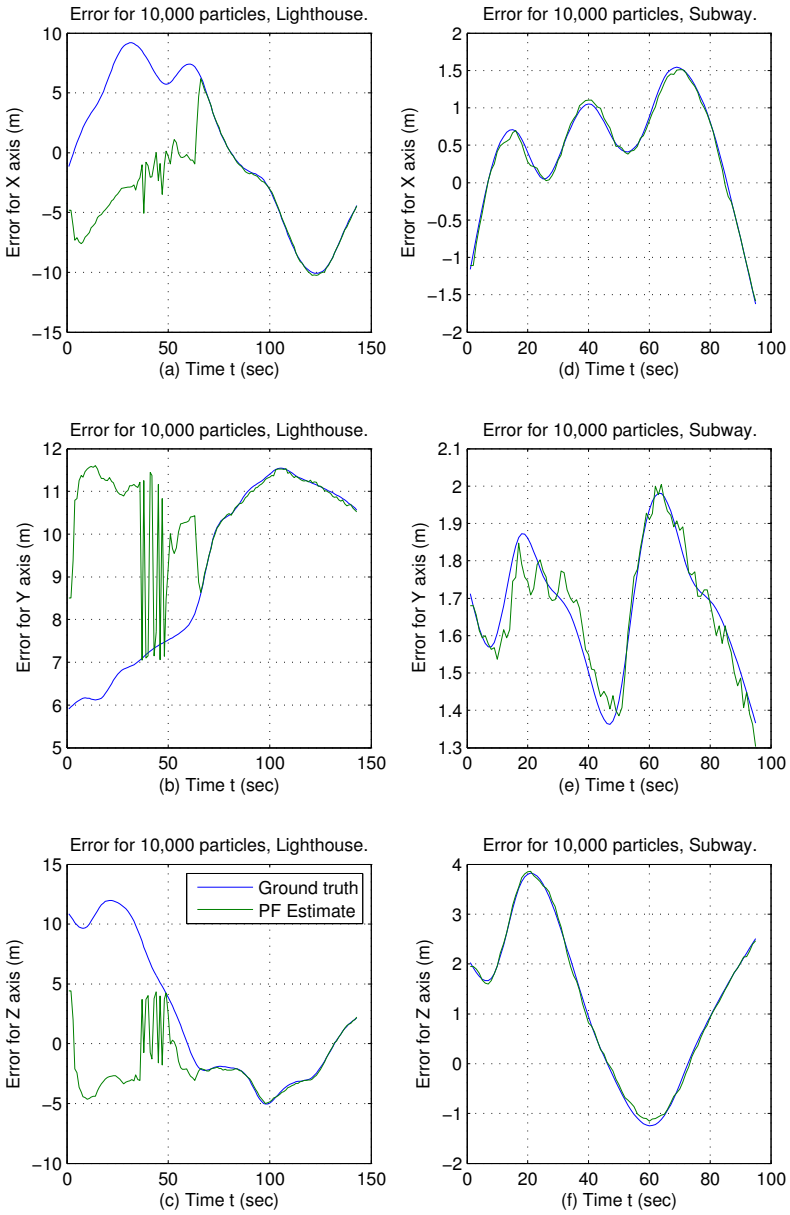
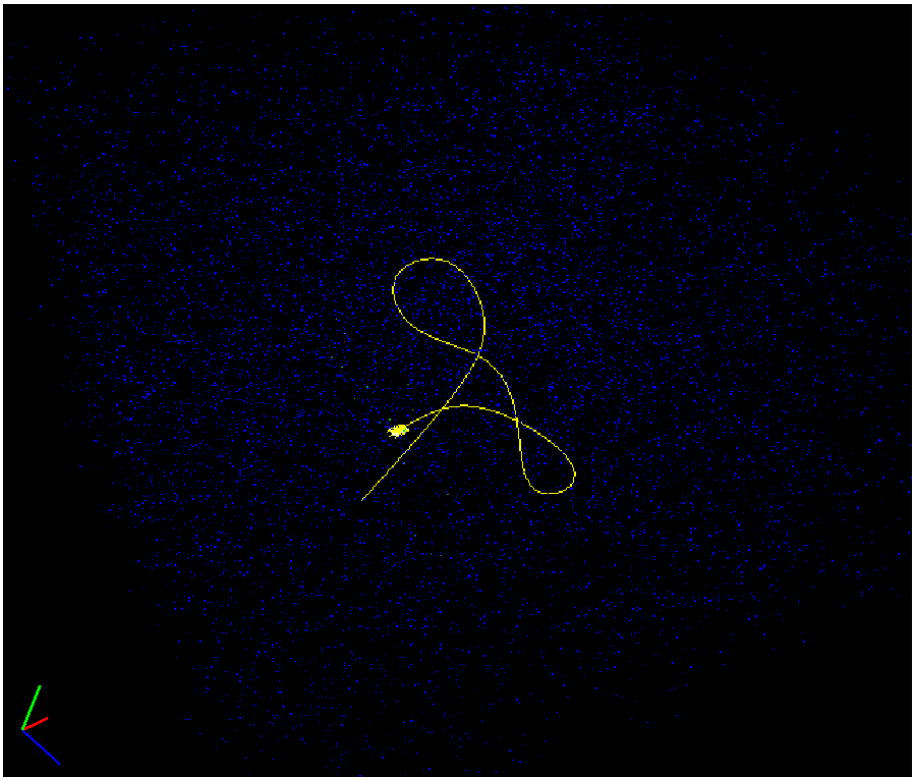
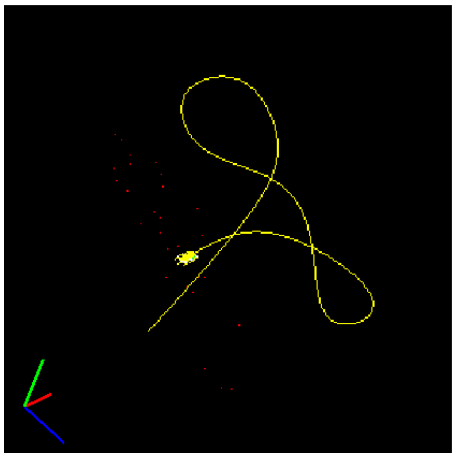


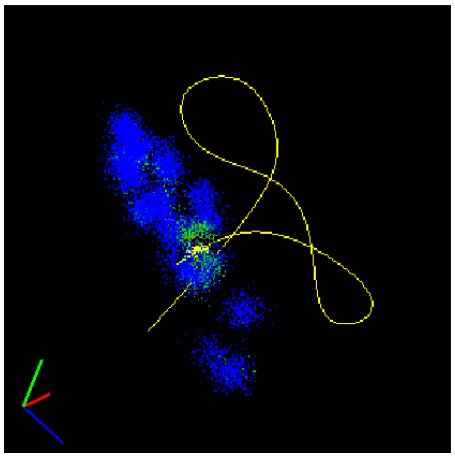
Figure 4.2: This figure shows the divergence of the lighthouse model (left column), and the convergence of the subway model (right column). Note that the lighthouse model fails repeatedly (three times), eventually reinitializing some of the particles into the correct space (which occurs around $t = 70$). The blue line indicates the true position, while the green line indicates the estimated position.



(a) Time $t = 0$. Nearly all particles have low likelihood (blue).



(b) Time $t = 0$, after resampling.



(c) At time $t = 1$.

Figure 4.3: Illustrating the initial rapid convergence of the filter. (a) shows the initial particle distribution, after the weights are updated using the latest measurements, but prior to resampling. (b) shows the result of resampling, which only retains non-zero particles. (c) shows the filter after the sensor measurements at time $t = 1$ have been incorporated.

Sudden reduction of the particle filter error indicates that an (incorrect) group of particles has been eliminated during a resample and is no longer influencing the particle cloud mean. As discussed in Section 2.1, the particle cloud mean (or the minimum mean square error, MMSE) is not usually an optimal estimator of the belief. A MAP estimate (or suitable alternative) is a better approach to improving this estimate error. Such methods may certainly be applied to this filter, but for simplicity this work makes use of the admittedly flawed mean. This means that initial errors could be reduced through the use of MAP, but once the filter has converged to a single cloud the estimate is robust. For verification of this filter the simple mean approach is sufficient, but if deployed on a robot a more robust approach should be employed.

Although the path recovered by the filter is not the true path, it closely approximates it. This is due to the particle filter sampling an alternative distribution (the observation likelihood), and not the desired position likelihood. Although the particles are weighed in an attempt to correct the difference, the filter cannot do so perfectly; the observation likelihood computed for the filter is merely an approximation of the actual observation likelihood.

The following pages contain the figures for each of the "10,000 particle" experiments performed. Each plot shows the results for ten random executions of the filter for the same robot track, each one using randomly generated controls. Sub-figures (a-c) display the estimation error along each axis, while (d) shows the distance between the known correct position and the estimated robot position. Figure (e) shows the update time required per particle filter update, which varies according to the number of range points used (f) and the number of particles in use at a particular time (h). The uncertainty of the particle filter is represented in (g), by the weighted standard deviation, computed as

$$\sqrt{\sum_i w_i(\mu - x_i)^2} \text{ as opposed to the typical } \sqrt{\frac{1}{N}\sum_i (\mu - x_i)^2}.$$

Here μ is the particle cloud mean, x_i is the position component of the pose described by the ith particle , and w_i is the particle's normalized weight such that $\sum w_i = 1$. This metric is used to indicate the "certainty" with which the filter believes the estimated pose, by showing how spatially distributed the particles are.

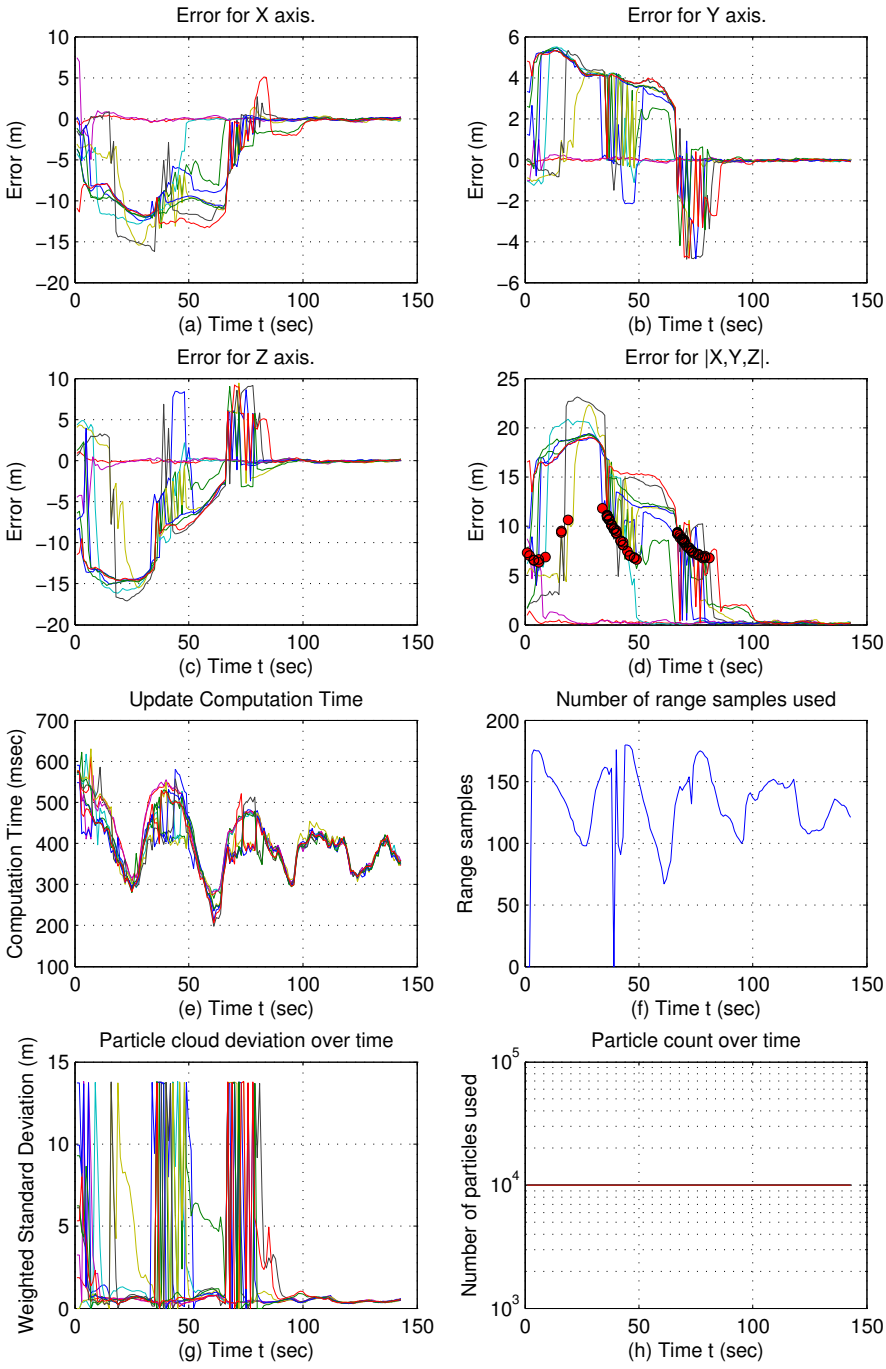


Figure 4.4: 10,000 Particles, Lighthouse. Each individual, randomly seeded experiment is plotted using a separate colour. The red dots indicate when the particle filter re-initializes, due to all particles having zero likelihood.

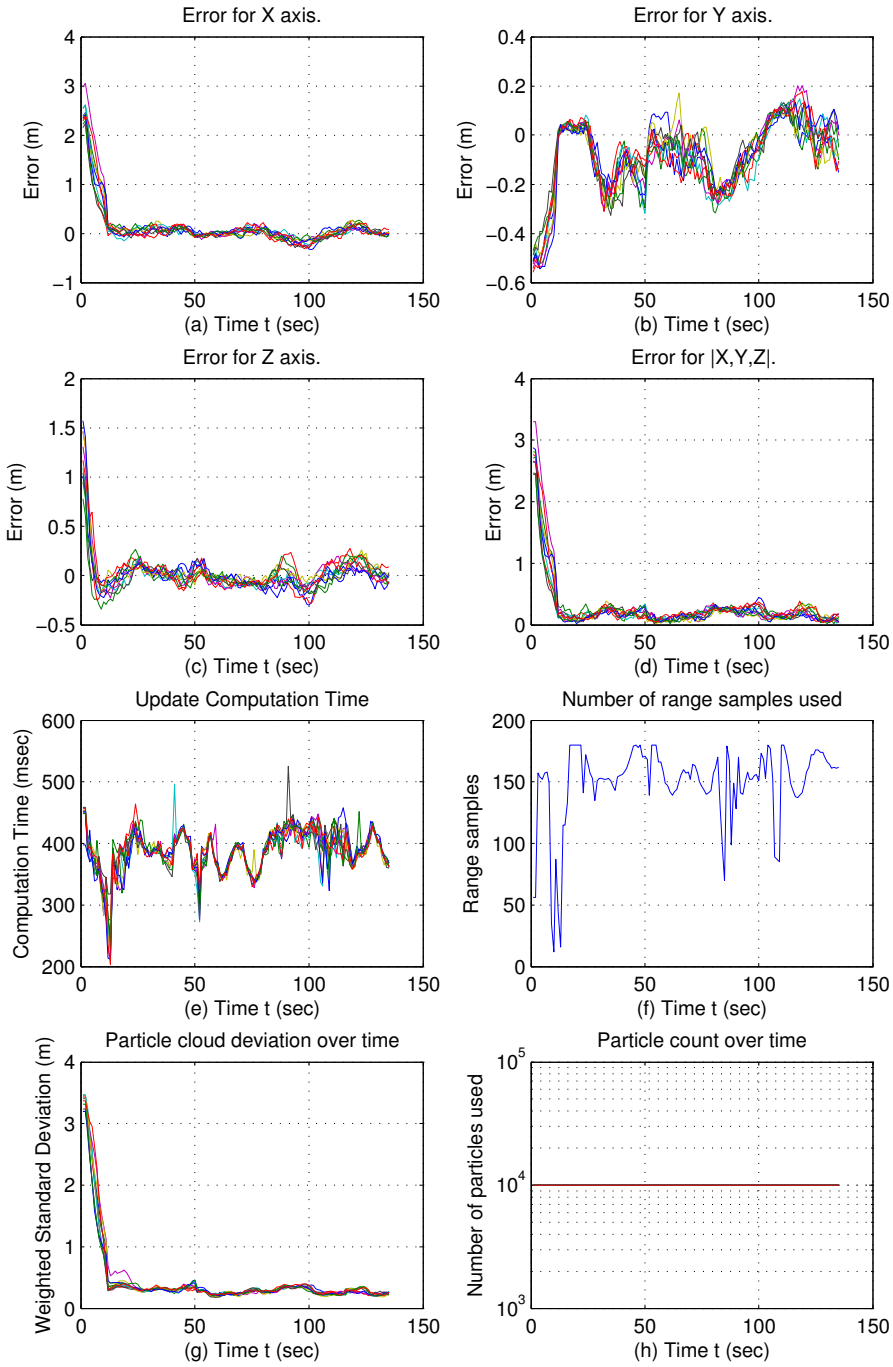
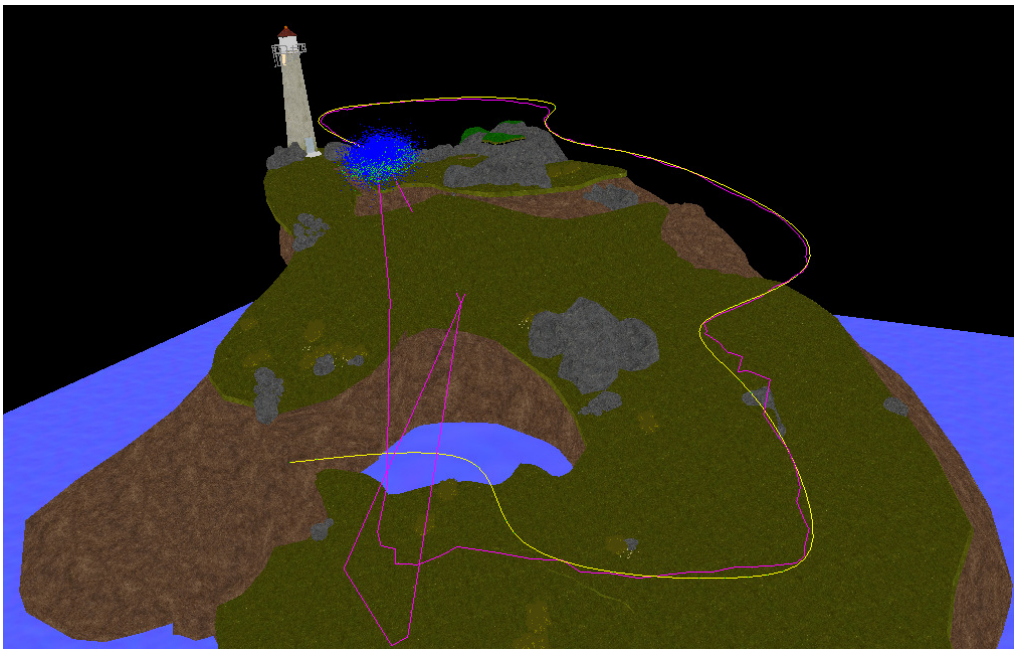


Figure 4.5: 10,000 Particles, Mini Level.



(a) Lighthouse model

(b) Mini Level

Figure 4.6: The yellow path indicates the ground truth, while the purple path shows the filter's best estimate.

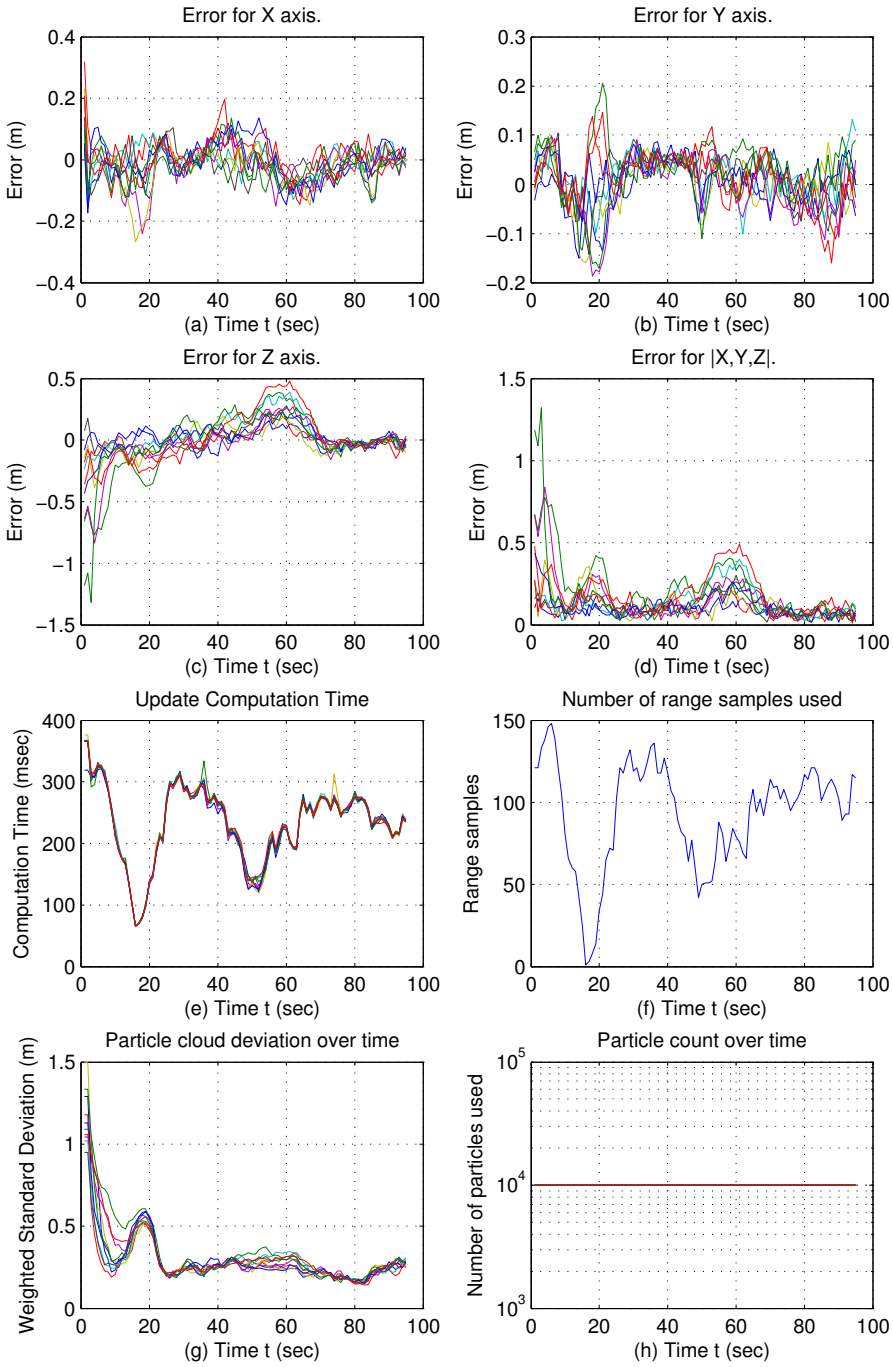


Figure 4.7: 10,000 Particles, Subway.

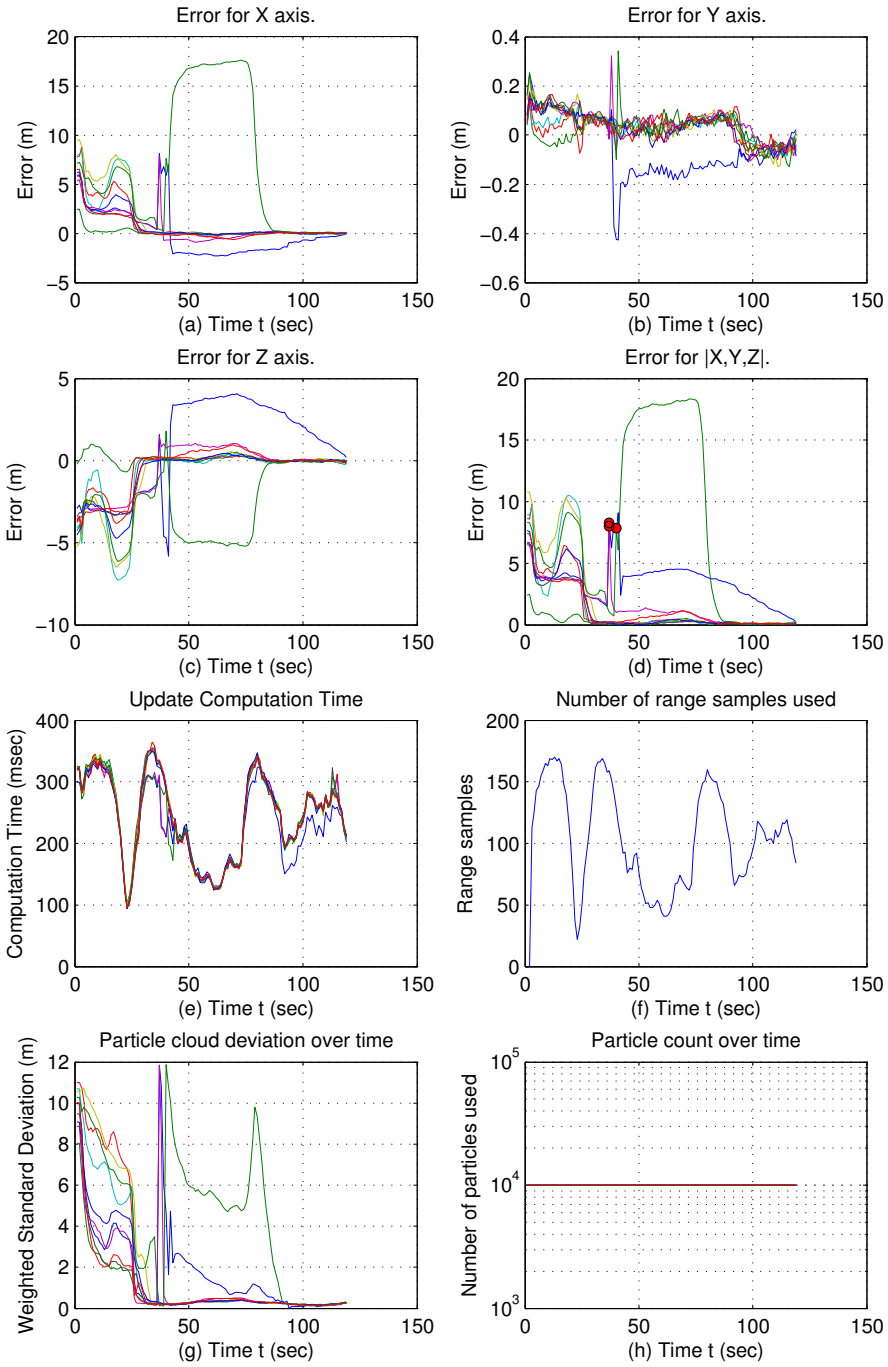
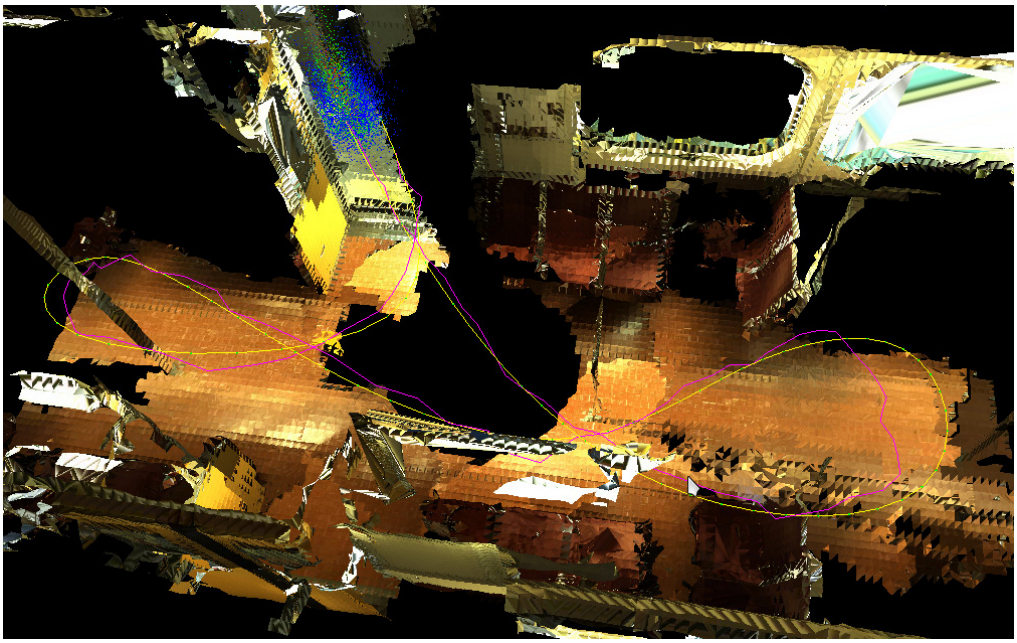
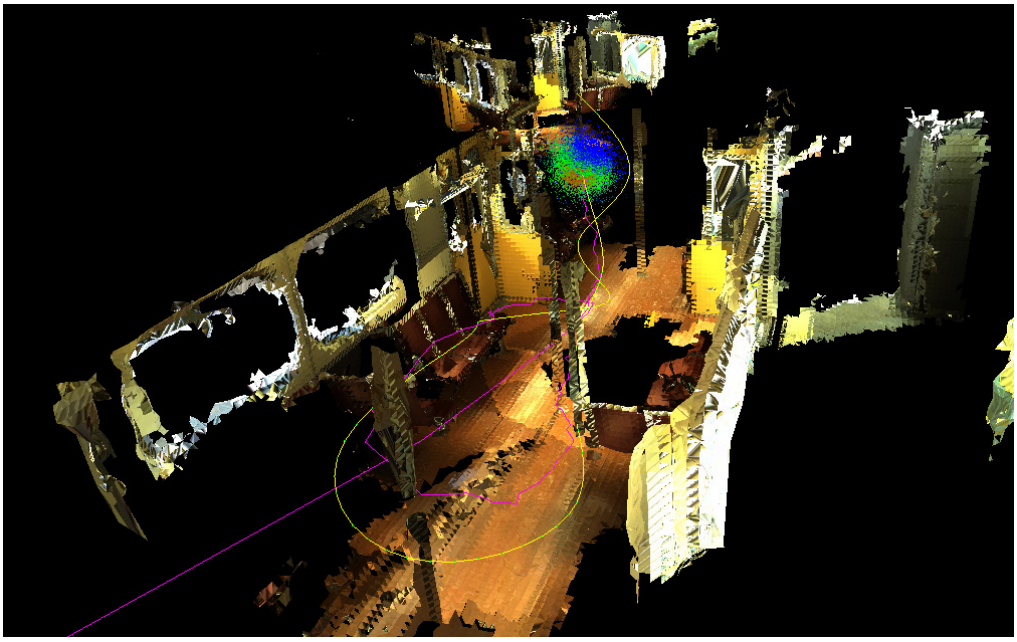


Figure 4.8: 10,000 Particles, Subway Large.



(a) Subway model



(b) Large Subway model

Figure 4.9: The yellow path indicates the ground truth, while the purple path shows the filter's best estimate. The purple line coming from the lower left in (b) is due to a simple mean being used for the best estimate.

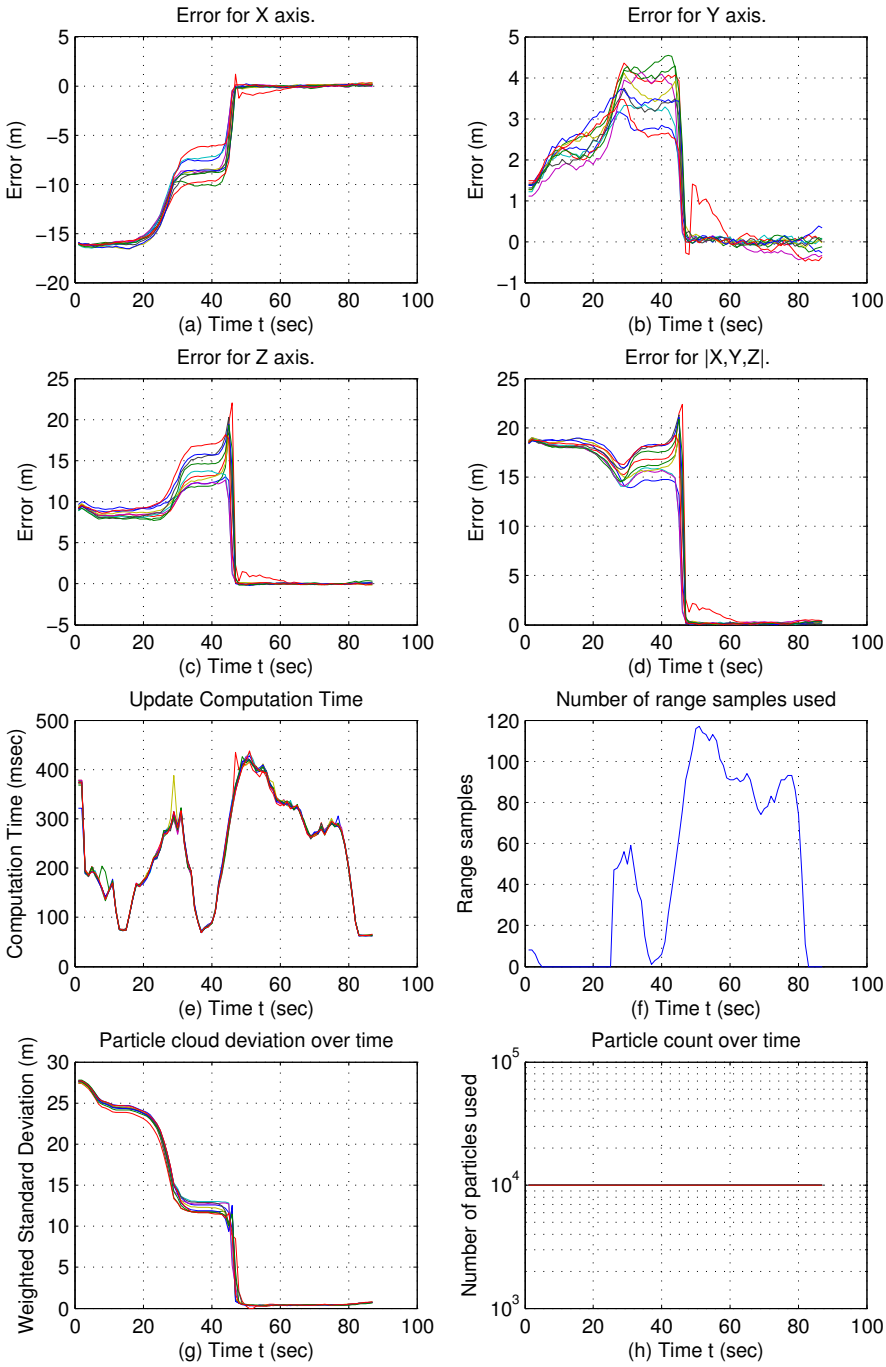


Figure 4.10: 10,000 Particles, Kiln.

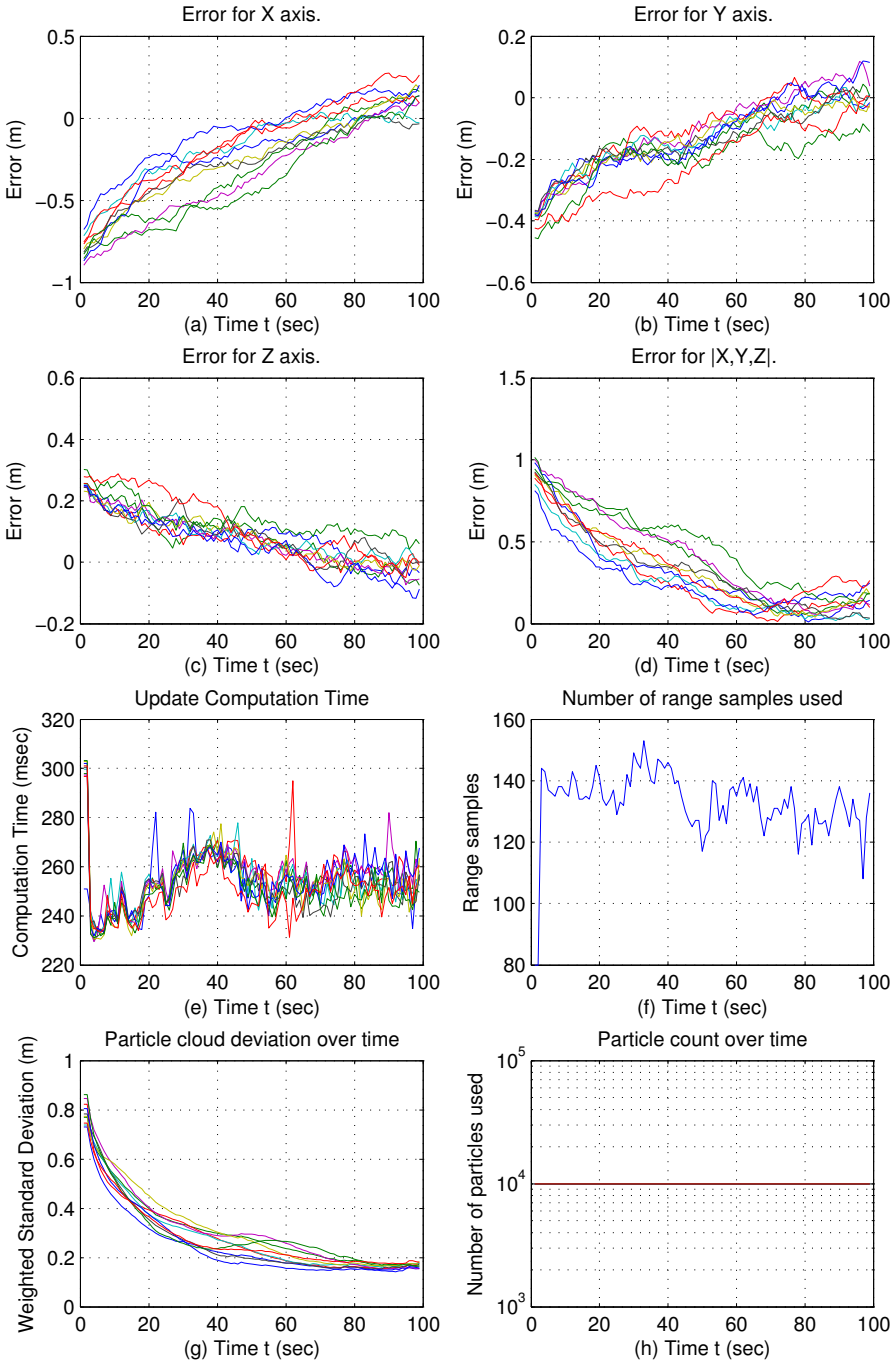


Figure 4.11: 10,000 Particles, AQUASensor.

4.3 KLD particle size

Clearly a small, constant number of particles is insufficient for estimating the position space when that search space becomes larger. Although the use of a constant, large number of particles is possible, it is inefficient once the particle filter starts to converge. Applying the KLD algorithm (see Section 2.2.2) to the particle filter allows the number of particles to be varied depending on the particle distribution. Initially the particles are distributed uniformly throughout the map; many particles are therefore needed to cover the entire space effectively. As the filter converges, the number of particles is reduced during the resampling phase, maintaining the represented distribution while using fewer particles and eliminating unlikely particles.

This section shows results using KLD sampling (see Figures 4.13 to 4.15 for a visual representation). As expected, the smaller maps still converge reliably, since they already worked well with fewer particles. The larger models, such as the lighthouse and large subway models (Figures 4.16 and 4.20, respectively) now converge much more reliably, as shown in Figure 4.12. At the same time, once the filter begins to converge the computational time drops to 50-100 msec per sensor update. The initial update time is pushed to a maximum of about ten seconds, during which time a large number of particles are seeded to cover the space. Since most particles die quickly, the required number of particles is quickly reduced, (even prior to the filter converging) meaning this time penalty only occurs during the first few time steps.

The results for the lighthouse model are much improved (two failures compared to 77 failures), with only a single experiment not converging rapidly. After the filter re-initializes twice, it rapidly converges on the correct position estimate. These two failures are the only ones recorded for the KLD experiments with the lighthouse model, while the particle filter using a constant number of particles failed 77 times on the same dataset (over 10 runs).

The use of 10,000 particles is clearly insufficient when estimating the robot's pose in the large subway model; particles initially converge along different paths. On the other hand, when using KLD sampling the same dataset consistently converges at around $t = 30$, with a small deviation around time $t = 70$. This clearly indicates that the filter is consistently converging to the same "optimal" path, even though a certain error remains due to approximation errors from the NDT (i.e., caused by the NDT's discrete nature). The kiln model on the other hand takes a long time to converge. Comparing Figure 4.22(d) and (f) shows that the data contained in the initial range scans is very limited (i.e., less than 50 points are being used). Shortly after more data becomes available the

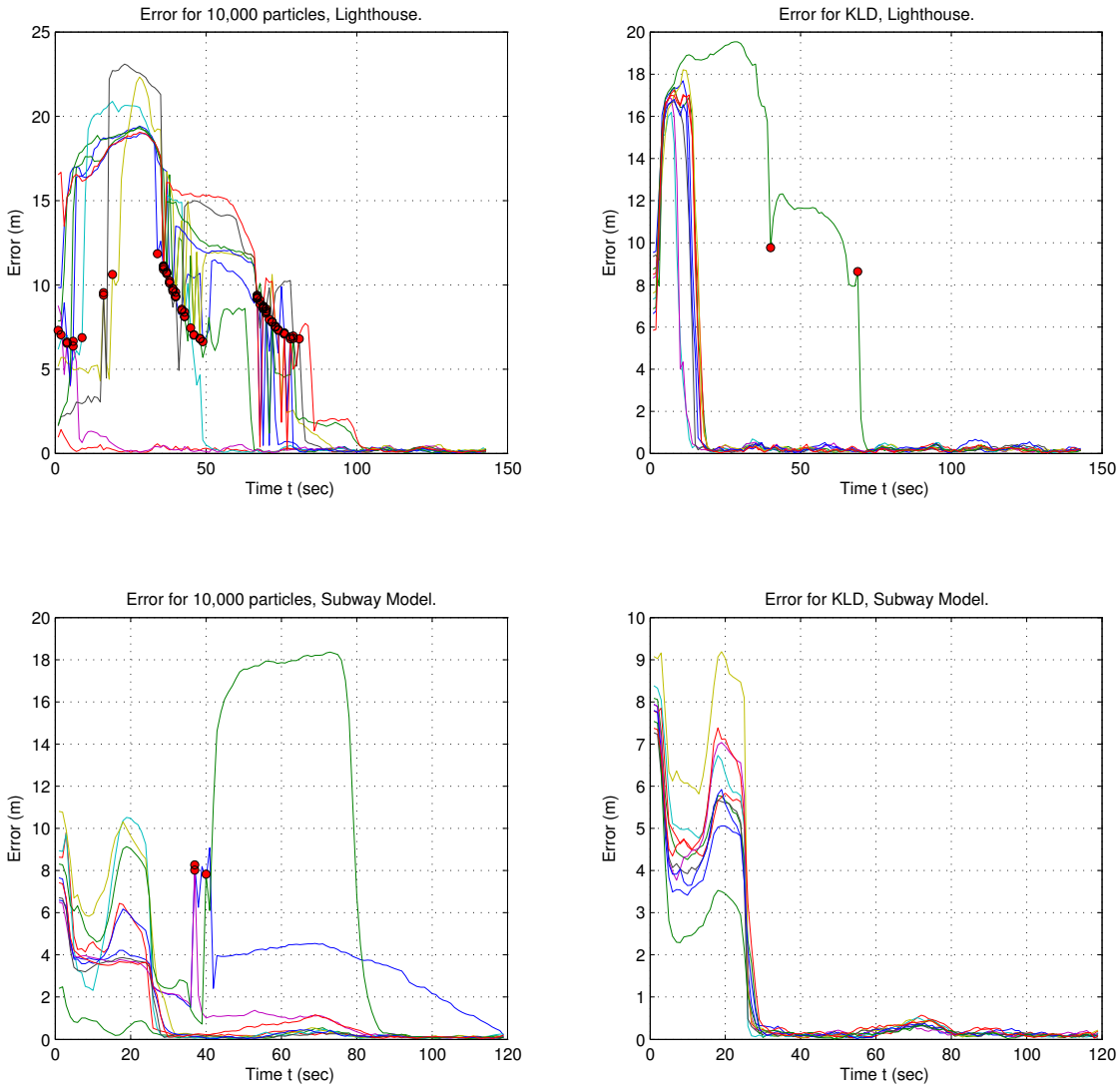
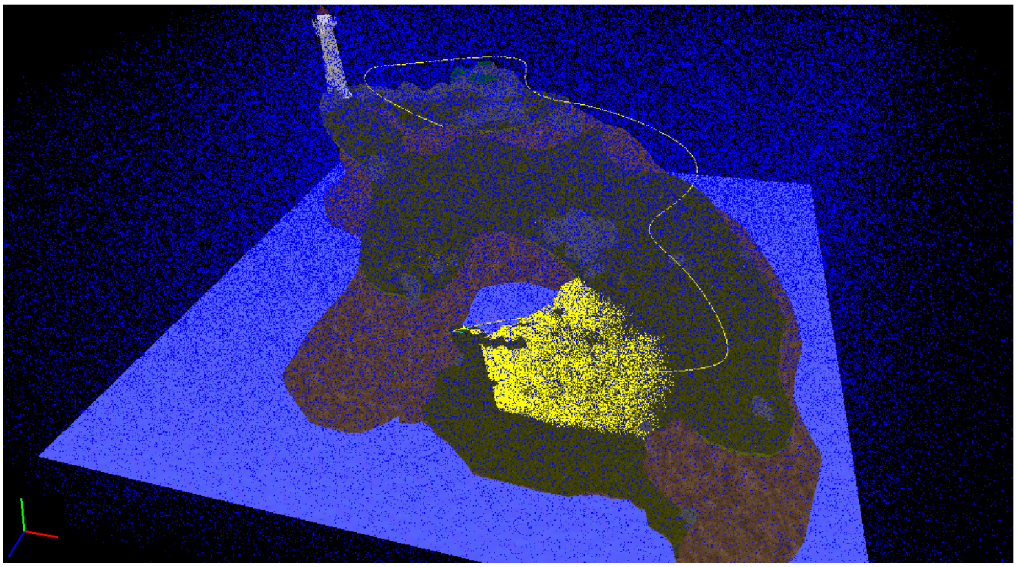
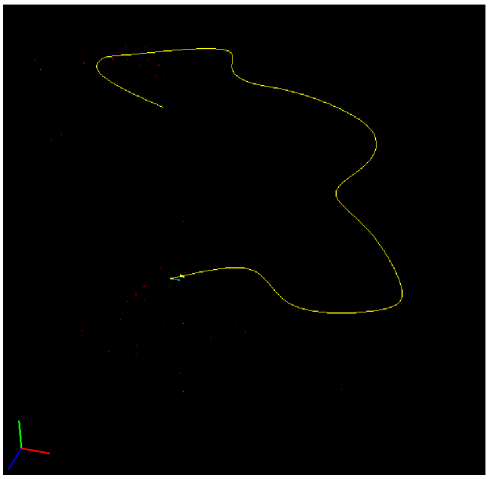


Figure 4.12: Convergence comparison between 10,000 particles (left) and a variable number of particles using KLD sampling (right). The results for the KLD approach clearly show rapid/consistent convergence of the pose estimate. The red dots indicate filter failures.

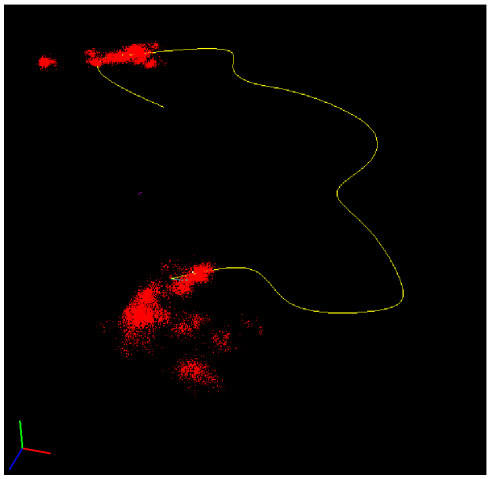
filter converges rapidly, finally eliminating particles in incorrect regions of the map.



(a) Initial random distribution at $t = 0$, using 250,000 particles.

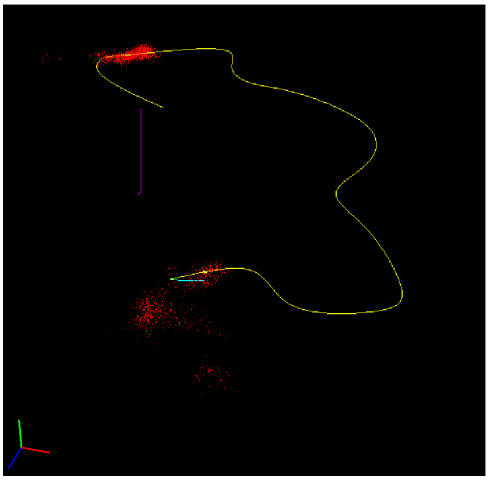


(b) Resampling the belief at $t = 0$, the majority of particles have zero weight and have been eliminated by the resampling step.

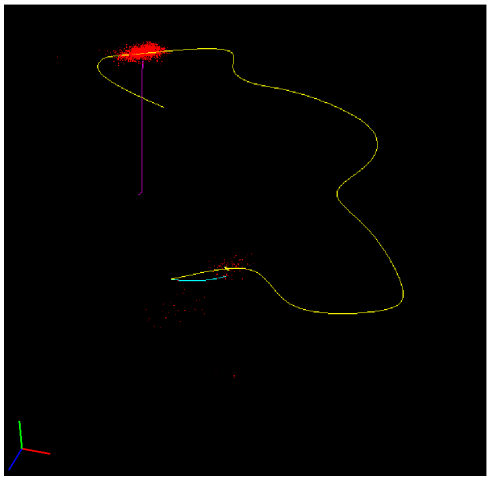


(c) Belief at $t = 2$. Since the filter is only sampling a portion of the position space, the particle count is reduced to 21282 particles.

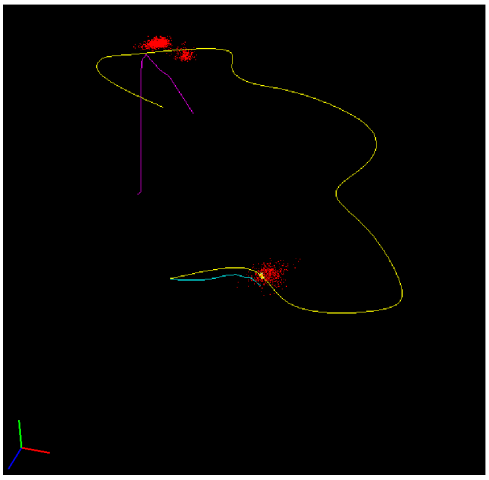
Figure 4.13: The following set of figures illustrate the position estimation component of the filter. Although difficult to see (b) contains particles in several locations. They become more visible at the next time step in (c), after control u_1 is integrated. The yellow line indicates the ground truth position path.



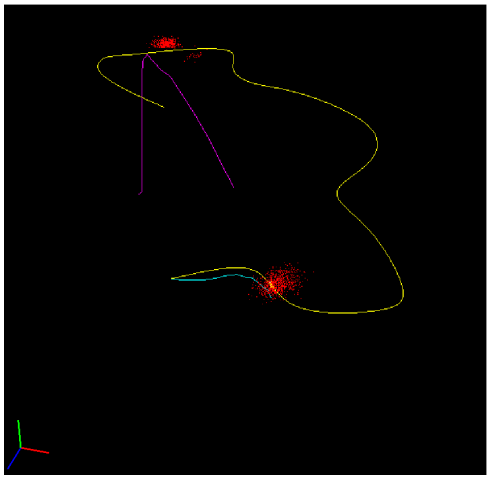
(a) Position estimate at $t = 4$, using 12,974 particles. The update took 1095ms.



(b) Position estimate at $t = 6$, using 9,365 particles. The update took 518ms.

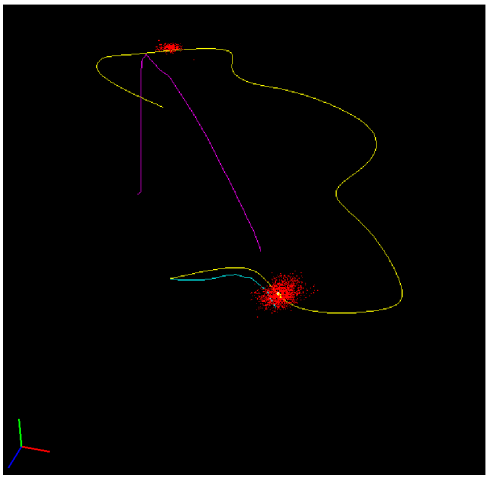


(c) Position estimate at $t = 11$, using 5,028 particles. The update took 254ms.

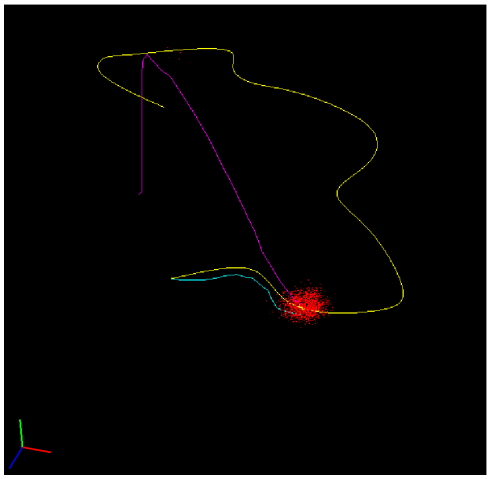


(d) Position estimate at $t = 13$, using 5,580 particles. The update took 308ms.

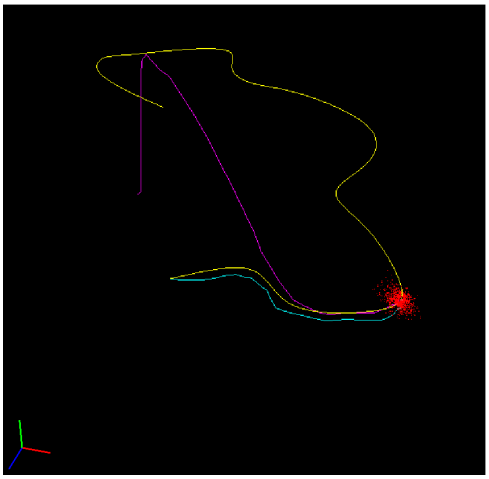
Figure 4.14: Continuing evolution of the particle fitler. Although the upper particle cloud initially gains increased likelihood, over time the sensor measurements become inconsistent with the map. At this point the particle cloud near the bottom of the figures gain likelihood. The purple line indicates the estimated robot pose, based on the particle cloud mean. This estimate is wrong since a simple weighted mean is used, even though two peaks exist in the likelihood (i.e., (d) shows an estimate inconsistent with either particle cloud). MAP estimation would be an effective option for recovering a more robust position estimate in this case.



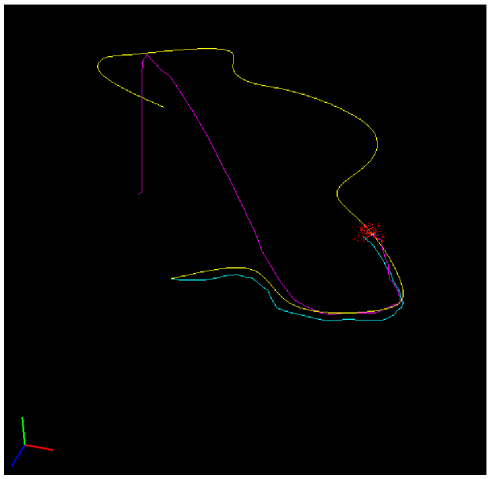
(a) Position estimate at $t = 15$, using 4,248 particles. The update took 232ms.



(b) Position estimate at $t = 18$, using 4,248 particles. The update took 229ms.



(c) Position estimate at $t = 30$, using 2,807 particles. The update took 118ms.



(d) Position estimate at $t = 40$, using 3,045 particles. The update took 161ms.

Figure 4.15: In (a) the upper cloud is still present, but after the next update it starts to disappear (b), and has vanished entirely in (c). The pose estimation problem is now akin to pose tracking, shown in (d). The light blue line indicates the raw odometry estimate.

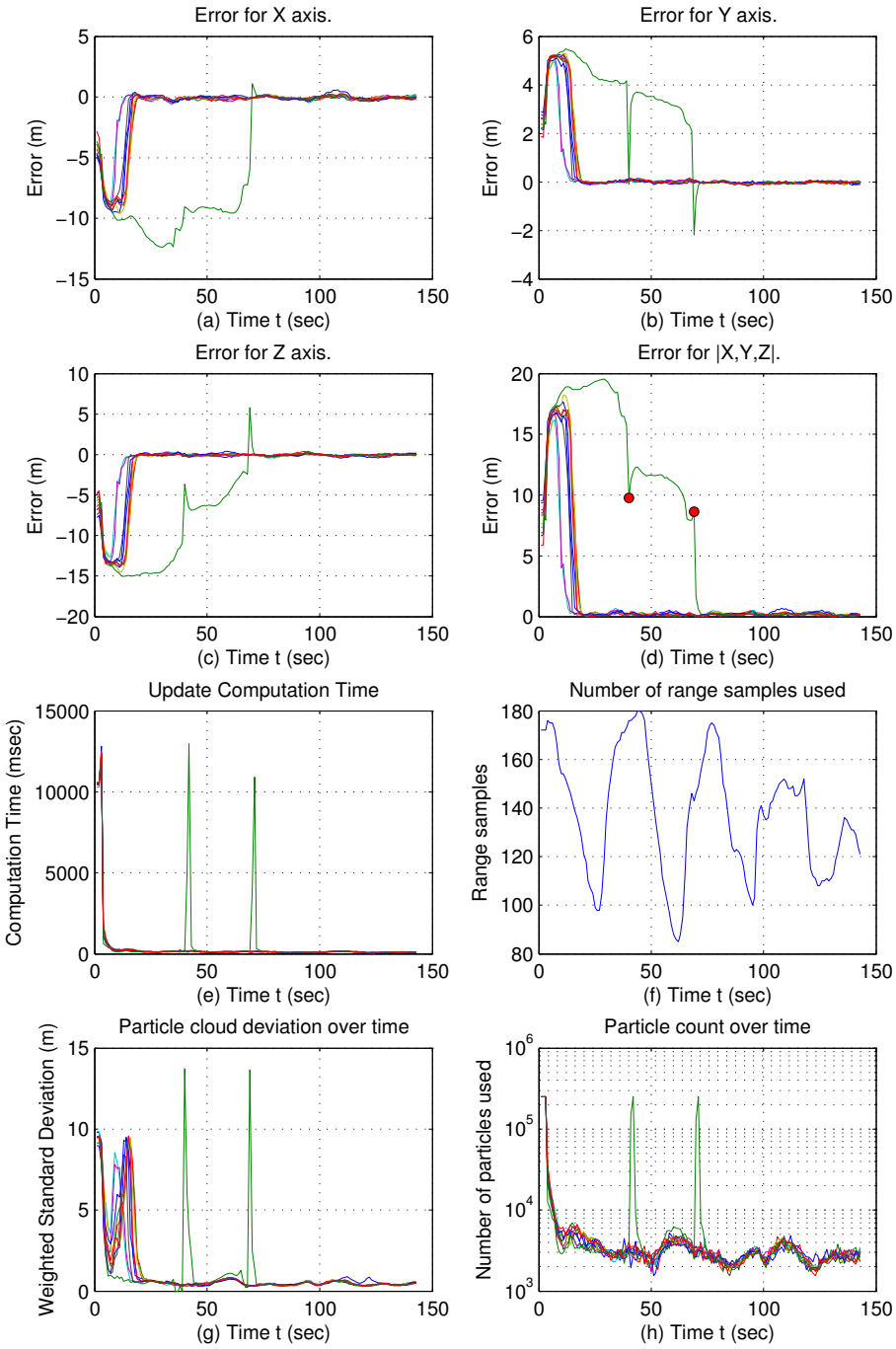


Figure 4.16: KLD Lighthouse.

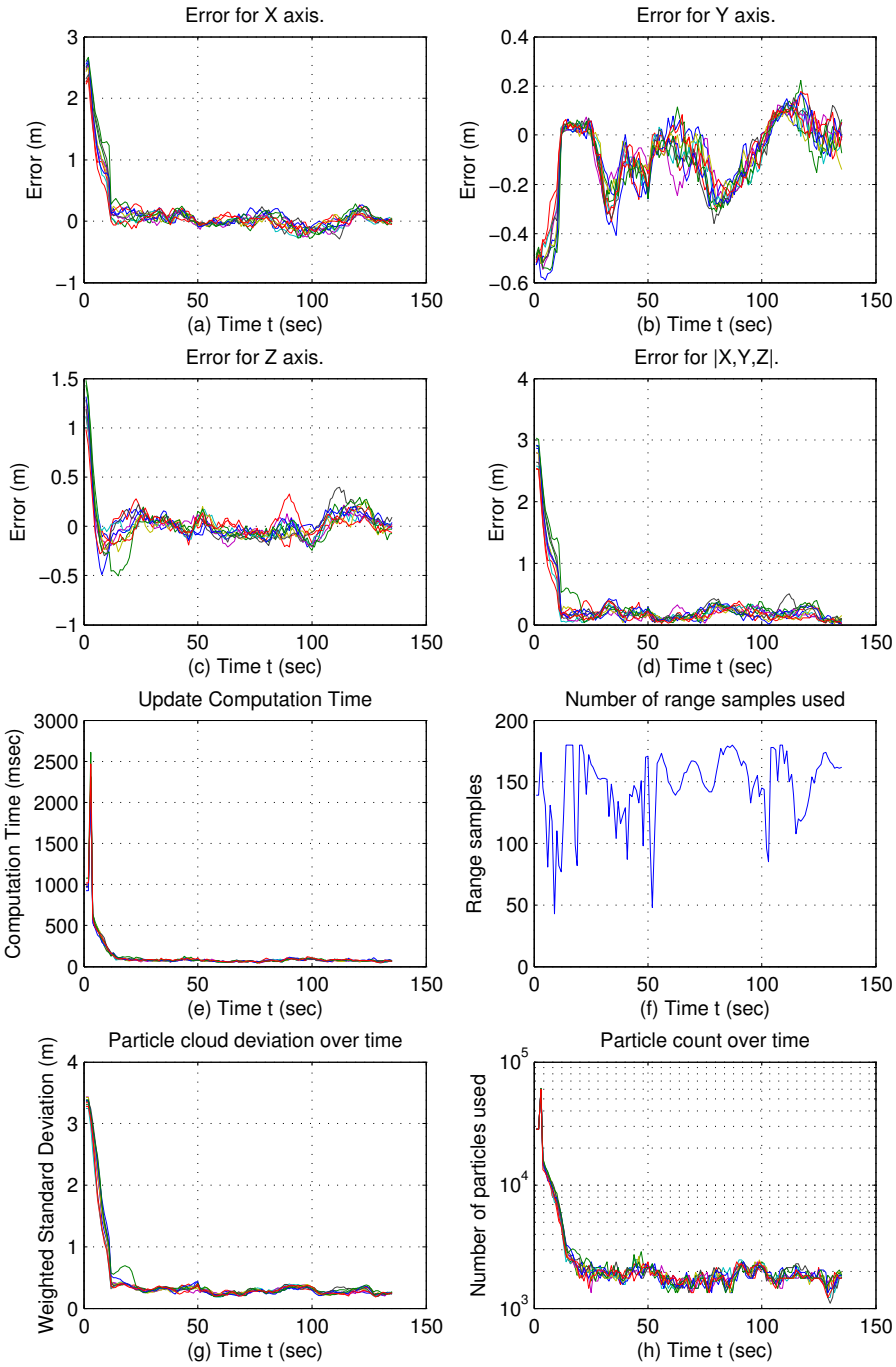
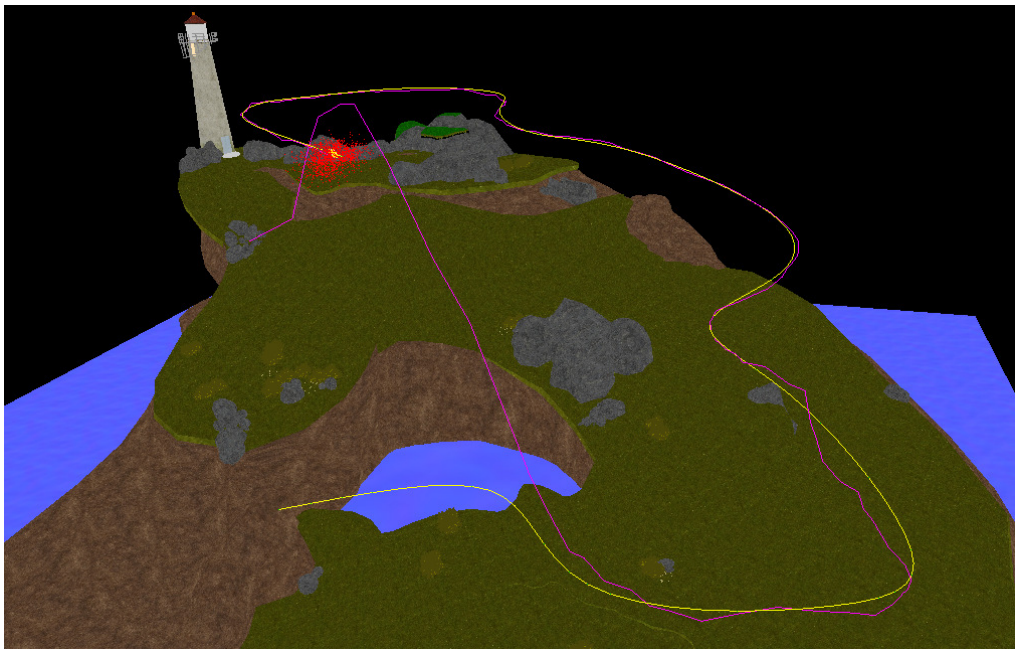


Figure 4.17: KLD Mini Level.



(a) Lighthouse model

(b) Mini Level

Figure 4.18: The yellow path indicates the ground truth, while the purple path shows the filter's best estimate.

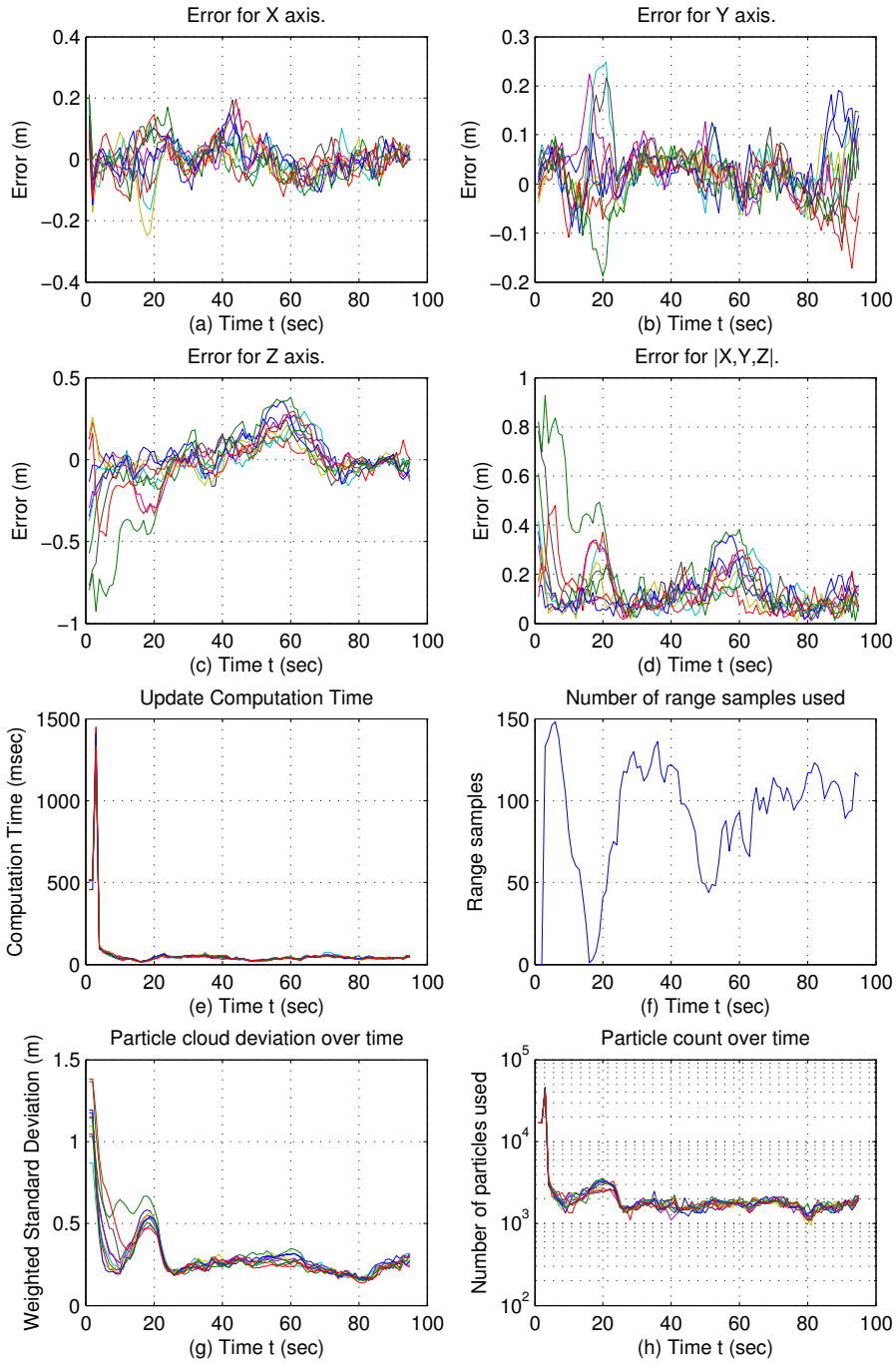


Figure 4.19: KLD Subway.

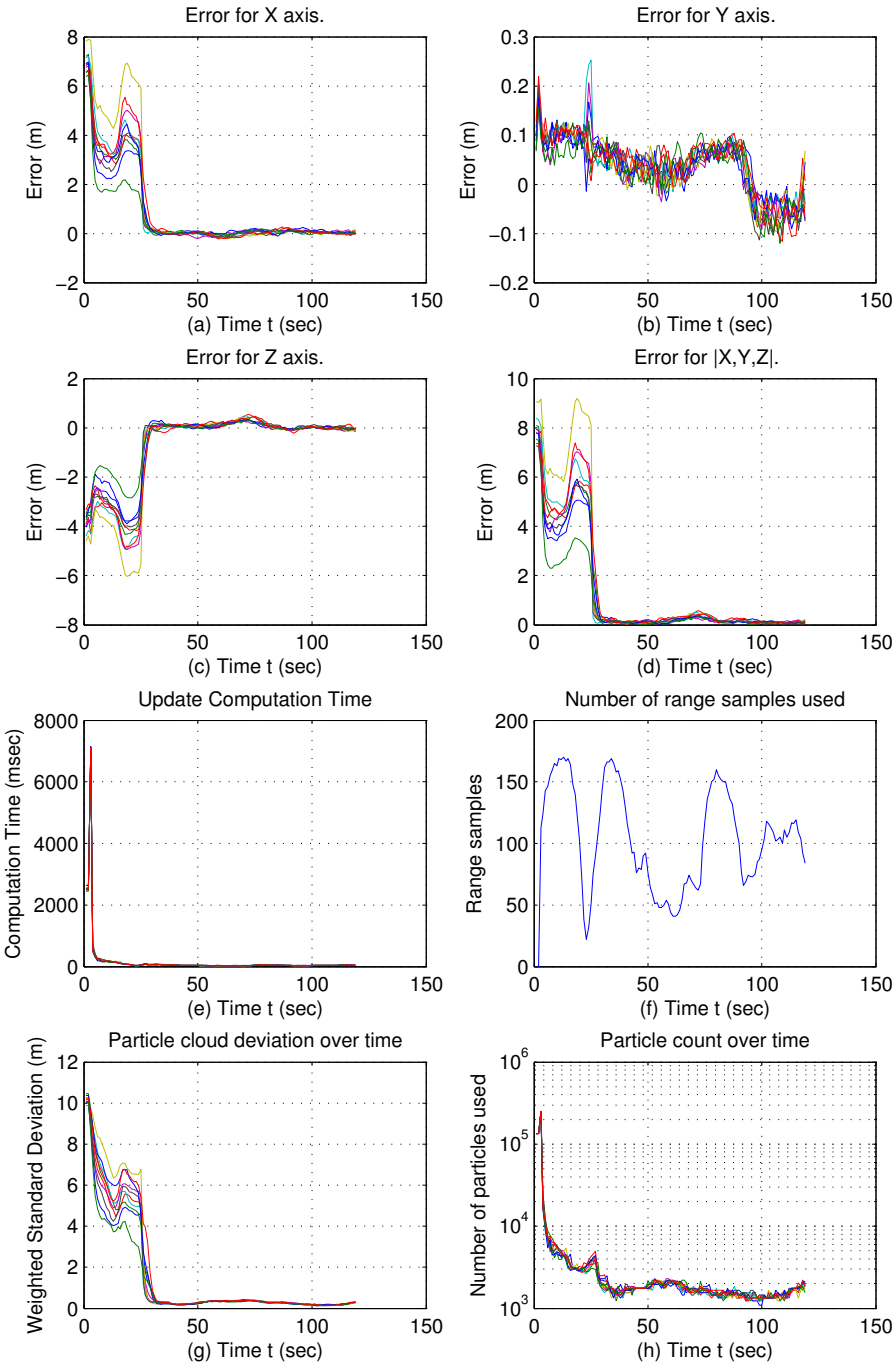
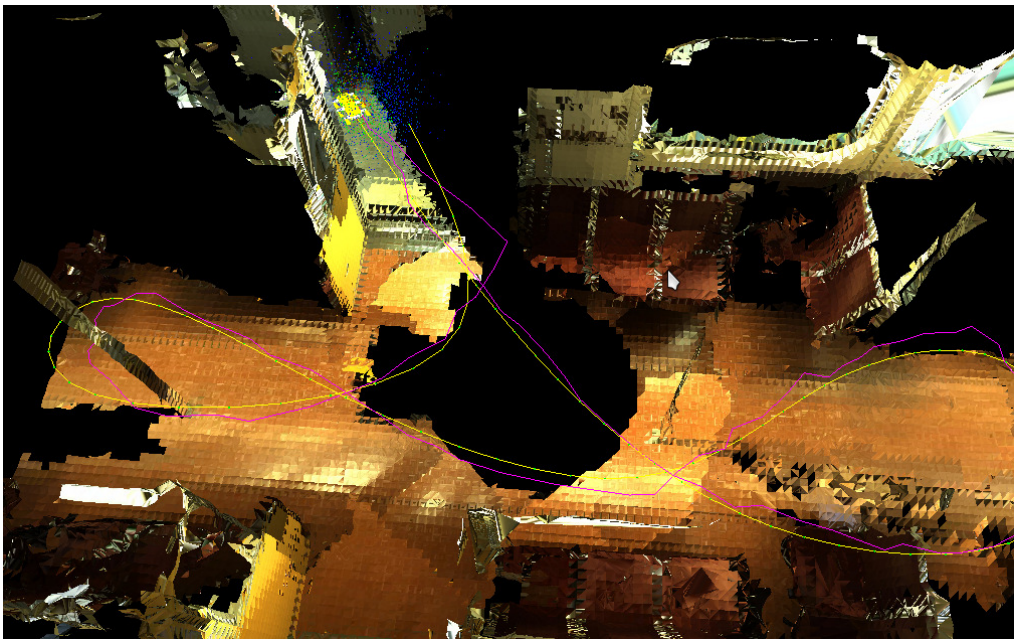
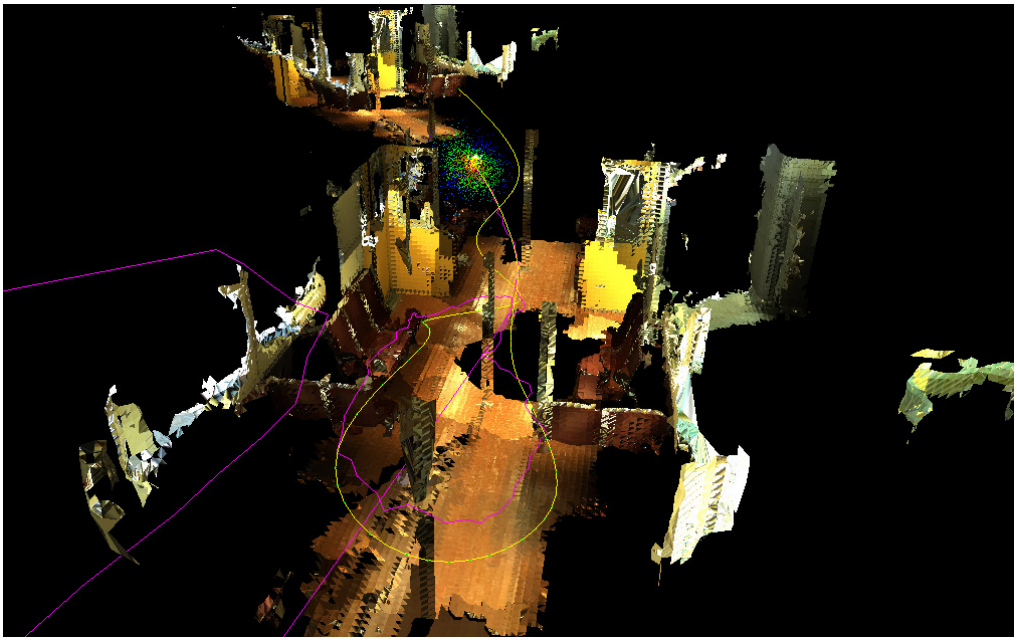


Figure 4.20: KLD Subway Large.



(a) Subway model



(b) Large Subway model

Figure 4.21: The yellow path indicates the ground truth, while the purple path shows the filter's best estimate. The purple lines in the lower left of (b) are due to a simple mean being used.

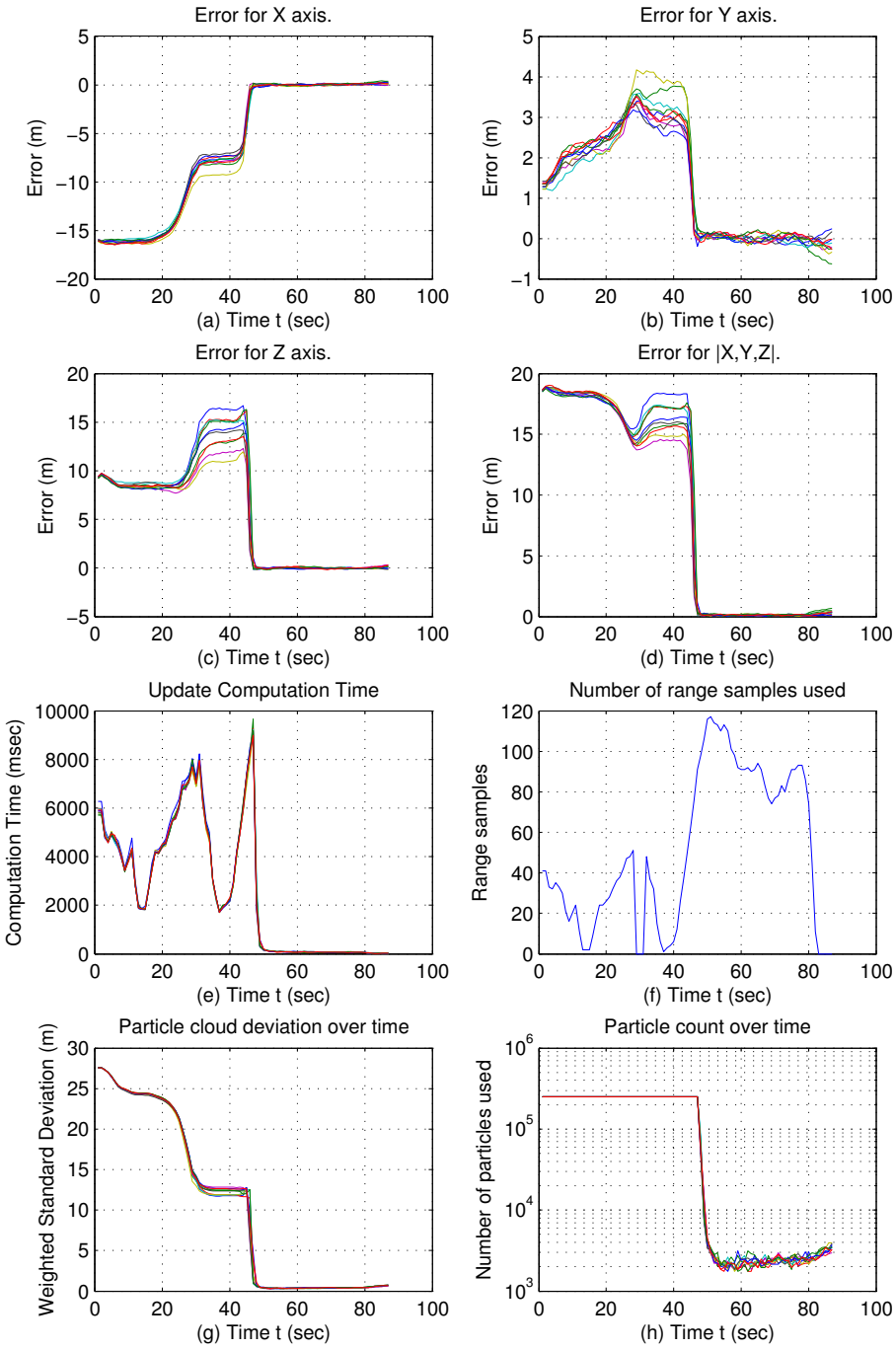


Figure 4.22: KLD Kiln.

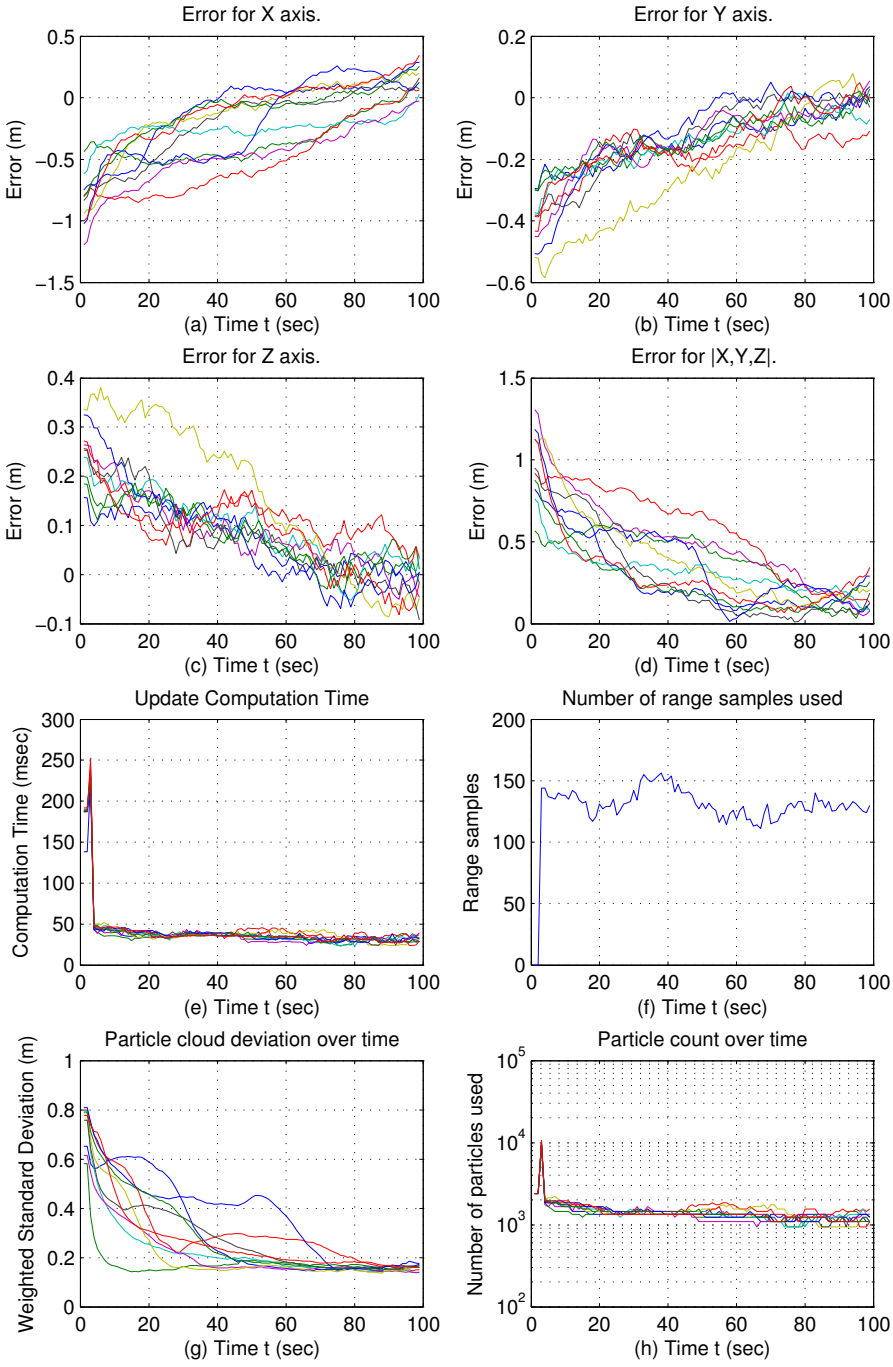


Figure 4.23: KLD AQUASensor.

4.4 Comparison with simple particle filter

Decoupling the orientation estimate from the position estimate allows pose estimation to be performed at a reasonable computational cost. In order to demonstrate the efficacy of this approach, a comparison with a traditional particle filter is presented. This example filter estimates all parameters of the state, both the position and orientation, using only particles. As such, the space that needs to be searched by the particle filter is much larger, and should require more particles. For fairness, the same simulator is used, allowing the same noise to be added to the system, and the same sensor readings to be presented to the filter. The only changes are that the orientation control in u_t (not used in the developed filter) is now used as well. In order to be fair, no noise is added to the change in orientation, i.e., for orientation states θ_{t-1}, θ_t, the control data (change in orientation, represented using quaternions) provided is $\theta_\Delta = \theta_{t-1}^{-1}\theta_t$. While not realistic, it gives the naïve 6DOF filter a reasonable advantage. In order to eliminate KLD sampling as a potential source for bias, the comparison uses a constant number of particles (for which reason the execution time is not listed). The tests are performed on the smaller game dataset in Figure 4.1(e).

Running the filter repeatedly and varying the size of the particle cloud reveals the number of particles required to converge consistently, as well as the minimum number of particles with which the filter starts to fail. Only experiments where the number of particles showed a result of interest are shown here, although experiments with different numbers of particles were also conducted. When using the EKF enabled filter, it consistently converges with 750 particles (See Figure 4.24). Using 500 particles the filter becomes less stable, and exhibits some occasional failures (10 failures in 100 experimental runs, where failure is defined as the position estimate being off by at least one meter from the correct estimate at time $t = 50$).

On the other hand, disabling the EKF, and instead sampling all six parameters, results in a requirement for a much greater number of particle. Figure 4.24 shows experiments using 50,000, 150,000 and 250,000 particles using a naïve implementation of the particle filter. The experiments using 50,000 or 150,000 particles show consistent failures. When using 150,000 particles the failure rate is around half, with the other half correctly converging on the desired pose. Increasing the number of particles to 250,000 (a squared increase compared compared to the 500 when using an EKF) results in 19 failures in 100 experimental runs.

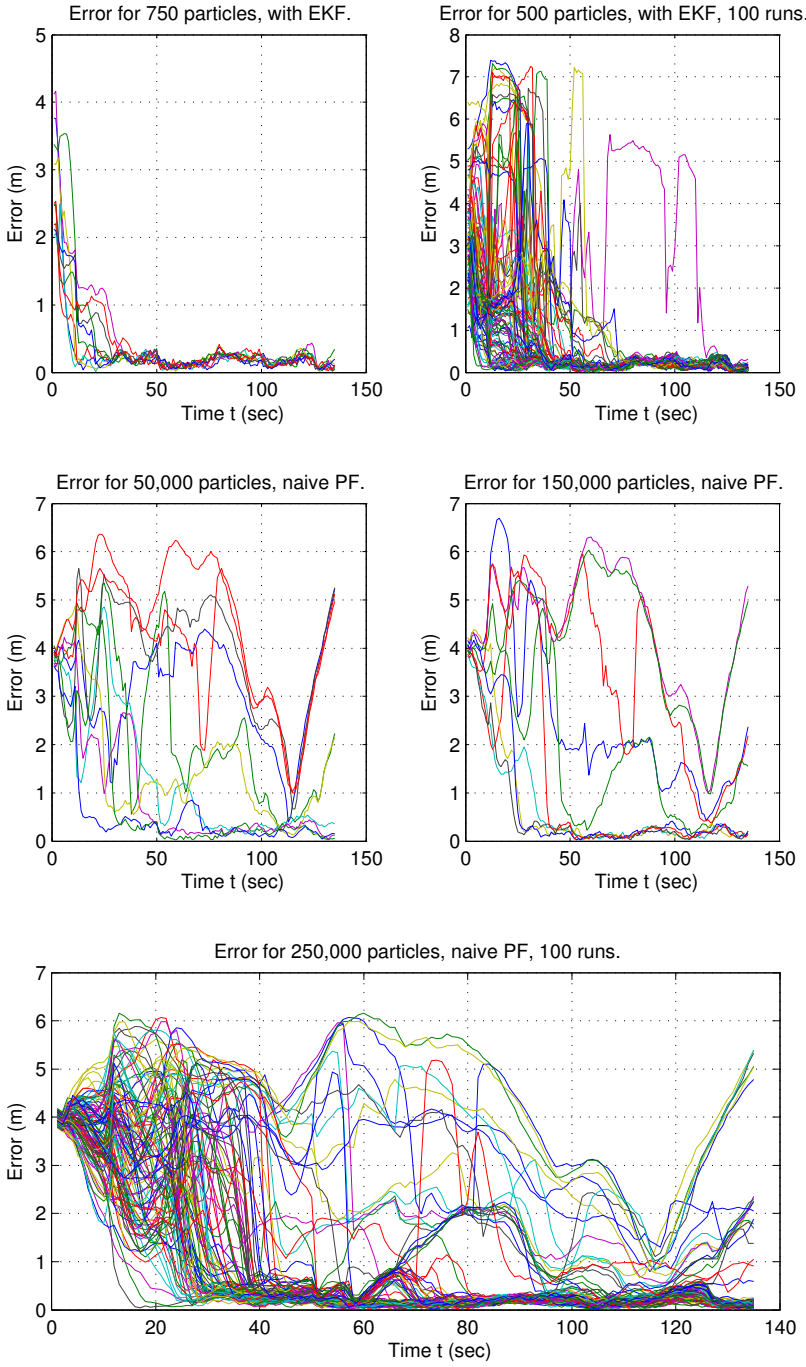
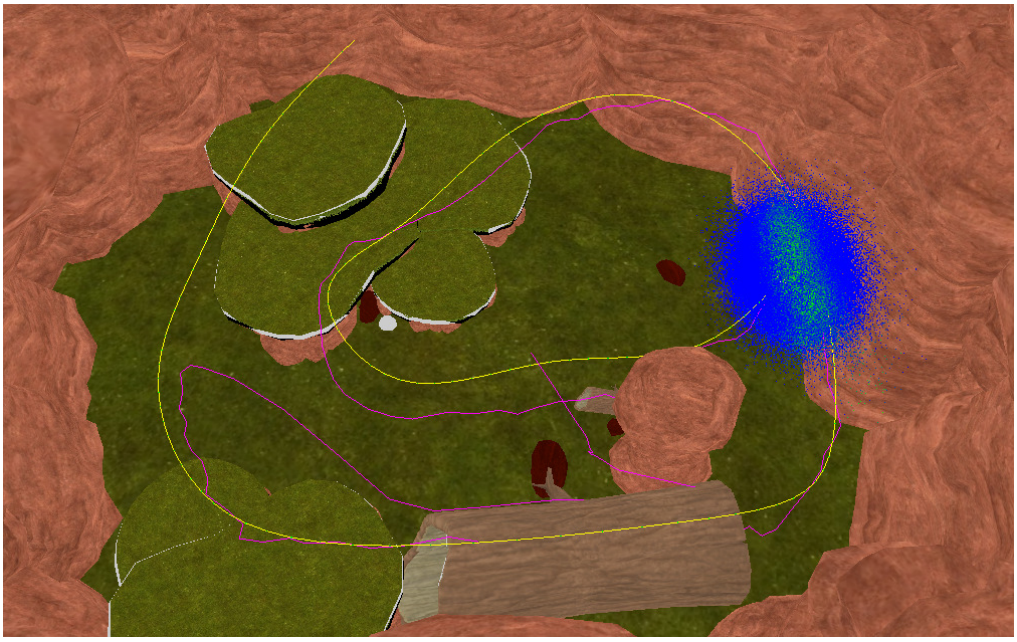


Figure 4.24: Comparing the filter in this work (top) with a naïve 6DOF particle filter (middle and bottom). The map size used for this experiment was $11 \times 3.8 \times 10.6m^3$.

(a) Mini Level, particle filter with EKF, 750 particles. ∼30 msec per update.



(b) Mini Level, naïve 6DOF particle filter, 250,000 particles. ∼10 seconds per update.

Figure 4.25: The yellow path indicates the ground truth, while the purple path shows the filter's best estimate. Clearly (a) shows better accuracy, at a vastly reduced computational cost.

4.5 Summary

The filter developed in Chapter 3 was tested on six datasets using a constant 10,000 particles. These experiments show that the particle filter works well on smaller maps, but also indicates that some of the larger models require more particles in order to converge reliably. This makes intuitive sense, as the filter would be under-sampling the position search space of the larger maps. Although alternative approaches may be possible (such as making the resolution for $p(z_t|x_t, \theta_t)$ coarser, discussed in the next chapter), an increase in particles can be used to solve the under-sampling problem. Using KLD sampling, we show that the filter consistently converges on all of the datasets when provided with more particles. The KLD algorithm allows the number of particles to be varied dynamically, resulting in increased accuracy, while allowing a fast update time once the filter begins to converge.

Naturally, a comparison with an alternative solution is warranted in order to show the effectiveness of the approach. The limited availability of existing 6DOF solutions leads to comparing the system to a naïve implementation of a 6DOF particle filter, which estimates all six parameters. In order to be fair to this alternative, both are provided with the same experiment parameters and the same data. In addition, the rotation component of the control u_t is given without noise, i.e., a perfect control is provided. Due to the inherent increased dimensionality of the approach, the comparison of course shows that a 6DOF particle filter requires a significant increase in the number of particles. The required number of particles for 6DOF pose estimation is exponentially increased (in terms of the number of dimensions) when compared to the filter developed in this work (from $500 \rightarrow 500^2$, for similar failure rates).

These results clearly indicate the advantage and effectiveness of efficiently estimating a portion of the state directly using a sensor such as an IMU. This approach leads to significant time savings, while using the NDT as a likelihood estimator allows for a greatly reduced memory footprint and additional computational savings. This allows the presented filter to be implemented on existing robot hardware, without requiring additional computational resources.

Chapter 5 Summary and future work

Pose estimation is an important capability for mobile robots; without it, motion planning may become difficult or even impossible. Although pose estimation is an important task for robots able to move with six degrees of freedom, directly extending the existing work for 2D pose estimation into the 3D realm is not computationally attractive. The increased dimensionality of the problem requires a vastly larger number of particles that typically cannot be effectively computed using current hardware. Reducing the dimensionality of the problem is key to reducing the computational resources required. Some solutions attempt to do so by limiting the motion of the device, or by first computing a 2D pose estimate and then extending that estimate into 3D. Such approaches assume the the motion of the device can be limited, or that a reasonable 2D pose estimate can be computed independently of the 3D pose. An alternative approach is needed if such assumptions are not valid; this work has developed such an alternative.

5.1 Contributions of the work

The primary contribution of this work is the development of a 6DOF pose estimation filter that separates pose estimation into two related problems. The first problem is the estimation of the uni-modal components, in this case the orientation components, of the pose using an efficient filter (such as the Kalman filter). The second portion of the filter then estimates the remaining parameters using a particle filter, conditioned on the parameters previously estimated efficiently. A secondary contribution in this work is the novel use of the NDT for estimating the range likelihood. This allows the range likelihood to be evaluated efficiently, while using relatively little memory to store the map.

Overall, this approach reduces the dimensionality of the pose which needs to be estimated using sampling methods, by taking advantage of an IMU to estimate the device's orientation separately from its position. This permits a significant reduction in the required computational resources. The effectiveness of this approach was demonstrated through a variety of experiments on both simulated and real world environment

models. In order to further reduce the computational requirements, the NDT is used as an efficient likelihood estimator for the range measurements. This eliminates the need to generate a pseudo-scan for each particle, as would be needed with some approaches. Instead the NDT is used to directly evaluate the likelihood of observing a range scan. Due to variable map sizes, a constant number of particles is likely to either under- or over-sample the position space, making it attractive to vary the number of particles based on the distribution of the particle cloud. To this end, the KLD algorithm is applied to the particle filter component of the filter to vary the number of particles as needed. This results in both reducing the number of failures of the filter, and also reduces the computational cost once the the filter has converged. Although the filter was developed as a result of work in underwater robotics, the filter is applicable to many 6DOF vehicles equipped with 3D sensors. For example, ground based outdoor vehicles could make use of the filter, as they operate in semi-6DOF environments, see Figure 5.1. Although the vehicle itself cannot come off the ground, the ground itself changes in elevation. This can cause the vehicle to operate in orientations far from the clean, level orientation expected for typical 2D pose estimation solutions, requiring at least a limited 6DOF solution.

5.2 Limitations of the work

The focus of this work has been on vehicles with characteristics similar to the AQUA robot. Certain implicit assumptions that are made regarding the robot may not be valid when considering other platforms. For example, the work assumes a vehicle that moves slowly enough such that there is significant scene overlap between successive video frames. This allows for visual odometry (egomotion estimation) to be used for a priori estimation of the change in pose for the robot. Some alternative method must be employed to estimate the change if this work would be applied to estimating the pose of a fast moving vehicle. The rate of such a vehicle would make visual odometry challenging, since overlapping video frames would be less likely. This work could still be applied using some other motion estimate/model, combined with a (potentially) more generous noise model.

The developed filter does not explicitly account for missing data in the models; such as the presence of people, windows, or “holes” in the map. People and objects that move around in the environment cause readings by the range sensor that will not match anything in the “known” environment. This may result in wasting resources on searching for objects that are not found the map. While, to a certain extent, the NDT approach is robust to such missing data, the evaluation of data that won’t match anything

Figure 5.1: YURT Mars rover prototype [69]. Extending work to semi-6DOF robot platforms, such as this rover, would be an area of interesting work. The reduced motion capabilities of the platform (compared to the AQUA robot for example) could be exploited to reduce the number of particles required to effectively sample the position space.

in the map is a waste of resources. Extension of the 2D novely and entropy filters in [8] to 3D could improve some of these results.

5.3 Future work

Many interesting areas remain for potential future investigation. The obvious experiment to conduct is the collection of a complete data set using a robot such as the AQUA robot. Although the range data for the barge model (generated using the AQUASensor) is real, simulated IMU data was used when testing the effectiveness of the filter. Most of the other world maps are based on real data, although the egomotion and range data

expected from the AQUASensor are simulated. Although the simulations in this work incorporate noise based on experimental values from real world trials, it is no substitute for real data. In addition, the ground truth used in this work is based on egomotion estimation. This works reasonably well for the experiments which simulate the AQUA sensor, but when testing on complete datasets, it would be better to have a more robust ground truth. Performing above ground pose estimation tests using a psuedo-robot (vision system and IMU) in a room equipped with an absolute tracking system (such as a VICON motion capture system [72]) would provide a reliable ground truth against which to test the algorithm's accuracy. Once such experiments have been performed, applying the algorithm to a live 6DOF robot would clearly be of interest. The current implementation, once stripped of the graphical components, is certainly capable of being run in real-time on current robot hardware, assuming the AQUASensor is also adapted for real-time use.

Extending the work to alternative robot platforms, which may not operate in true 6DOF environments, would also be an interesting area for future work. There are many applications where a vehicle is not truely 6DOF, yet existing technologies for 3DOF pose estimation prove to be limited. For example, using the filter with senors mounted on a ground based vehicle operating outdoors in rough terrain. Although not truly 6DOF, the environment is sufficiently complex to introduce elevation changes, as well as orientation changes beyond the standard, level, yaw angle. Such conditions make 3DOF solutions ineffective at best or even unapplicable. Application of the developed filter could be made more efficient by adjusting the motion model to consider the constant height of the sensor off the ground ($\pm\epsilon$). Such a model could prevent particles from moving too far (or too close) from the ground, while still allowing a much more realistic 3D map to be used, and also considering the non-planar nature of the environment.

Another potential area of interest is the orientation estimate. The current implementation of the filter uses a single Kalman filter for estimating the orientation. Given sufficient computational resources, a potential improvement could be to compute a Kalman filter for each individual particle. While more time would be required per update, and in theory each Kalman filter should converge to the same orientation estimate, it may none-the-less improve the overall accuracy of the filter. It would also allow integrating the expected change in orientation for each particle into the particle's Kalman filter, which is currently not done.

Another area worthwhile investigating is what happens when the orientation estimate is not reliable. If, for example, the robot moves violently for a short period, the Kalman estimate may not be reliable. Instead of fully updating the particle filter

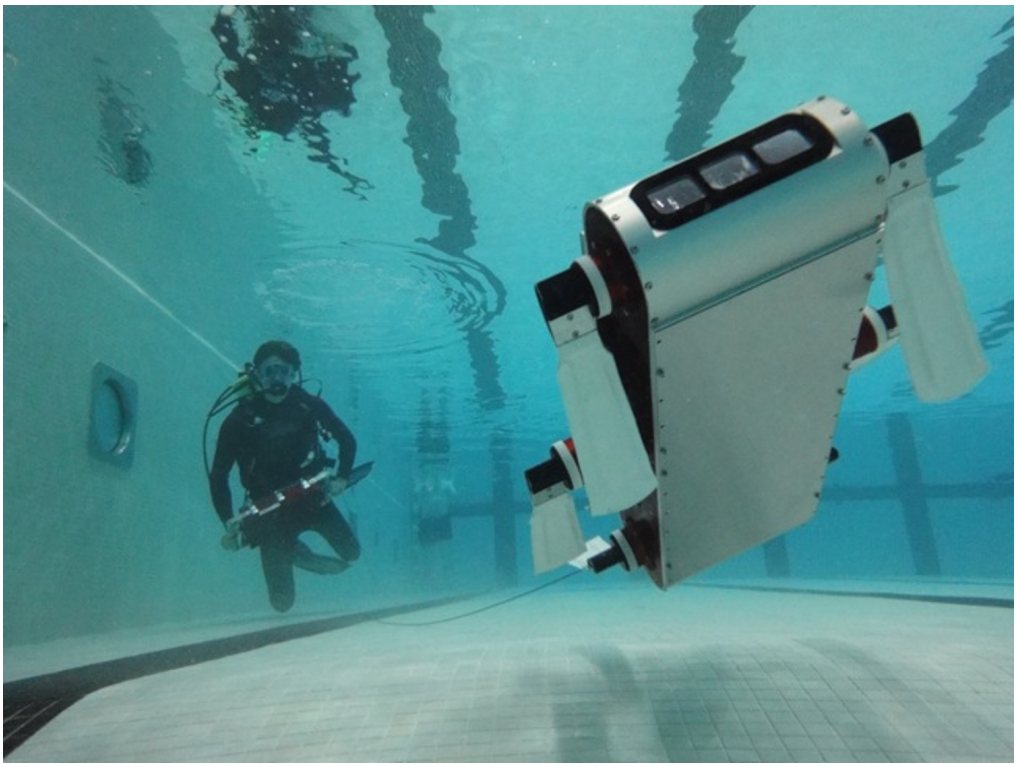


Figure 5.2: The AQUA robot deployed in a 6DOF environment, illustrating the roll and pitch capabilities of the vehicle. Stereo vision sensors are visible at the front of the robot.

and potentially destroying the previously accurate estimate, it may be advantageous to temporarily halt the particle filter. It may be possible during such a time to accumulate the controls, and once the Kalman filter becomes more reliable again integrate all the controls in one (accumulated) step. This would cause the particle filter to diverge considerably, but possibly leave the particles in likely regions. The next measurement update should then be able to eliminated those particles that strayed too far.

The filter, as developed in Chapter 3, contains the term $p(i_t|x_t, \theta_t)$, i.e., the likelihood of observing the unfiltered IMU data given the currently estimated state. The current implementation of the filter ignores the computation of this distribution, instead focusing on $p(z_t|x_t, \theta_t)$; i.e., the likelihood $p(i_t|x_t, \theta_t)$ is assumed independent of x_t. Exploring cases where this condition does not hold, such as in areas with high magnetic variations, would be an interesting area for future work.

The position result of the current filter tends to converge with an error margin of approximately 20cm. More accurate results would likely be achieved through the use of several refinements. The NDT used in this work is run using a cell size of 20cm; varying this size dynamically while the filter is running should vary the accuracy as well. Increased cell size would likely lead to a less accurate estimate, but might work well when the particles are widely distributed. Once the filter begins converging, it may be advantageous to decrease the cell size, most likely resulting in a more accurate estimate. A different approach would use the current pose estimate and then use the NDT as intended by its original authors; as a scan alignment tool. Starting with the best estimate from the filter, the position (and orientation) could be iteratively improved through standard optimization techniques, outlined in [46, 53]. Obviously, a similar approach could be accomplished using the ICP (or similar) algorithm, but this would require the original world model to be stored in memory, which may be unattractive.

The developed filter can be thought of as separating the pose into two groups. One of these can be estimated efficiently, and is characterized by a uni-modal distribution; the other may be multi-modal and requires a sampling approach. Since the filter is developed based on vectors, the specific dimensions to which the filter is applicable is flexible. This work focused on a task with six parameters, three for position and three for orientation. It would be relatively straightforward to extend the work to a greater number of parameters. Uni-modal parameters (assuming they do not depend on the sampled parameters), could be added to the state θ_t; those parameters that should be sampled, or are dependent on the θ_t parameters, could be added to the state x_t. Investigating such an extension could be useful for a variety of systems.

An area unrelated to robotics that could benefit from this work is in archaeological digs. Such sites could initially be mapped using LIDAR technology (the kiln dataset in Chapter 4 is from an archaeological dig), producing a detailed map of the site. Once such a map is produced, localizing an observer inside the map may be desirable. Unfortunately, as these digs may occur inside caves, the use of GPS localization is often not effective. Interest has been expressed in the ability for an archaeologist to use a hand held device (i.e., a unit containing a stereo camera and an IMU), and with it be able to track the observer's movements throughout a site. For example, such a device would permit the accurate tagging of items found in the dig site, among other things.

Bibliography

[1] A. Abdelhafiz, B. Riedel, and W. Niemeier. Towards a 3D true colored space by the fusion of laser scanner point cloud and digital photos. In *Proceedings of the ISPRS Working Group V/4 Workshop (3D-ARCH)*, Mestre-Venice, Italy, 2005.

[2] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5:349–359, 1999.

[3] Y. Boers, H. Driessen, and A. Bagchi. Particle filter based MAP estimation for jump Markov systems. Technical report, University of Twente, Twente, Netherlands, 2009. Preliminary Version.

[4] W. Burgard, F. Dellaert, D. Fox, and S. Thrun. Monte Carlo Localization: Efficient position estimation for mobile robots. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 343–349, Orlando, Florida, 1999.

[5] W. Burgard, D. Fox, J. Gutmann, and K. Konolige. An experimental comparison of localization methods. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 736–743, Victoria, B.C., Canada, 1998.

[6] W. Burgard, D. Fox, and S. Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

[7] W. Burgard, S. Thrun, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, 2005.

[8] W. Burgard, S. Thrun, D. Fox, and A. Cremers. Position estimation for mobile robots in dynamic environments. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 983–988, Menlo Park, CA, USA, 1998.

[9] O. Cappé, S. Godsill, and E. Moulines. An Overview of Existing Methods and Recent Advances in Sequential Monte Carlo. *Proceedings of the IEEE*, 95:899–924, 2007.

[10] R. Chen, J. Liu, and T. Logvinenko. A theoretical framework for sequential importance sampling and resampling. Technical report, Stanford University, Department of Statistics, 2000.

[11] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations (Intelligent Robotics and Autonomous Agents)*. The MIT Press, Cambridge, MA, 2005.

[12] D. Choukroun, I.Y. Bar-Itzhack, and Y. Oshman. Novel quaternion Kalman filter. *IEEE Transactions on Aerospace and Electronic Systems*, 42:174–190, 2006.

[13] Paolo Cignoni. Meshlab. `meshlab.sourceforge.net`.

[14] P. Corke, C. Detweiler, M. Dunbabin, M. Hamilton, D. Rus, and I. Vasilescu. Experiments with underwater robot localization and tracking. In *Proceedings of 2007 International Conference on Robotics and Automation (ICRA)*, pages 4556–4561, Albuquerque, New Mexico, 2007.

[15] I.J. Cox. Blanche-an experiment in guidance and navigation of an autonomous robot vehicle. *IEEE Transactions on Robotics and Automation*, 7:193–204, 1991.

[16] A. Derr, D. Fox, W. Burgard, and A. Cremers. Integrating global position estimation and position tracking for mobile robots: the Dynamic Markov Localization approach. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 730–735, Victoria, B.C., Canada, 1998.

[17] A. Doucet, N. de Freitas, K. Murphy, and S. Russell. Rao-Blackwellised particle filtering for Dynamic Bayesian Networks. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI '00)*, pages 176–183, San Francisco, CA, 2000.

[18] A. Doucet, S. Godsill, and C. Andrieu. On Sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10:197–208, 2000.

[19] A. Doucet and A. Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. Technical report, Department of Statistics, University of Warwick, Coventry, UK, 2008.

[20] Hans Driessen and Yvo Boers. Map estimation in particle filter tracking. In *2008 IET Seminar on Target Tracking and Data Fusion: Algorithms and Applications*, pages 41–45, Birmingham, UK, April 2008.

[21] G. Dudek and M. Jenkin. *Computational Principles of Mobile Robotics*. Cambridge University Press, New York, NY, 2000.

[22] G. Dudek and M. Jenkin. *Computational Principles of Mobile Robotics*. Cambridge University Press, New York, NY, 2nd edition, 2010.

[23] D. Fox. Adapting the sample size in particle filters through KLD-Sampling. *International Journal of Robotics Research*, 22:985–1003, 2003.

[24] D. Fox, W. Burgard, S. Thrun, and F. Dellaert. Robust Monte Carlo Localization for mobile robots. *Artificial Intelligence Journal*, 128:99–141, 2001.

[25] D. Fox, D. Hennig, W. Burgard, and T. Schmidt. Estimating the absolute position of a mobile robot using position probability grids. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 896–901, Portland, Oregon, 1996.

[26] D. Fox, C. Kwok, and M. Meila. Adaptive real-time particle filters for robot localization. In *Proceedings of the IEEE International Conference on Robotics and Automation, 2003. Proceedings (ICRA)*, volume 2, pages 2836–2841, Taipei, Taiwan, 2003.

[27] D. Fox, C. Kwok, and M. Meilă. Real-time particle filters. In *Proceedings of the IEEE*, pages 469–484. MIT Press, 2004.

[28] N. De Freitas, A. Doucet, and N. Gordon, editors. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, Freiburg, Germany, 2001.

[29] A. German, A. Hogue, H. Liu, C. Prahacs, A. Ripsman, R. Sim, L. Torres, P. Zhang, M. Buehler, G. Dudek, M. Jenkin, C. Georgiades, and E. Milios. AQUA: An aquatic walking robot. In *Proceedings of the IEEE/RSJ/GI International Conference on Intelligent Robots and Systems (IROS)*, pages 3525–3531, Sendei, Japan, 2004.

[30] S. Godsill, A. Doucet, and M. West. Maximum a Posteriori sequence estimation using Monte Carlo particle filters. *Annals of the Institute of Statistical Mathematics*, 53:82–96, 2001.

[31] Point Grey. Bumblebee2 stereo sensor, August 2009. `www.ptgrey.com/products/stereo.asp`.

[32] E. Guizzo. Three Engineers, Hundreds of Robots, One Warehouse. *IEEE Spectrum*, 45:26–34, 2008.

[33] F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, F. Gustafsson, and P. Nordlund. Particle filters for positioning, navigation, and tracking. *IEEE Transactions on Signal Processing*, 50:425–437, 2002.

[34] J. Gutmann. Markov-Kalman localization for mobile robots. In *Proceedings of the 16th International Conference on Pattern Recognition.*, volume 2, pages 601–604, Québec City, QC, 2002.

[35] M. Hadwiger, J. M. Kniss, C. Rezk-salama, D. Weiskopf, and K. Engel. *Real-time Volume Graphics*. A. K. Peters Ltd., Natick, MA, 2006.

[36] M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. H. Gross. Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum*, 24:303–312, 2005.

[37] A. Hogue. *SensorSLAM: an investigation into the utilization of sensor parameters within the SLAM framework*. PhD thesis, York University, Department of Computer Science and Engineering, 2008.

[38] A. Hogue and M. Jenkin. Development of an underwater vision sensor for 3D reef mapping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5351–5356, Beijing, China, 2006.

[39] InterSense. Inertiacube 3, August 2009. `intersense.com`.

[40] iRobot Inc. Roomba vacuum cleaner, August 2009. `www.irobot.com`.

[41] C. L. Jackins and S. L. Tanimoto. Oct-trees and their use in representing three-dimensional objects. *Computer Graphics and Image Processing*, 14:249–270, 1980.

[42] M. Jenkin, A. Hogue, A. Gerrnan, S. Gill, A. Topol, and S. Wilson. Underwater surface recovery and segmentation. In *IEEE International Conference on Cognitive Informatics*, pages 373–380, Los Alamitos, CA, 2007.

[43] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[44] M. Klaas, M. Briers, N. de Freitas, A. Doucet, S. Maskell, and D. Lang. Fast particle smoothing: if I had a million particles. In *Proceedings of the 23rd International Conference on Machine Learning (ICML)*, pages 481–488, New York, NY, 2006.

[45] Y. Ma, Z. Wang, M. Bazakos, and W. Au. 3D scene modeling using sensor fusion with laser range finder and image sensor. In *Proceedings of the 34th Applied Imagery and Pattern Recognition Workshop (AIPR)*, pages 224–229, Washington, DC, 2005.

[46] M. Magnusson. *The Three-Dimensional Normal-Distributions Transform an Efficient Representation for Registration, Surface Analysis, and Loop Detection*. PhD thesis, Örebro University, Örebro, Sweden, 2009.

[47] 3D Laser Mapping. LMS-Z420i, August 2009. `www.3dlasermapping.com`.

[48] J. L. Marins, X. Yun, E. R. Bachmann, R. McGhee, and M. J. Zyda. An extended Kalman filter for quaternion-based orientation estimation using MARG sensors. In *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2003–2011, Maui, Hawaii, 2001.

[49] J. Meneely. Lidar based kiln map. Lidar model of an archaeological site, 2010.

[50] A. Milstein, J. Sánchez, and E. Williamson. Robust global localization using clustered particle filtering. In *Eighteenth national conference on Artificial intelligence (AAAI)*, pages 581–586, Menlo Park, CA, 2002.

[51] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: a factored solution to the simultaneous localization and mapping problem. In *Eighteenth national conference on Artificial intelligence (AAAI)*, pages 593–598, Menlo Park, CA, 2002.

[52] R. Negenborn. Robot localization and Kalman filters. Master's thesis, Utrecht University, Utrecht, The Netherlands, September 2003.

[53] W. Straber P. Biber. The normal distribution transform: a new approach to laser scan matching. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 2743–2748, Las Vegas, NV, 2003.

[54] S. Panzieri, F. Pascucci, and G. Ulivi. An outdoor navigation system using GPS and inertial platform. In *Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, volume 2, pages 1346–1351, Como, Italy, 2001.

[55] B. Parkinson and J. Spilker, editors. *Global Positioning System: Theory & Applications. Progress in Astronautics and Aeronautics*, volume 1. AIAA (American Institute of Aeronautics and Astronautics), New York, NY, January 1996.

[56] S.H. Pourtakdoust and G. H. Asl. An adaptive unscented Kalman filter for quaternion-based orientation estimation in low-cost Attitude and Heading Reference Systems. *Aircraft Engineering and Aerospace Technology: An International Journal*, 79:485–493, 2007.

[57] Apricot Open Game Project. Yo frankie! `http://www.yofrankie.org`.

[58] I. Rekleitis. A particle filter tutorial for mobile robot localization. Technical Report TR-CIM-04-02, Centre for Intelligent Machines, McGill University, Montreal, QC, 2004.

[59] A.M. Sabatini. Quaternion-based extended kalman filter for determining orientation by inertial and magnetic sensing. *IEEE Transactions on Biomedical Engineering*, 53:1346–1356, 2006.

[60] S. Saha, Y. Boers, J. N. Driessen, P. K. Mandal, and A. Bagchi. Particle filter based map state estimation: A comparison. In *Proceedings of 12th International Conference on Information Fusion 2009*, pages 278–283, Seattle, WA, 2009.

[61] D. Salmond, N. Gordon, and A. Smith. Novel approach to nonlinear/non-gaussian Bayesian state estimation. *IEEE Proceedings for Radar and Signal Processing*, 140:107–113, 1993.

[62] A. Schultz and W. Adams. Continuous localization using evidence grids. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 2833–2839, Lueven, Belgium, 1998.

[63] R. Sim and G. Dudek. Learning and evaluating visual features for pose estimation. In *Proceedings of the Seventh International Conference on Computer Vision (ICCV)*, pages 1217–1222, Kerkyra, Greece, 1999.

[64] H. W. Sorenson, editor. *Kalman Filtering: Theory and Application*. IEEE Press, Cambridge, MA, 1985.

[65] T. Suzuki, Y. Amano, and T. Hashizume. 6DOF localization for a mobile robot using outdoor 3D point clouds. *Journal of Robotics and Mechatronics*, 22:158:166, 2010.

[66] S. Thrun. Probabilistic Algorithms in Robotics. *AI Magazine*, 21:93–109, 2000.

[67] S. Thrun, F. Dellaert, W. Burgard, and D. Fox. Particle filters for mobile robot localization. In A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*. Springer Verlag, New York, NY, 2000.

[68] A. Topol, M. Jenkin, J. Gryz, S. Wilson, M. Kwietniewski, P. Jasiobedzki, H. Ng, and M. Bondy. Generating semantic information from 3D scans of crime scenes. In *Proceedings of the 2008 Canadian Conference on Computer and Robot Vision (CRV)*, pages 333–340, Windsor, Ontario, 2008. IEEE Computer Society.

[69] York University. York university rover team, 2009. `www.yuroverteam.com`.

[70] P. Vernaza and D. Lee. Rao-Blackwellized particle filtering for 6-DOF estimation of attitude and position via GPS and inertial sensors. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1571–1578, Orlando, Florida, 2006.

[71] J. Van Verth and L. Bishop. *Essential Mathematics for Games and Interactive Applications: A Programmer's Guide*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 2004.

[72] Vicon. Vicon motion capture systems, 2010. `www.vicon.com`.

[73] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269, 1967.

[74] J. R. Wertz, editor. *Spacecraft Attitude Determination and Control*, volume 73. D. Reidel Publishing Company, Dordrecht, Holland, 1978.

[75] P. Zarchan and H. Musoff. *Fundamentals of Kalman Filtering: A Practical Approach (Progress in Astronautics and Aeronautics)*. AIAA (American Institute of Aeronautics and Astronautics), 2000.

[76] R. Zhu, D. Sun, Z. Zhou, and D. Wang. A linear fusion algorithm for attitude determination using low cost MEMS-based sensors. *Measurement*, 40:322–328, 2007.