Optimizing Robot Trajectories using Reinforcement Learning

by

Thomas Kollar

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

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2007
[June 2007]

Author ....
[illegible]
Department of Electrical Engineering and Computer Science
February 2, 2007

Certified by ....
[illegible]
Nicholas Roy
Assistant Professor of Aeronautics and Astronautics
Thesis Supervisor

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

Optimizing Robot Trajectories using Reinforcement Learning

by

Thomas Kollar

Submitted to the Department of Electrical Engineering and Computer Science
on February 2, 2007, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

The mapping problem has received considerable attention in robotics recently. Mature techniques now allow practitioners to reliably and consistently generate 2-D and 3-D maps of objects, office buildings, city blocks and metropolitan areas with a comparitively small number of errors. Nevertheless, the ease of construction and quality of map are strongly dependent on the exploration strategy used to acquire sensor data.

Most exploration strategies concentrate on selecting the next best measurement to take, trading off information gathering for regular relocalization. What has not been studied so far is the effect the robot controller has on the map quality. Certain kinds of robot motion (e.g, sharp turns) are hard to estimate correctly, and increase the likelihood of errors in the mapping process. We show how reinforcement learning can be used to generate better motion control. The learned policy will be shown to reduce the overall map uncertainty and squared error, while jointly reducing data-association errors.

Thesis Supervisor: Nicholas Roy
Title: Assistant Professor of Aeronautics and Astronautics

Acknowledgments

For all his support, effort, consultations, and patience, I would like to thank my advisor Nicholas Roy. For their belief in education, humanity, and for their support through times that are good and bad, I fould like to thank my parents. For the many productive discussions with labmates, friends, and comrades, I would like to thank you; you know who you are.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The mapping problem has received considerable attention in robotics recently. Mature mapping techniques now allow practitioners to reliably and consistently generate 2-D and 3-D maps of objects, office buildings, city blocks and metropolitan areas with a comparatively small number of errors. Nevertheless, the ease of construction and quality of map are strongly dependent on the exploration strategy used to acquire sensor data.

Most exploration strategies concentrate on selecting the next best measurement to take, trading off information gathering for regular relocalization. What has not been studied so far is the effect the robot controller has on the map quality. Certain kinds of robot motion (e.g, sharp turns) are hard to estimate correctly, and increase the likelihood of errors in the mapping process. We show how reinforcement learning can be used to generate better motion control. The learned policy will be shown to reduce the overall map uncertainty and squared error, while jointly reducing data-association errors.

1.1 Overview

Mobile robots are becoming increasingly capable both in theory and in practice. In theory, robots could already perform nearly innumerable useful tasks, but bringing theory to practice has proved difficult. The complexity of long-term execution, state estimation, planning, learning, and decision-making under uncertainty have prevented the actual implementation of more general-purpose robotic systems.

In practice the application domains of robots have been impressive, but constrained. Although robots may now build cars, explore Mars, defuse bombs, perform reconnaissance, execute surgeries and search for victims of catastrophic events; many of these applications today involve a human controller. Though value can nevertheless be achieved with a human in the loop, many applications will necessitate a high degree of autonomy. Today autonomous robots can drive long distances across the desert, clean household floors, guide museums, mow the grass, and deliver mail. With few exceptions, however, the environment has been constrained as much as possible to prevent failure.

The failure of robots at certain tasks is inevitable, but a certain amount of flexibility in the set of actions a robot is able to perform can prevent complete failure. If a robot can simulate that it will likely get lost by moving down a particular hallway, then taking an action that will prevent this is highly desirable. By taking such actions to prevent failure, a robot will become more useful more of the time. The subject of this thesis is how to select actions that will prevent failure in mapping. It is our thesis that in general providing a self-model (simulator) to the robot will allow it to take actions that prevent modes of failure that would otherwise be difficult or impossible to detect.

1.2 Problems addressed

A particular area in which robotics has advanced significantly in the last few years is in the mapping problem. One of the remaining problems in mapping is how to choose motions during mapping that will positively impact map quality. This work focuses on how to integrate motion planning into the mapping process. By choosing trajectories during planning, we will show how to create better maps with a minimal number of errors.

The creation of a map involves three components: mapping, localization and path planning, as indicated by Figure 1.2.

1. *Mapping*: Make a map when the pose of the robot is known.
2. *Localization*: Find the robot given a map.
3. *Path planning*: Follow a trajectory through the environment that avoids obstacles and achieves the primary and secondary goals.

Each of these may be performed independently, and solves a useful problem in robotics. When a robot's pose is known precisely, mapping will be useful to identify obstacles and generate motion plans. When a map is known, then localization can provide information about the current location of the robot and given a destination can provide a path there. Given a map and a position then path planning can achieve a goal location.

Though each of these can be performed independently, the joint execution of them will provide better approximations of uncertainty, better algorithms, and in the end, better maps. Not surprisingly, the most interesting aspects of creating a map come at the intersections of these problems.

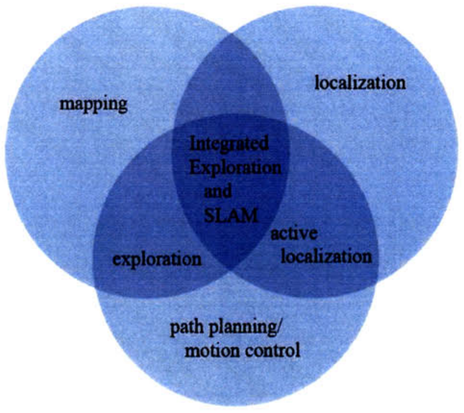
Combining path planning and localization begets *active localization*, where a robot will take actions to maximize its localizability. Combining mapping and path planning, a robot explores its world so a secondary goal is achieved (distance, time, coverage). Combining mapping and localization, a robot simultaneously estimates its position and the map by *simultaneous localization and mapping (SLAM)*. The approach presented here is at the intersection of all three of these sets and is called the *simultaneous planning, localization, and mapping (SPLAM)* problem [33], where the robot will jointly take actions to make the best map possible as well as estimate its pose and the map. In general, we will use the shorthand *exploration* to encompass *integrated exploration and SLAM*.

1.3 Exploration

Building maps is one of the most fundamental problems in robotics. As operation times become longer and less structured environments become commonplace, the destinations and motion of the robot become critical for the acquisition of high quality sensory information and high quality maps.

There are a number of objective functions that may be considered when building a map: time, power, coverage, and map accuracy are just a few. While distance and power consumed are factors that need to be taken into account, the most critical factor for many applications is acquiring a high-quality map. Optimizations with respect to time or coverage are only useful if the user will not spend hours editing poor scan alignments, pose estimates, or feature extractions, and the trajectory taken by the robot should be designed to lead to the best map in minimum time. For example, when creating 2D maps with wheeled mobile robots, certain kinds of robot motion (e.g, sharp turns) are hard to estimate correctly, and are known to increase the likelihood of errors in the mapping

Figure 1.2.1: The mapping problem [21]


mapping
localization
Integrated Exploration and SLAM
active localization
exploration
path planning/ motion control

process (see Figure 2-1(b)).

The exploration problem can be formalized as a belief-space *Markov Decision Process* (MDP), which will allow us to bring multistep policies to bear on our problem and thereby maximize long-term map accuracy. A policy for a MDP will give a mapping from robot and feature locations to an optimal action.

1.4 Contributions

The contributions of this work are four-fold. First, we will use a non-standard motion model developed in [8], and derive the relevant mean and covariance updates for the Extended Kalman Filter used for many of the experiments. Secondly, in Chapter 3 we will produce a characterization of the *trajectory exploration* problem, which to our knowledge has not been considered before in this context and will discuss the properties of the determinant and trace as optimization metrics. Third, we discuss the formalization of the *trajectory exploration* problem as a belief space Markov Decision Process, and why this formalization makes sense. Finally, we will show how to solve this problem with a particular policy search method [1] and give experimental results.

1.5 Overview / Organization of Chapters

The Extended Kalman Filter with range and bearing measurements is the simultaneous localization and mapping algorithm used in this work. A Kalman filter has been shown under a set of common assumptions, in the limit of infinite time and observations, to converge to a perfect map. The speed with which the estimated map converges to the true map is strongly dependent on how observations of the environment are taken and by the trajectory of the robot [17]. The specific assumptions and relevant properties of the EKF are discussed in Chapter 2.

Given the EKF as our starting point, we will discuss the trajectory exploration problem in Chapter 3. Though there may be a number of measures of uncertainty for each probability distribution, one of the most commonly used is related to entropy. Reducing entropy makes the values of the state variables more certain. We argue that increasing certainty about state variables is a good optimization criteria without ground truth, while squared error is a better metric when it is known. The optimization criteria will also be discussed in Chapter 3.

Though an optimization metric may be given, the exact consequences of an action cannot be predicted precisely since the motion and sensor readings from the robot are not known perfectly. The same action will generally result in different rewards due these sources of error. Thus, we will want to maximize the expected reward with respect to this optimization metric (e.g. maximize the certainty, minimize the squared error).

There does not unfortunately exist a computationally tractable way to optimize the posterior uncertainty of an extended robot trajectory (with or without integrating measurements). Our approach will therefore be to use reinforcement learning to find policies that minimize the posterior uncertainty given an ordered sequence of k future points that an exploration planner has selected for data gathering. Given this set of points, our algorithm can generate the best k step policy for visiting these points. The success of our approach depends on careful parametrization of the action space. An overview of reinforcement learning is performed in Chapter 4.

The optimal solution to this problem would involve arbitrary trajectories with arbitrary turning rates. For our solution we approximately fix the translational velocity and the class of trajectories to be cubic splines. Using reinforcement learning, any model and any controller can be used to acquire optimal trajectories. A discussion of the particular design decisions is undertaken in Chapter 4.3.

Finally, it will be shown that our reinforcement learner successfully generates motion trajectories that minimize the posterior covariance of the robot in contrast to a standard hand-tuned controller that minimizes distance. We will also show that the posterior map learned when using our policy generates more accurate maps. A discussion of the experiments is performed in Chapter 5.

Chapter 2

The Bayes Filter

Solutions to the Simultaneous Localization and Mapping (SLAM) problem are at the core of integrated exploration algorithms. In this work we focus on linear Gaussian models because of their standard use for tracking and estimation problems. When compared with other SLAM solutions, linear Gaussian models are particularly amenable to fast computation of information and related metrics, which we will show in Chapter 3.

This chapter begins first with a review of the Bayes filter. We then progress to a review of the SLAM literature and then subsequently derive the Kalman filter. We then describe the design decisions for our particular implementation of the Kalman filter: the motion model, the sensor model, and the relevant Jacobians.

2.1 Overview

The Simultaneous Localization and Mapping problem is how to estimate the robot pose and a map from noisy sensor and motion measurements. We can get an idea of how difficult the problem is by looking at Figure 2-1(a). Here, the robot's internal odometry sensor is used as the true position of the robot and the range measurements from the laser range scanner are plotted with respect to this pose.

Navigation with respect to this map would be impossible. One cannot in general find an open path between any two points, much less generate a motion plan that takes the robot between any two locations. There exist a variety of methods to recover the underlying map, each of which have their own difficulties.

In Figure 2-1(b), successive scans are matched and a map is recovered [25]. The mapping, however, has aligned a couple of scans incorrectly, thus rendering much of the map unusable. There are also a number of places where scans are roughly lined up, but where the result is far from perfect.

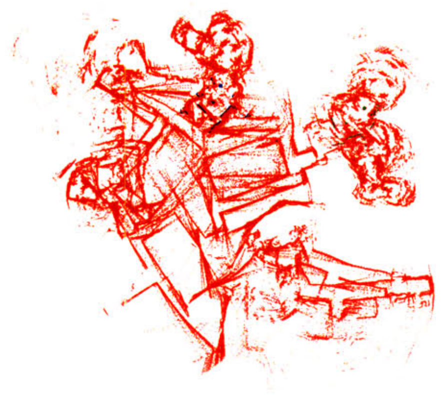
Data-association and closing the loop are two of the most difficult problems in mapping. Each laser scan is unique, but all, part or none of each scan corresponds to places in the world that have already been seen. Deciding which parts of a new scan correspond to parts already seen is the problem of *data-association.* Data-association is more pronounced when returning to a previously visited location, when the most recently mapped area may misalign with the previously generated map. Noticing that this location is being revisited and fusing the previous and current data is known as *closing the loop.*

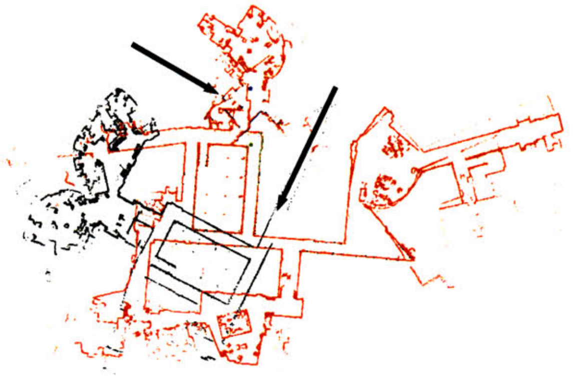
Though these errors can be manually corrected as shown in Figure 2-1(c) and Figure 2-1(d), it is our goal to mitigate or prevent them completely by choosing better motion plans for the robot. Our motion plan should jointly make better *data-associations*, make it easier to *close the loop*, take better measurements, and minimize motion error. The two places where the map misaligns are locations where the robot takes point turns; if the robot had taken a smoother path or gotten a more unique laser signature, then these errors could have been mitigated.

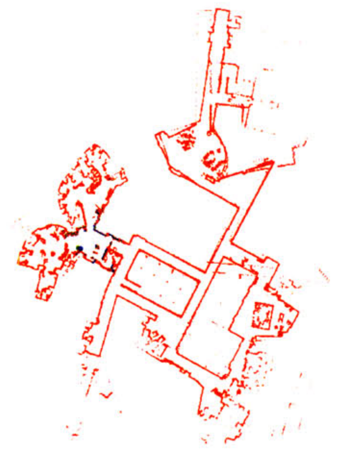
Data-association, loop-closing and motion selection all affect the quality of the resulting map. With integrated exploration we hope to mitigate this problem. The issue of exploration and its affect on the resulting map will be discussed at length in Chapter 3. In this chapter, the EKF will be derived from a Hidden Markov Model and the particular parameters for an EKF that performs SLAM will be elaborated (motion model, measurement model, Jacobians... etc).

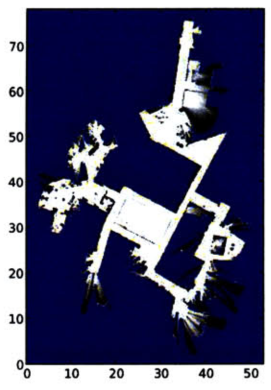
2.2 The Bayes Filter

There is an abundance of work on performing exact and approximate inference in general graphical models [41, 16]. Similar attention has been given to Gaussian graphical models, which has recently been a very active area of research [39, 36, 5, 27, 28]. In this section, we will develop the Bayes filter as a special form of hidden state inference. In following sections we will further specialize the


(a) Raw Measurements


(b) Mapping Errors


(c) Corrected Measurements

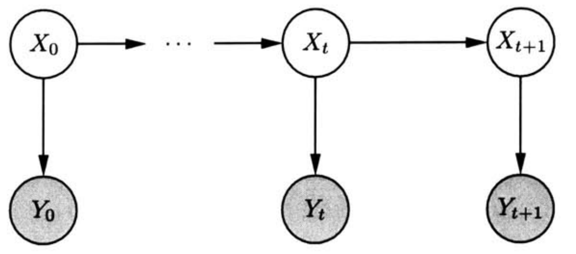


(d) Final Map

Figure 2.1.1: Simultaneous Localization and Mapping. In (a)-(c), the black dots are laser range points projected from the estimated or actual robot pose. In (a), the robot odometry is used. In (b) CARMEN is used to estimate the location. In (c), the robot pose and measurements are corrected. In (d), the resulting map is shown: black is occupied, blue is unknown, and white is free.

Bayes filter to the Kalman Filter, which has garnered wide attention in the SLAM community.

Figure 2.2.1: The Graphical Model for the Hidden Markov Model



To start with, we will define the graphical model shown in Figure 2.2.1. The graph encodes the Markov assumption that given X_k the future timesteps and the previous timesteps are independent. Further, using the independence assumptions of the graph, we can obtain the joint probability distribution as:

$$\begin{aligned} P(X_{0:t}, Y_{0:t}) &= P(Y_t|X_{0:t}, Y_{0:t-1})P(X_t|X_{0:t-1}, Y_{0:t-1})P(Y_{t-1}|X_{0:t-1}, Y_{0:t-2}) \\ &\qquad P(X_{t-1}|X_{0:t-2}, Y_{0:t-2}) \ldots P(Y_1|X_1, Y_0)P(X_1|X_0, Y_0)P(Y_0|X_0)P(X_0) \\ &= P(Y_t|X_t)P(X_t|X_{t-1})P(Y_{t-1}|X_{t-1})P(X_{t-1}|X_{t-2}) \ldots P(Y_1|X_1)P(X_1|X_0)P(X_0) \end{aligned}$$

There are many ways inference can be performed in this model, depending on what result we want to obtain. We could intend to obtain a full inference over the hidden variables, which would correspond to the full trajectory of the robot and the full set of landmark positions. This inference is often called Smoothing and Mapping (SAM) [5]. If the X_i are discrete variables, then this reduces to the Viterbi algorithm for hidden Markov models. Alternatively, an inference can be obtained at any particular time by marginalizing out the unused variables.

If we assume that the algorithm will run online and that the state is represented as the robot pose and the map, a reasonable inference is the state at the latest time. This is the inference performed by a Bayes filter. To understand how perform this inference, we will compute quantity: $P(X_t|Y_{0:t})$. Note that there is a normalization constant that we will ignore for the moment. First, we expand using Bayes rule so that $P(X_t|Y_{0:t}) = P(X_t, Y_{0:t})/P(Y_{0:t})$. For now, we will call $1/P(Y_{0:t}) = Z$.

$$\begin{aligned} P(X_t|Y_{0:t}) \quad &= Z\, P(Y_t|X_t) \int_{X_{t-1}} P(X_t|X_{t-1}) P(Y_{t-1}|X_{t-1}) \int_{X_{t-2}} P(X_{t-1}|X_{t-2}) \\ &\quad \ldots \int_{X_1} P(X_2|X_1) P(Y_1|X_1) \int_{X_0} P(X_1|X_0) P(X_0) dX_{0:t-1} \end{aligned} \tag{2.2.1}$$

Because of the independence assumptions made in the filter design, we can push the integrals very far into the relation. This is called the Bayes filter and can be seen in Figure 2.2.2.

Figure 2.2.2: The Bayes Filter

- Given: $bel(X_{t-1})$, $P(X_0)$
- Time Projection: $\overline{bel}(X_t) = \int_{X_{t-1}} P(X_t|X_{t-1}) bel(X_{t-1})$
- Measurement Update: $bel(X_t) = Z\, P(Y_t|X_t)\overline{bel}(X_t)$

The Bayes filter will start at an initial time t_0 and will recursively compute an estimate of the state at each time t. It can be seen in Figure 2.2.2 that two updates are performed: a *motion* (time-projection) and *measurement* update. The motion update corresponds to moving the robot and introducing uncertainty into its position. The measurement update corresponds to incorporating new sensory feedback: laser, vision or otherwise, thus reducing the uncertainty in the landmark positions and often the robot position. It remains for us to prove that the Bayes filter corresponds to Equation 2.2.1.

Theorem 2.2.1. *The Bayes filter algorithm corresponds to the computation of the distribution shown in Equation 2.2.1 for t timesteps.*

Proof. The proposition that we are trying to prove is that the distribution at time t is equal to the probability distribution described in Equation 2.2.1. We use proof by induction.

Note first that the base case is true since $P(X_0)$ is the base case of the Bayes filter algorithm and corresponds to $P(X_0|\phi)$ Equation 2.2.1. Assume that the Bayes filter algorithm is equivalent to the equation at time $t-1$. It remains for us to show that the two are equivalent at time t. From the inductive hypothesis, we know that:

$$\begin{aligned} bel(X_{t-1}) = P(X_{t-1}|Y_{0:t-1}) &= Z\,P(Y_{t-1}|X_{t-1}) \int_{X_{t-2}} P(X_{t-1}|X_{t-2}) \\ &\quad \dots \int_{X_1} P(X_2|X_1)P(Y_1|X_1) \int_{X_0} P(X_1|X_0)P(X_0)dX_{0:t-1} \end{aligned}$$

But then by the Bayes filter algorithm:

$$bel(X_t) = \nu P(Y_t|X_t) \int_{X_{t-1}} P(X_t|X_{t-1})bel(X_{t-1})$$

Substituting the previous expression in for $bel(X_{t-1})$, we can compare to the above equation and see that the two are the same. Since this proposition is true at time t, then the proposition is true for all $t \in \mathbb{N}$. Q.E.D. $\square$

2.3 Related Work

SLAM has received a wealth of attention recently and includes two components of any integrated exploration: localization and mapping. Nearly all of the SLAM solutions can be formulated as a Bayes filter.

There are a number of assumptions that one may make when deciding on a SLAM algorithm. When the measurement and motion uncertainty and noise are Gaussian and the motion and measurement updates are linear functions of the previous state and measurements (respectively), then efficient algorithms can be derived for recursively updating the mean and covariance. The Kalman filter is one instance of such algorithms [34]. When linearizing non-linear measurement and motion update functions, the Extended Kalman filter can be derived to obtain approximations of the posterior covariance [34].

Simply linearizing the motion and measurement models does not always give a conservative estimate of the covariance when the motion and measurement updates are highly nonlinear. One way around this is to sample the prior distribution $P(X_{t-1})$ and propagate the samples through the non-linear update functions and then re-estimate the covariance. This is called the *Unscented Kalman Filter.* The computational complexity for the general instantiation of the Kalman filter is $O(n^3)$ [38], where n is the number of state variables. However, when looking at specific problems, the complexity is often much better.

Alternatives to the EKF exist, many of which may perform the same or similar inference much faster, with the tradeoff that they may not be able to run online for large datasets or may not be exact. While in the EKF the inference is recursively computed, it also happens that *all* of the variables become highly correlated and the independancies that exist in the observations are quickly lost. As a result the covariance becomes dense and updates are computationally expensive. This property has recently been exploited by using the information form of the filter [36, 9]. Most of this information matrix (inverse of the covariance) is sparse, which allows updates to be done more efficiently by setting certain values to zero. By doing this in a principled way, [9, 35] obtain very fast measurement and motion updates to the state.

Another optimization that can be performed is to choose a good ordering in which to eliminate variables [5]. While this method may not necessarily be an online technique for large datasets, it nevertheless offers significant speedups in inference. By performing the variable ordering in a clever manner, orders of magnitude can be removed from the time necessary to integrate the measurements and generate a map. Often the whole path of the robot is desired, in in which case this becomes a Smoothing and Mapping (SAM) problem.

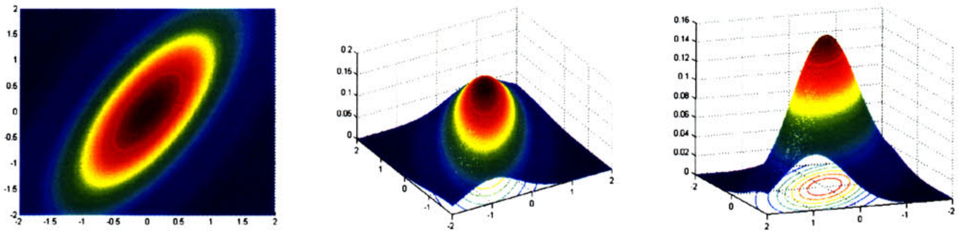
Less related to the above techniques is to take a network of pose relations and perform an optimization to compute the optimal set of robot poses. Fast optimization on the pose graph can also create very fast initial pose estimates and also high quality maps [27, 7, 20].

A final technique is called FastSLAM, where the posterior distribution is a Rao-Blackwellized particle filter over robot poses and maps. Each particle corresponds to a sampled trajectory and map. Particles are propagated according to the motion model and weighted by the likelihood of the resulting map. Data association and loop-closing still remain problems; associating data becomes matching laser scans and loop-closing becomes consistently rectifying the previous map. Though the first version of *FastSLAM* used a landmark-based approach [24], more recent versions track particles containing full gridmaps[14].

2.4 The Extended Kalman Filter

The approach chosen here is an Extended Kalman filer because of its standard use for tracking and estimation [23, 34]. The Kalman filter is one form of the Bayes filter, where $P(X_t|X_{t-1})$

Figure 2.4.1: A 2D Gaussian distribution



and $P(Y_t|X_t)$ are Gaussian for all t. When these are Gaussian and the state update is linear or a linearized function then it is possible to show that the posterior belief at the next timestep is Gaussian. The two strongest assumptions for the Kalman filter are linearity and Gaussianity; we start by defining the latter. A Gaussian distribution is shown in Figure 2.4.1.

Definition 2.4.1. Letting μ be a vector and Σ be a positive definite covariance matrix and $\mathbf{x} = (x_1, \ldots, x_n)$, then the *multivariate Normal Distribution* or *multivariate Gaussian Distribution* is defined as:

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{N}{2}} |\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1} (\mathbf{x}-\mu)}$$

For the Kalman filter, there are two kinds of updates that we anticipate: *measurement* and *motion*. The function g will describe how to update all to new state variables given the current state variables. The function h will describe how to map to a measurement from the state variables, so that we can either associate it with current state variables or create new state variables if necessary.

There is, however, noise introduced into each measurement and motion update. This noise must be Gaussian in nature and will be defined as $\epsilon_t \sim N(\mu_\delta, R)$ and $\delta_t \sim N(\mu_\epsilon, W)$, which are the noise for the motion and measurement, respectively.

In general, we will need to linearize g and h. The standard way is via Taylor expansion. At this point, we linearize around the mean at each previous value, which will be the maximum likelihood estimate at the previous time, and is thus a reasonable point to linearize about. In particular, the new state is computed as:

$$\begin{aligned} X_t &= g(u_t, X_{t-1}) + \epsilon_t && (2.4.1) \\ &\approx g(u_t, \mu_{t-1}) + \frac{\partial g}{\partial X_{t-1}}(X_{t-1} - \mu_{t-1}) + \epsilon_t && (2.4.2) \\ &= g(u_t, \mu_{t-1}) + G_t(X_{t-1} - \mu_{t-1}) + \epsilon_t && (2.4.3) \end{aligned}$$

Note here that g and h are in general vector valued as is X_t and X_{t-1}. When both are vector valued, then these partial derivatives become the Jacobian matrix. The Jacobian of g with respect to X_{t-1} is G and the Jacobian of h with respect to X_t is H_t. Thus, the measurement function is defined as:

$$\begin{aligned} Y_t &= h(X_t) + \delta_t && (2.4.4) \\ &\approx h(\mu_t) + \frac{\partial h}{\partial X_t}(X_t - \mu_t) + \delta_t && (2.4.5) \\ &= h(\mu_t) + H_t(X_t - \mu_t) + \delta_t && (2.4.6) \end{aligned}$$

We now have the tools to discuss the Extended Kalman Filter, as stated in Figure 2.4.2. In the first step, we are given the mean and covariance at the previous timestep. From these, we will compute the new mean and covariance after the time projection step: $\bar{\mu}$ and $\bar{\Sigma}$. We then compute what is called the Kalman gain K, which says how much to believe the measurements. Finally, we compute the new mean and covariance μ_t and Σ_t from the Kalman gain K. The following theorem encapsulates the assumptions and the truth of Figure 2.4.2. To preserve the flow, a proof of this statement is given in the Appendix.

Theorem 2.4.2. *If $P(X_t|X_{t-1})$, $P(Y_t|Y_{t-1})$ and the prior distribution over X_0 are all Gaussian and the X_t is a* linearized *function of X_{t-1} and Y_t is a linearized function of X_t for all t, then $bel(X_t)$ for all t will be Gaussian. Further, the updates are as shown as in Figure 2.4.2.*

Proof. Shown in the Appendix. □

For the above algorithm it is assumed that each measurement has already been associated with a subset of the state (i.e. a specific landmark position l_x, l_y). However, associating a measurement to a set of state variables is nevertheless a difficult problem. In the case of SLAM one obvious

Figure 2.4.2: The Extended Kalman Filter

- Given μ_{t-1}, Σ_{t-1}, u_t, z_t
- $\overline{\mu}_t = g(u_t, \mu_{t-1})$
- $\overline{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
- $K_t = \overline{\Sigma}_t H_t^T (H_t \overline{\Sigma}_t H_t^T + Q_t)^{-1}$
- $\mu_t = \overline{\mu_t} + K_t(z_t - h(\overline{\mu_t}))$
- $\Sigma_t = (I - K_t H_t)\overline{\Sigma_t}$

option is the minimum Euclidean distance to a landmark mean. While this works, it can be brittle. Another option is the minimum Mahalanobis distance to the portions or all of the state, which takes into account the uncertainty in the measurement [26].

Definition 2.4.3. For mean μ and covariance matrix Σ, the *Mahalanobis distance* is defined as:

$$D_M(x) = \sqrt{(x-\mu)^T \Sigma^{-1} (x-\mu)}$$

In all applications of a Kalman filter, a reasonable prior distribution for the initial state X_0 is decided using prior knowledge about the problem. Typically for SLAM the covariance matrix for the pose variables are set to zeros, reflecting the fact that the coordinate system is arbitrary and that the initial pose in it should have no uncertainty.

2.5 An Extended Kalman Filter for SLAM

There are a number of design decisions that one needs to make when implementing a specific Kalman filter. These include the choice of state, measurement and motion models, a data association algorithm, and the means by which the state is expanded are added when new state features (e.g. new landmarks) are encountered.

2.5.1 State

There are a number of ways in which we might define our state variables. One option is to place the robot at the center of the coordinate system and update a number of landmark locations. Instead

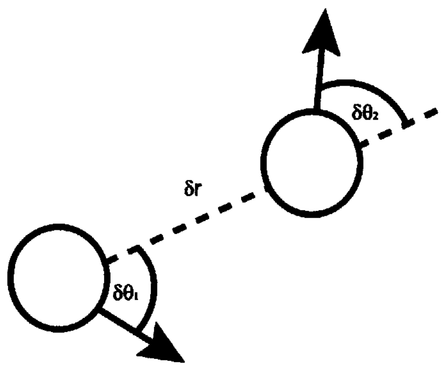


Figure 2.5.1: Robot Motion Geometry

of doing this, the map will be stored in global coordinates so that the robot pose $r_p = (r_x, r_y, r_\theta)$ as well as the x, y, and a signature s for each landmark, $l_i = (l_x, l_y, l_s)$ will all be tracked. The full state is, in global coordinates:

$$X_t = (r_x, r_y, r_\theta, l_{1,x}, l_{1,y}, l_{1,s}, \dots l_{n,x}, l_{n,y}, l_{n,s})$$

2.5.2 Robot Motion

The model g describing the motion of the robot, along with the noise model is a critical part of the optimization that will be performed, and therefore is chosen to be as general as possible. If the model poorly reflects the actual motion or error, the state inference and trajectory optimization could provide poor solutions.

It will be assumed that the robot will give some notion of odometry, either a relative velocity or a relative change in pose. Further, the robot motion will be parametrized by an initial rotation, a translation and a final rotation: $d\theta_1$, dr, and $d\theta_2$, respectively. This parametrization can be seen in Figure 2.5.2.

There are two ways in which the readings may arrive to the robot: as relative *position* or *velocity*. Velocity readings give the translational velocity v and the rotational velocity ω of the robot and a time over which the velocities were executed dt. Position readings give the change in position: Δx, Δy, $\Delta\theta$.

$$\begin{array}{ll} \textit{velocity} & \begin{pmatrix} d\theta_1, & dr, & d\theta_2 \end{pmatrix}^T = \begin{pmatrix} \omega\frac{dt}{2}, & v\,dt, & \omega\frac{dt}{2} \end{pmatrix}^T \\ \textit{position} & \begin{pmatrix} d\theta_1 & dr & d\theta_2 \end{pmatrix}^T = \begin{pmatrix} atan2(\Delta y, \Delta x), & \sqrt{(\Delta x)^2 + (\Delta y)^2}, & \Delta\theta - d\theta_1 \end{pmatrix}^T \end{array}$$

In this work, the motion model we use is more general than those usually implemented [34]. The way this model differs is that it allows the robot to slip as well as translate and rotate. Slip means that the vehicle translates in the direction perpendicular to the direction of motion.

Definition 2.5.1. Assuming that the slip of the robot is ds, that $X_{t-1} = (r_{x,t-1}, r_{y,t-1}, r_{\theta,t-1}, \ldots)$, and that the motion is contained in the control $u_t = (d\theta_1, dr, d\theta_2)$, then the *standard motion model with slip* can be defined as:

$$g(X_{t-1}, u_t) = \begin{pmatrix} r_{x,t} \\ r_{y,t} \\ r_{\theta,t} \end{pmatrix} = \begin{pmatrix} r_{x,t-1} + dr\,cos\,(r_{\theta,t-1} + d\theta_1) + ds\,cos\,(r_{\theta,t-1} + d\theta_1 + \frac{\pi}{2}) \\ r_{y,t-1} + dr\,sin\,(r_{\theta,t-1} + d\theta_1) + ds\,sin\,(r_{\theta,t-1} + d\theta_1 + \frac{\pi}{2}) \\ r_{\theta,t-1} + d\theta_1 + d\theta_2 \end{pmatrix}$$

Now that we have the motion function g, we must specify the Jacobian of g with respect to the state variables:

$$G_t = \begin{pmatrix} \frac{\partial g_1}{\partial x} & \frac{\partial g_1}{\partial y} & \frac{\partial g_1}{\partial z} & \frac{\partial g_1}{\partial l_1} & \cdots & \frac{\partial g_1}{\partial l_n} \\ \frac{\partial g_2}{\partial x} & \frac{\partial g_2}{\partial y} & \frac{\partial g_2}{\partial z} & \frac{\partial g_2}{\partial l_1} & \cdots & \frac{\partial g_2}{\partial l_n} \\ \frac{\partial g_3}{\partial x} & \frac{\partial g_3}{\partial y} & \frac{\partial g_3}{\partial z} & \frac{\partial g_3}{\partial l_1} & \cdots & \frac{\partial g_3}{\partial l_n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial g_{l_n}}{\partial x} & \frac{\partial g_{l_n}}{\partial y} & \frac{\partial g_{l_n}}{\partial z} & \frac{\partial g_{l_n}}{\partial l_1} & \cdots & \frac{\partial g_{l_n}}{\partial l_n} \end{pmatrix} = \begin{pmatrix} 1 & 0 & a & 0 & \ldots & 0 \\ 0 & 1 & b & 0 & \ldots & 0 \\ 0 & 0 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & 1 \end{pmatrix}$$

a and b are computed as:

$$\begin{aligned} a &= \frac{\partial g_1}{\partial \theta} = \frac{\partial}{\partial \theta} dr\,cos(\theta_s + d\theta) + ds\,cos(\theta_s + d\theta + \frac{\pi}{2}) \\ &= -(v\Delta t)\,sin(\theta_s + \omega\,\frac{\Delta t}{2}) - \mu_{ds}\,sin(\theta_s + \omega\frac{\Delta t}{2} + \frac{\pi}{2}) \end{aligned}$$

$$
\begin{aligned}
b &= \frac{\partial g_2}{\partial \theta} = \frac{\partial}{\partial \theta} dr\, sin(\theta_s + d\theta) + ds\, sin(\theta_s + d\theta + \frac{\pi}{2}) \\
&= (v\Delta t)\, cos(\theta_s + \omega\,\frac{\Delta t}{2}) + \mu_{ds}\, sin(\theta_s + \omega\frac{\Delta t}{2} + \frac{\pi}{2})
\end{aligned}
$$

To map from the error variables to the state space, we must describe how our low level measurements dr, $d\theta$ and ds change error in the state. This can be done with the Jacobian of g with respect to the error variables. Thus, the Jacobian of the error matrix will now be computed:

$$
\begin{aligned}
V_t &= \begin{pmatrix}
\frac{\partial g_1}{\partial\, dr} & \frac{\partial g_1}{\partial\, d\theta} & \frac{\partial g_1}{\partial\, ds} \\
\frac{\partial g_2}{\partial\, dr} & \frac{\partial g_2}{\partial\, d\theta} & \frac{\partial g_2}{\partial\, ds} \\
\frac{\partial g_3}{\partial\, dr} & \frac{\partial g_3}{\partial\, d\theta} & \frac{\partial g_3}{\partial\, ds} \\
\vdots & \vdots & \vdots \\
\frac{\partial g_{1_n}}{\partial\, dr} & \frac{\partial g_{1_n}}{\partial\, d\theta} & \frac{\partial g_{1_n}}{\partial\, ds}
\end{pmatrix} \\
&= \begin{pmatrix}
cos(\theta_s + d\theta) & -dr\, sin(\theta_s + d\theta) - ds\, sin(\theta_s + d\theta + \frac{\pi}{2}) & cos(\theta_s + d\theta + \frac{\pi}{2}) \\
sin(\theta_s + d\theta) & dr\, cos(\theta_s + d\theta) + ds\, cos(\theta_s + d\theta + \frac{\pi}{2}) & sin(\theta_s + d\theta + \frac{\pi}{2}) \\
0 & 2 & 0 \\
\vdots & \vdots & \vdots \\
0 & 0 & 0
\end{pmatrix}
\end{aligned}
$$

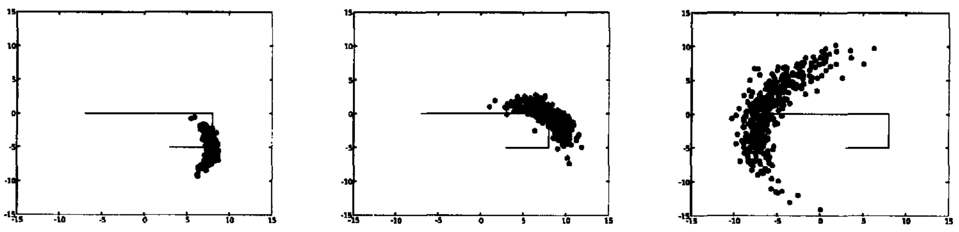
We are now in the position to specify the state update for the mean and covariance. Since everything is in a global coordinate system and only the robot pose changes, only the robot pose (x_{t-1}, y_{t-1} and θ_{t-1}) needs to be updated. Thus, we will compute the new mean $\bar{\mu}_t$ and covariance $\bar{\Sigma}_t$ as follows:

$$
\bar{\mu}_t = \mu_{t-1} + F_x^T \begin{pmatrix}
dr\, cos(\theta_{t-1} + d\theta) + ds\, cos(\theta_{t-1} + d\theta + \frac{\pi}{2}) \\
dr\, sin(\theta_{t-1} + d\theta) + ds\, sin(\theta_{t-1} + d\theta + \frac{\pi}{2}) \\
2\, d\theta
\end{pmatrix}, F_x = \begin{pmatrix}
1 & 0 & 0 & \dots & 0 \\
0 & 1 & 0 & \dots & 0 \\
0 & 0 & 1 & \dots & 0
\end{pmatrix}
$$

$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + V_t R_t V_t^T$

Recalling Figure 2.4.2, given the motion noise covariance R, then we are able to compute the

Figure 2.5.2: Samples from the motion model



motion update step of the Extended Kalman Filter. The computation of this matrix, which will change depending on the motion that the robot takes, is the subject of the next section.

2.5.3 Probabilistic Motion Model

In general the motion of the robot will not be zero-mean and the error will depend on how far and how fast the robot rotates and translates. In this section, we show a particular noise model and how to estimate the mean and variance of each of these based on the actual rotation and translation of the robot.

For our purposes here, we will assume that the measured translation and rotation of the robot jointly contribute to the actual rotation, translation and slip of the robot for a total of six motion model noise terms. We will build these six different noise terms as independent univariate Gaussian distributions. This assumption makes sense since given the given information about the turns, the translational velocity or the slip, in principle nothing can be known about the the other variables. Each column corresponds to the two components that contribute to the noise in the three parameters we are after: *ds*, *dr*, *dθ*. For example, $m_{r,r}$ and $m_{r,\theta}$ correspond to the contributions of translational and rotational motion to *dr*.

$$\begin{array}{lll} m_{r,r} \sim N(\alpha_{r,r}, \beta^2_{r,r}) & m_{\theta,r} \sim N(\alpha_{\theta,r}, \beta^2_{\theta,r}) & m_{s,r} \sim N(\alpha_{s,r}, \beta^2_{s,r}) \\ m_{r,\theta} \sim N(\alpha_{r,\theta}, \beta^2_{r,\theta}) & m_{\theta,\theta} \sim N(\alpha_{\theta,\theta}, \beta^2_{\theta,\theta}) & m_{s,\theta} \sim N(\alpha_{s,\theta}, \beta^2_{s,\theta}) \end{array}$$

We can now state the error model over *dr*, *dθ*, and *ds*.

Table 2.1: Parameters for our learned motion model.

$\alpha_{r,r} = 1.0065$	$\alpha_{r,\theta} = -0.0072$	$\alpha_{\theta,r} = 0.0144$	$\alpha_{\theta,\theta} = 0.8996$	$\alpha_{s,r} = -0.0182$	$\alpha_{s,\theta} = -0.105$
$\beta^2_{r,r} = 0.0932$	$\beta^2_{r,\theta} = 0$	$\beta^2_{\theta,r} = 0$	$\beta^2_{\theta,\theta} = 0.3699$	$\beta^2_{s,r} = 0$	$\beta^2_{s,\theta} = 0.0612$

$$\begin{aligned} \mu_{z,t} &= (\mu_{dr}, \mu_{d\theta}, \mu_{ds})^T \\ &= (\alpha_{r,r} dr + \alpha_{r,\theta} d\theta, \quad \alpha_{\theta,r} dr + \alpha_{\theta,\theta} d\theta, \quad \alpha_{s,r} dr + \alpha_{s,\theta} d\theta)^T \end{aligned}$$

$$\begin{aligned} R_{z,t} &= diag \quad (\sigma^2_{dr}, \sigma^2_{d\theta}, \sigma^2_{ds})^T \\ &= diag \quad (\beta^2_{r,r}|dr|^2 + \beta^2_{r,\theta}|d\theta|^2, \quad \beta^2_{\theta,r}|dr|^2 + \beta^2_{\theta,\theta}|d\theta|^2, \quad \beta^2_{s,r}|dr|^2 + \beta^2_{s,\theta}|d\theta|^2)^T \end{aligned}$$

We still have the goal of estimating accurate values of α and β, and to provide an approximation of the motion noise. Expectation Maximization, a means of finding the parameters of a model by maximizing the probability of the data, can be used to find the parameters of the motion model [8].

Figure 2.5.2 shows samples from the motion model in Table 2.1 and was learned from data collected on our robot. Note that these are samples from the motion model and that the Extended Kalman Filter will approximate the uncertainty in these particles by a Gaussian distribution.

2.5.4 Measurement Update

The measurement update is the second step that happens in the Kalman filter. In it we reweight the belief based on measurements from the world. As we just performed the derivation for the motion update function g, we will do the same for the measurement update function h.

The measurement update function h maps the state onto the measurement given range and bearing. In this work, we will assume that measurements arrive as range and bearing. In the EKF we will assume that measurement updates are computed per landmark so as to minimize linearization errors. In reality, many landmarks may be seen at once, but we will iteratively update the mean and covariance. The measurement function is as follows:

$$h(X_t) = \begin{pmatrix} \sqrt{(r_x - l_x)^2 + (r_y - l_y)} \\ atan2(r_y - l_y, r_x - l_x) - r_\theta \\ l_s \end{pmatrix}$$

The measurement Jacobian H_t of $h(X_t)$, which is a function of the state and the measurement, will now be computed. Here (r_x, r_y, r_θ) is the robot pose and (l_x, l_y) is the position of the landmark, both of which are tracked in the state. The process is symmetric for all landmarks. H_t, is the partial derivative of h with respect to the state X_t. Defining $q_t = (r_x - l_x)^2 + (r_y - l_y)^2$, $\delta_x = l_x - r_x$ and $\delta_y = l_y - r_y$:

$$\begin{aligned} H_t &= \begin{pmatrix} \frac{\partial h_1}{\partial x} & \frac{\partial h_1}{\partial y} & \frac{\partial h_1}{\partial \theta} & \cdots & \frac{\partial h_1}{\partial l_k} & \cdots \\ \frac{\partial h_2}{\partial x} & \frac{\partial h_2}{\partial y} & \frac{\partial h_2}{\partial \theta} & \cdots & \frac{\partial h_2}{\partial l_k} & \cdots \\ \frac{\partial h_3}{\partial x} & \frac{\partial h_3}{\partial y} & \frac{\partial h_3}{\partial \theta} & \cdots & \frac{\partial h_3}{\partial l_k} & \cdots \end{pmatrix} \\ &= \begin{pmatrix} \frac{-l_x + r_x}{\sqrt{q_t}} & \frac{-l_y + r_y}{\sqrt{q_t}} & 0 & \cdots & 0 & \cdots & \frac{l_x - r_x}{\sqrt{q_t}} & \frac{l_y - r_y}{\sqrt{q_t}} & 0 & \cdots \\ \frac{l_y - r_y}{q_t} & \frac{-l_x + r_x}{q_t} & -1 & \cdots & 0 & \cdots & \frac{-l_y + r_y}{q_t} & \frac{l_x - r_x}{q_t} & 0 & \cdots \\ 0 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & 1 & \cdots \end{pmatrix} \\ &= \frac{1}{q_t} \begin{pmatrix} -\delta_x \sqrt{q_t} & -\delta_y \sqrt{q_t} & 0 & \cdots & 0 & \cdots & \delta_x \sqrt{q_t} & \delta_y \sqrt{q_t} & 0 & \cdots \\ \delta_y & -\delta_x & -q_t & \cdots & 0 & \cdots & -\delta_y & \delta_x & 0 & \cdots \\ 0 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & q_t & \cdots \end{pmatrix} \end{aligned}$$

Finally, the measurement noise is set to be:

$$Q_t = \begin{pmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_\theta^2 & 0 \\ 0 & 0 & \sigma_s^2 \end{pmatrix}$$

Note here that we do not have to map Q into the state space, as we did with the motion covariance R. Putting the pieces together, the overall algorithm can be seen in Algorithm 1, which ignores the problem of data association.

2.6 Adding Features to the Map

If the robot does not have prior knowledge of the total number of features in the map, it may be necessary to have a process for adding them. When adding features to an existing map, we must be particularly careful since both the uncertainty in the robot pose and the uncertainty of the measurement must be taken into account. While this may not be necessary in some implementations of SLAM, it is critical if our objective function is entropy. In particular, adding features with arbitrary initial uncertainty is clearly wrong since this introduction of uncertainty will hurt the value of taking various actions.

In this method, variances and covariances are taken into account. Let us assume that we want to add some number of random variables to the map and let us assume that there is some reasonable function $g(\hat{x}, z)$ that gives us the values of the new mean locations. Thus, we want to expand the previous covariance matrix by A and B.

$$C(x^{(+)}) = \left[\begin{array}{c|c} C(x^{(-)}) & B^T \\ \hline B & A \end{array}\right]$$

Assuming that $C(x)$ is defined to be the covariance of the existing state vector x and G_x is the Jacobian of g with respect to x, then the optimal way to add state features y, according to Smith, Self and Cheeseman [32], is to calculate A and B as follows:

$$\begin{aligned} A &= G_x C(x) G_x^T + G_y C(y) G_y^T \\ B &= G_x C(x) \end{aligned}$$

Algorithm 1 EKF_SLAM($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)

```
1.  $\mu_{t,1:3} = \mu_{t-1,1:3} + \begin{pmatrix} dr\,cos(\theta_s + d\theta) + ds\,cos(\theta_s + d\theta + \frac{\pi}{2}) \\ dr\,sin(\theta_s + d\theta) + ds\,sin(\theta_s + d\theta + \frac{\pi}{2}) \\ 2\,d\theta \end{pmatrix}$
2.  $G_t = \begin{pmatrix} 1 & 0 & -(v\triangle t)\,sin(\theta_s + \omega\,\frac{\Delta t}{2}) - \mu_{ds}\,sin(\theta_s + \omega\frac{\Delta t}{2} + \frac{\pi}{2}) & 0 & \dots & 0 \\ 0 & 1 & (v\triangle t)\,cos(\theta_s + \omega\,\frac{\Delta t}{2}) + \mu_{ds}\,sin(\theta_s + \omega\frac{\Delta t}{2} + \frac{\pi}{2}) & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \end{pmatrix}$
3.  $V_t = \begin{pmatrix} cos(\theta_s + d\theta) & -dr\,sin(\theta_s + d\theta) - ds\,sin(\theta_s + d\theta + \frac{\pi}{2}) & cos(\theta_s + d\theta + \frac{\pi}{2}) \\ sin(\theta_s + d\theta) & dr\,cos(\theta_s + d\theta) + ds\,cos(\theta_s + d\theta + \frac{\pi}{2}) & sin(\theta_s + d\theta + \frac{\pi}{2}) \\ 0 & 2 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{pmatrix}$
4.  $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + V_t R_t V_t^T$
5.  $H_t = \frac{1}{q_t} \begin{pmatrix} -\delta_x\sqrt{q_t} & -\delta_y\sqrt{q_t} & 0 & \dots & 0 & \dots & \delta_x\sqrt{q_t} & \delta_y\sqrt{q_t} & 0 & \dots \\ \delta_y & -\delta_x & -q_t & \dots & 0 & \dots & -\delta_y & \delta_x & 0 & \dots \\ 0 & 0 & 0 & \dots & 0 & \dots & 0 & 0 & q_t & \dots \end{pmatrix}$
6.  $Q_t = \begin{pmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_\theta^2 & 0 \\ 0 & 0 & \sigma_s^2 \end{pmatrix}$
7.  *For every observed feature* $z_l$*:*
8.      $\Psi_t = H_t \bar{\Sigma}_t H_t^T + Q_t$
9.      $K_t = \bar{\Sigma}_t H_t^T \Psi_t^{-1}$
10.     $\hat{z}_t = \begin{pmatrix} \sqrt{(x_r - x_l)^2 + (y_r - y_l)} \\ atan2(y_r - y_l, x_r - x_l) - \theta_r \\ s_l \end{pmatrix}$
11.     $\bar{\mu}_t = \mu_t + K_t(z_t - \hat{z}_t)$
12.     $\bar{\Sigma}_t = (I - K_t H_t)\bar{\Sigma}_t$
13. *endfor*
14. $\mu_t = \bar{\mu}_t$
15. $\Sigma_t = \bar{\Sigma}_t$
```

2.6.1 Adding 2D features

Presuming that we are working in 2 dimensions with the motion and observation models of point features described above, we obtain a signature s, a range r and a bearing ϕ to the landmark. Initialize each feature at a reasonable location with respect to the reference frame, this location is tracked in the global reference frame. Note that we could simply initialize each feature at the origin, but this would provide a poor linearization. More explicitly:

$$g(\hat{x}, z) = \begin{pmatrix} \hat{x_r} + r\,cos(\phi + \theta) \\ \hat{y_r} + r\,sin(\phi + \theta) \\ s \end{pmatrix}$$

Working from the definition above, we find that G_x is $3xN$ as follows:

$$G_x = \begin{pmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} & \frac{\partial f_1}{\partial \theta} & \frac{\partial f_1}{\partial l_1} & \cdots & \frac{\partial f_1}{\partial l_N} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} & \frac{\partial f_2}{\partial \theta} & \frac{\partial f_2}{\partial l_1} & \cdots & \frac{\partial f_2}{\partial l_N} \\ \frac{\partial f_3}{\partial x} & \frac{\partial f_3}{\partial y} & \frac{\partial f_3}{\partial \theta} & \frac{\partial f_3}{\partial l_1} & \cdots & \frac{\partial f_3}{\partial l_N} \end{pmatrix} = \begin{pmatrix} 1 & 0 & -r\,sin(\phi + \theta) & 0 & \dots & 0 \\ 0 & 1 & r\,cos(\phi + \theta) & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \end{pmatrix}$$

Checking our work, we can confirm that $G_x \Sigma^{(-)} G_x^T$ is $(3xN)(NxN)(Nx3) = 3x3$, as expected. Further, we still need to compute G_y as:

$$G_y = \begin{pmatrix} \frac{\partial f_1}{\partial r} & \frac{\partial f_1}{\partial \phi} & \frac{\partial f_1}{\partial s} \\ \frac{\partial f_2}{\partial r} & \frac{\partial f_2}{\partial \phi} & \frac{\partial f_2}{\partial s} \\ \frac{\partial f_3}{\partial r} & \frac{\partial f_3}{\partial \phi} & \frac{\partial f_3}{\partial s} \end{pmatrix} = \begin{pmatrix} cos(\phi + \theta_r) & -r\,sin(\phi + \theta_r) & 0 \\ sin(\phi + \theta_r) & r\,cos(\phi + \theta_r) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Since $C(y) = Q$ is the observation uncertainty it is a 3x3 diagonal matrix. Thus, $(3x3)(3x3)(3x3) = (3x3)$ matrix, as expected. Now we can update the map with these quantities using the update rules for A and B, above.

2.7 Conclusions

In this chapter, an extensive derivation of the Extended Kalman Filter for SLAM was performed. First, the Bayes filter was presented, from which the extended Kalman Filter was derived. The specific instantiation of the EKF was presented with a non-standard motion model. Finally, the optimal method of adding landmarks to the state based on the current covariance estimates was shown as well as how to learn motion model parameters. Overall, a complete method for mapping range and bearing measurements was shown.

Chapter 3

Exploration

The result of the mapping process is highly dependent on the sensor measurements and locations visited along the path of the robot. Arbitrary sensor measurements are expensive with respect to time, power and certainty. It is therefore critical to choose a sequence of sensor measurements that will maximize the accuracy of the map, otherwise additional time and power must be spent to achieve desired map accuracy. The problem of how to choose measurements to obtain the best map will be referred to as the *exploration* problem. This chapter will start with a review of the exploration literature, will progress to discuss information-theoretic techniques and will finally discuss the squared error objective function.

3.1 Background

There are a number of objective functions that may be considered when building a map: time, power, coverage, and quality are just a few. While distance and power consumed are certainly factors that need to be taken into account, the most critical factor for many applications is acquiring a high-quality map. Optimizations with respect to time or coverage are only useful if the user will not spend hours editing poor scan alignments, pose estimates, or feature extractions.

The first reward function is to minimize the unexplored area. In [40], the authors point the robot to the nearest unknown territory in hopes of exploring the entire world. The planning in this work is limited to choosing the nearest unexplored territory and does not factor in global distance or time.

Another reward function is to take observations that make sure the robot sees all of the environment. This is closely related to the variants of the *art-gallery* problem, which will find the minimum number of locations that cover all surfaces [12]. This, however, requires a rough map beforehand. Also, both of these reward functions may produce poor quality maps in many situations, since map quality is not taken into account.

A reward function that reflects the certainty of the map is reflected in the amount of information gained during the mapping process. This reward function will be one of the primary focuses of this thesis, along with the squared error. There is an abundance of related work using information gain for choosing actions, among which the most closely related comes from [21, 33, 10, 30]. In [21, 33], a weighted objective function is used that balances the amount of distance the robot will travel with the amount of information gain it will receive. The robot will then take the action which maximizes the myopic reward.

Information-theoretic reward functions have also been used in a non-myopic manner to choose the next best measurement in sensor networks, with a number of interesting results [18]. This problem differs substantially from mobile-robot exploration in that and measurements cannot be taken at any time, but occur as the result of the robot pose.

3.2 Trajectory Exploration

Exploration in the above works tend to have a number of limitations. Destinations are chosen without considering the controller, real-time operation may be impossible and destination selection is myopic. We consider a slightly different problem, but intend to overcome all of these limitations. The specific differences can be seen in Table 3.2.

Trajectory Exploration	**Location-based Exploration**
A method for visiting keypoints	A method for choosing keypoints
Slow learning phase, very fast execution phase	Slow execution phase/no learning
Multi-step	Myopic

Definition 3.2.1. (The trajectory exploration problem) Given an action space of trajectories A, a controller C which can execute $a \in A$, a robot simulator S to simulate robot motion and

measurement errors, a state-estimator E that tracks the robot position (x_r, y_r, θ_r) and a map M, a sequence of destinations $X_g = (x_g^0, y_g^0, x_g^1 y_g^1 \ldots, x_g^n, y_g^n)$, and a reward function R, the *trajectory exploration problem* is to find a sequential choice of actions $a \in A$ such that the overall trajectory of the robot through the destinations will maximize the expected reward corresponding to the accuracy of the map.

The action space A will be a set of parametrized curves and the controller C will be a carrot-following technique with a fixed translational velocity. The simulator S uses learned motion and sensor models and will simulate robot motion and laser measurements. The state estimator E will be an Extended Kalman Filter with the form as discussed in Chapter 2. The destinations X_g that the robot is to visit have been given from an oracle. Our precise solution will be discussed in depth in Chapter 4.3.

These particular design decisions are not completely critical for the use of this algorithm. However, there are two things that will become critical: a robust and accurate simulator and an easily computable scalar reward function. Other state estimators may be used as long as the entropy, information or squared error are available. For much of this chapter, we will discuss the particular reward functions R that we will use in this work. The next two sections will discuss the information-theoretic and squared error objective functions, respectively.

3.2.1 Why trajectory exploration?

A good trajectory planner will balance exploration, re-localization, and travel-time. Observing very certain features will allow the robot's own uncertainty to drop and will allow it to transfer this certainty into newly observed landmarks.

Most waypoint-based selection algorithms assume a fixed trajectory between waypoints, which limits the quality of the map by not taking into account a number of factors. These factors include the number of observations, the limited visibility of the robot, the amount of error introduced by particular motions, the particular motion controller, and directional visibility of landmarks.

Since the mapping process is inherently stochastic, the observations of landmarks will be correspondingly stochastic. However, this randomness is controllable. When close to a landmark, feature extraction will often occur more frequently. It is also possible that at certain angles it is more likely that a landmark will be extracted; two posts may not be detected when the second post

Factors influencing trajectory choice
Number/quality of observations
Orientation/Visibility of landmarks
Motion errors

is occluded.

Many robots also have limited visibility and can see only in 180 degrees. In this case, the robot orientation at each point along the path is important for increasing the number of measurements. Occlusions may also enable the robot to see a landmark at one time, but not at another. In this case, the robot may want to choose trajectories that maximize the amount of time it sees this landmark, even traveling between waypoints.

However, any particular landmark is not enough, as each visible landmark will need to be balanced so that the robot may want to maximize the number of landmarks it sees. Further, it may want to have a dearth of visible landmarks early to see a wealth of them later or conversely. In such a case, the multistep nature of our solution will come into play.

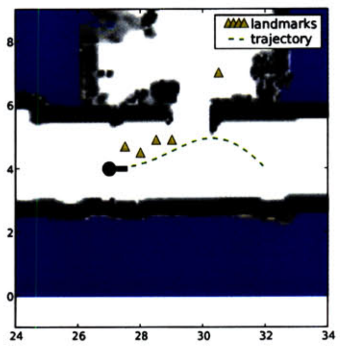
The amount of error introduced by a particular robot motion is critical to the mapping process. When a significant amount of error is introduced into the robot pose and yet few or no landmarks are visible, the SLAM filter may have increasing difficulties maintaining an accurate posterior. By choosing good trajectories, the robot can minimize the error introduced by its own motion and overcome a dearth of landmarks over small timeperiods.

The basic lesson is that, though the measurements are inherently random, the randomness of the measurements can be controlled through careful choice of actions.

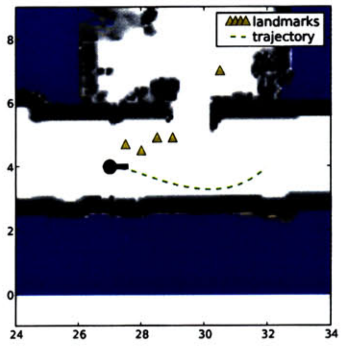
3.2.2 Examples

To give a flavor of the type of problems we hope to solve, we will now give a few examples. In Figure 3.2.2, four trajectories are shown. The one in the upper left corner is the best; it goes near all of the landmarks to maximize the number of observations, it does not have any places where rotational velocity is high, and it takes as many observations of the occluded landmark as is possible.

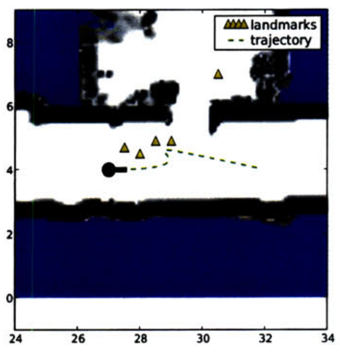
However, each of the others has at least one part that could be improved. The upper right trajectory could go nearer to the landmarks and could observe the occluded landmark in the room



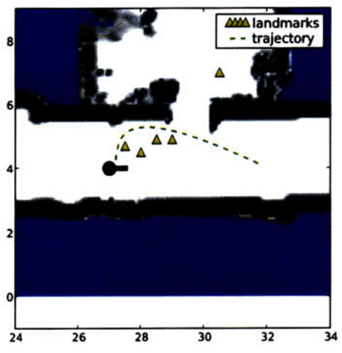
(a) Good Trajectory



(b) Bad Trajectory (visibility/distance from landmarks)



(c) Bad Trajectory (high rotational velocity)



(d) Bad Trajectory (occluded landmarks)

in the upper right more often. In the lower left figure, the rotational velocity of the robot will be very high in the center part of this trajectory, thus increasing the likelihood of a mapping error. In the bottom right, the trajectory is fairly good, however the occluded landmark could be visible. Given these constraints, clearly the first trajectory is desirable over the other three.

Though not explicitly optimizing with respect to any of these things, we will see that the learning and objective function will take all of these qualities into account. In the next sections we will be discussing the particular reward function and how it will allow all of these to be realized.

3.3 Squared Error Reward

While information-theoretic techniques allow for a certain sort of generality, there are a number of drawbacks to its use in practice that make sum-squared error a more desirable metric. While the expected error of each landmark location will decrease with each observation, this does not mean that the sum-squared error in the map will decrease.

Overconfidence, bad data-associations, and filter divergence account for a number of problems that may be encountered with a Kalman filter. While none of these can be completely avoided, a well-chosen objective function will discourage trajectories that introduce such problems. Sum-squared error represents such an objective function.

On the other side, squared error cannot always be computed. In our situation squared error is only reasonable because the robot will have access to a simulator. Were this not the case, then the only recourse is to use information gain.

Definition 3.3.1. Given a state estimator E tracking features $l_1 \dots l_k$ and a simulator S with the true locations $s_1 \dots s_k$ of those features the *squared error* is defined as:

$$SE = \sum_{i=0}^{k} (l_i - s_i)^2$$

Note of course that the landmarks l_i will not in general be in the same reference frame as the simulated positions. Either the associations may be stored or iterative closest point may be run to return the optimal rotation and translation between the datasets [2, 37].

3.4 Information-Theoretic Reward

In this section we perform a review of the information-theoretic reward function and specialize it to the case of the Kalman filter. We will begin by defining and deriving differential entropy for a Gaussian random vector. We will also show the trace matrix norm and why it is a good metric for uncertainty. With these tools we will then define the notion of *information gain*.

3.4.1 Differential Entropy

Roughly speaking, entropy measures the amount of information in a probability distribution. When a distribution is peaked, there is low entropy and when the distribution is much like a uniform distribution there is high entropy. With discrete random variables, Shannon entropy corresponds to the expected number of bits that it will take to encode a symbol [22].

Differential entropy is the generalization of entropy to continuous probability distributions and is defined as follows:

Definition 3.4.1. (Differential Entropy) Given a probability distribution $P(x)$ and observations A, then *differential entropy* and *conditional differential entropy* are defined as:

$$\begin{aligned} H(V) &= \int_x P(x)\ \log P(x) dx \\ H(V|A) &= \int_x \int_a P(x,a)\ \log P(x|a) da\, dx = H(V,A) - H(A) \end{aligned}$$

There are only a few distributions where this integral can be solved exactly, or where this integral is even defined. Among these is the multivariate Gaussian distribution. The expression for the differential entropy of a multivariate Gaussian is as follows [4].

Theorem 3.4.2. *(Gaussian Differential Entropy) For a Gaussian random vector x of length n with covariance Σ and mean μ, the Gaussian Differential Entropy may be derived as:*

$$H_d(x) = \frac{1}{2} \ln[(2\pi e)^n \det \Sigma\,]$$

Further, we note a very useful property of a square, symmetric matrix; the determinant is the product of the eigenvalues.

3.4.2 The trace matrix norm

The differential entropy is not the only measure that we might use for our reward function. Though it roughly corresponds to information, we do not claim that the trace can be derived from a more general theory. However, we will show the intuition and reasons for its use. We will misuse notation a bit by calling the trace H, though it is hoped that the reader will bear with us.

Definition 3.4.3. (Trace matrix norm) For a Gaussian random vector x of length n with covariance Σ and mean μ, the *trace matrix norm* may be defined as:

$$H_t(x) \quad = \quad \sum_{i=0}^{n} \sigma_{i,i}^2$$

Similarly to the determinant, the trace has an interpretation in terms of the eigenvalues of the covariance matrix; it is the sum of the eigenvalues.

3.4.3 Trace and Determinant Intuition

While knowing that two forms of uncertainty roughly correspond to the trace and determinant of the covariance matrix is useful, there is quite a bit of intuition that underlies these metrics. Reminding ourselves that the eigenvectors define the principal directions of the covariance and the eigenvalues roughly correspond to the length of these vectors, we can understand the information metrics.

The determinant roughly corresponds to the volume of the ellipse defining the covariance. Reducing one of the dimensions of uncertainty to zero corresponds to driving the determinant of the covariance matrix to zero. We can see this through the eigenvalue argument since uncertainty about one variable being zero corresponds to an eigenvalue being zero.

The trace roughly corresponds to the circumference of the covariance ellipse. We can see this from the eigenvalues, since we are taking the sum of the size of each of the principal directions. On the surface, there are fewer problems for the trace, since when one random variable is zero, the uncertainty will not be zero. However, we will in fact show some problems with the trace in later sections of this chapter.

3.5 Information Gain (IG)

Information gain corresponds to how much more certain and peaked the probability distribution has become after making a number of observations. Intuitively, when the information gain is positive, then the probability distribution is more peaked; when it is negative it is less peaked. More formally information gain corresponds to the decrease in the entropy $H(V)$ of the distribution after observing a subset of the variables A.

Definition 3.5.1. (Information Gain) Given $H(V)$ and $H(V|A)$ are the entropy and conditional entropy, the *information gain* I is defined as:

$$I(V;A) = H(V) - H(V|A)$$

How the variables are observed is very important to how the probability distribution changes. Information gain can be negative, so that we may lose information about a number of variables. Regardless, in robot exploration we can attempt to influence the type and number of variables that will be observed.

Definition 3.5.2. Given $H(x)$ is a form of entropy, the *exploration* problem is to find a subset of variables A^* such that:

$$A^* \quad = \quad argmax_A H(x) - H(x|A)$$

For our purposes here, we will use the information from Gaussian differential entropy and the trace as two forms of information gain.

The literature on optimal experimentation defines two objective functions on a Gaussian distribution in terms of information: D-optimality and A-optimality. D-optimality is simply the determinant as a reward function, while A-optimality is the trace as a reward function. Formally we can define them as follows:

Definition 3.5.3. Given a Gaussian distribution with covariance Σ and mean μ, the *D-optimal* objective function is defined as:

$$I(x) \quad = \quad \det(\Sigma_V) - \det(\Sigma_{V|A})$$

Definition 3.5.4. Given a Gaussian distribution with covariance Σ and mean μ, the *A-optimal* objective function is defined as:

$$I(x) \quad = \quad tr(\Sigma_V) - tr(\Sigma_{V|A})$$

3.6 Information Gain Examples

The properties of these objective functions are already of extreme importance to us, so we examine some of their properties in general and with respect to robot motion. Knowing what the posterior trace and determinant look like will enable us to develop our intuition of what sorts of trajectories will be generated.

It is desirable during the motion update step of the Kalman Filter the robot can only become more uncertain about its position. In the following sections, we will look at the motion update step and show a few striking properties of the trace and covariance matrix. Specifically, we will show that when using the determinant one of the principal directions can be reduced to zero, which in turn reduces the determinant to zero. Conversely, even when the determinant does not change, the trace can be reduced a significant amount. See Appendix for matrix properties when relevant.

3.6.1 Properties of the determinant

The time projection step of the EKF is: $\bar{\Sigma}_t = G\Sigma_{t-1}G^T + VWV^T$. For the moment let us assume there is no noise added to the covariance matrix so that VWV^T is 0. We want to know what we can say about $\bar{\Sigma}_t$.

The following theorem says that whenever a matrix A has a determinant smaller than 1, then $A\Sigma A^T$ will have smaller determinant than Σ.

Theorem 3.6.1. $det(A\Sigma A^T) \leq det(\Sigma) \iff det(A) \leq 1.$

Proof. The proof is simple and uses the fact that $det(AB) = det(A)det(B)$ and $det(A) = det(A^T)$.

Assuming that $det(A) \leq 1$, then

$$det(A\Sigma A^T) = det(A)det(\Sigma)det(A^T) = det^2(A)det(\Sigma) \leq det(\Sigma)$$

Assuming $det(A\Sigma A^T) \leq det(\Sigma)$, then

$$det(A\Sigma A^T) = det^2(A)det(\Sigma) \leq det(\Sigma) \quad \iff det^2(A) \leq 1$$

$\square$

Example 3.6.2. A has a determinant smaller than 1 when it is a diagonal matrix with entries less than 1. Thus $A\Sigma A^T \leq \Sigma$, for any covariance matrix Σ.

3.6.2 Properties of the trace

The general case of the trace is much less satisfying than the determinant. However, we espouse the general form here and then make it specific to the problem of robot motion.

Theorem 3.6.3. *For Σ a covariance matrix and*

$$A = \begin{pmatrix} | & | & & | \\ a_1 & a_2 & \dots & a_n \\ | & | & & | \end{pmatrix} \textit{ then,} \quad \begin{aligned} tr(A\Sigma A^T) = \;& (1 - a_1 \cdot a_1)\sigma_{11} + (1 - a_2 \cdot a_2)\sigma_{22} \cdots + (1 - a_n \cdot a_n)\sigma_{nn} \\ & +2a_1 \cdot [a_2\sigma_{12} + a_3\sigma_{13} + \dots a_n\sigma_{1n}] \\ & +2a_2 \cdot [a_3\sigma_{23} + a_4\sigma_{24} + \dots a_n\sigma_{2n}] \\ & \cdots + 2a_{n-1} \cdot [a_n\sigma_{n-1,n}] \end{aligned}$$

Proof. The proof simply uses the property of the trace that $tr(AB) = tr(BA)$. Using this property, we can see that $tr(A\Sigma A^T) = tr(AA^T\Sigma)$. Expanding this equation, we can easily find the condition where $tr(A\Sigma A^T) \leq tr(\Sigma)$. □

3.6.3 General Metric Properties

So far we have talked in general about the properties of the trace and determinant. In this section we will see specific instances of how they can be directly manipulated by particular actions.

Example 3.6.4. Reducing the determinant without reducing the trace.

It turns out that there are certain circumstances when one can reduce the determinant without reducing the trace. In particular, if we assume that A is a $2x2$ matrix and that it is diagonal $A = diag\ (a, b)$, then the cross terms in Theorem 3.6.3 disappear and we have only the first set of terms so that $(-1 + a^2)\sigma_{11} + (-1 + b^2)\sigma_{22} < 0$. We will assume that $0 < b < 0.5$. We want to set

a so that the trace is the same, but the determinant is different, thus, we do the following:

$$\begin{aligned}(-1+a^2)\sigma_{11}+(-1+b^2)\sigma_{22} &= 0\\ (-1+a^2) &= \frac{(1-b^2)\sigma_{22}}{\sigma_{11}}\\ a &= \sqrt{\frac{\sigma_{22}}{\sigma_{11}}(1-b^2)+1}\end{aligned}$$

By construction the trace will be the same. However, $det(A\Sigma A^T) = det^2(A)det(\Sigma)$. Without loss of generality we assume that $\frac{\sigma_{22}}{\sigma_{11}} < 1$, since we can always choose a and b such that this is the case.

$$\begin{aligned}det^2(A) &= b^2\left[\frac{\sigma_{22}}{\sigma_{11}}(1-b^2)+1\right]\\ &\leq \frac{1}{4}\left[\frac{\sigma_{22}}{\sigma_{11}}+1\right]\\ &\leq \frac{1}{2}\\ &< 1\end{aligned}$$

Thus, we can conclude that the determinant of the posterior will always be reduced for a certain selection of A, even though the trace will remain the same. Note that repeated applications of A will reduce the determinant to 0.

Example 3.6.5. Reducing trace without reducing the determinant.

Taking $A = \begin{pmatrix} 1 & a \\ 0 & 1 \end{pmatrix}$, the $tr(\bar{\Sigma_t}) = \sigma_{11} + 2a\sigma_{12} + a^2\sigma_{22} + \sigma_{22}$ and $tr(\Sigma_{t-1}) = \sigma_{11} + \sigma_{22}$. We want $tr(\bar{\Sigma_t}) < tr(\Sigma_{t-1})$. Rewriting, we get:

$$\begin{aligned}tr(\bar{\Sigma_t}) &< tr(\Sigma_{t-1})\\ \sigma_{11} + 2a\sigma_{12} + a^2\sigma_{22} + \sigma_{22} &< \sigma_{11} + \sigma_{22}\\ 2a\sigma_{12} + a^2\sigma_{22} &< 0\end{aligned}$$

Thus, there are two cases, either when $\sigma_{12} > 0$ or when $\sigma_{12} < 0$.

1. If $a > 0$ and $\sigma_{12} < 0$, then we obtain the condition: $a < -2\frac{\sigma_{12}}{\sigma_{22}}$ or $a \in (0, 2\frac{\sigma_{12}}{\sigma_{22}})$

2. If $a < 0$ and $\sigma_{12} > 0$, then we obtain the condition: $a > -2\frac{\sigma_{12}}{\sigma_{22}}$ or $a \in (2\frac{\sigma_{12}}{\sigma_{22}}, 0)$

Clearly the $det(A) = 1$ so $det(A\Sigma A^T) = det^2(A)det(\Sigma) = det(\Sigma)$.

3.6.4 Metric properties applied to the EKF

So far we have given examples that do not take into account the specific form of the Jacobians for robot motion. If we now specialize to this case, we can find even more structure in the posterior distribution after robot motion. Recall that for robot motion we derived A to be:

$$A = \begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix}$$

Theorem 3.6.6. *(The determinant and robot motion) For A as defined above for robot motion and Σ a covariance matrix, $det(A\Sigma A^T) = det(\Sigma)$.*

Proof. Again, it should be clear that $det(A) = 1$. Thus, for the same reasons as before, it should be clear that $det(A\Sigma A^T) = det(\Sigma)$. This is a good thing, since we would like the uncertainty in robot pose not to decrease with robot motion. □

While one might expect the trace of the pose uncertainty to always grow if no measurements are observed, there are times it may shrink. Let us examine why this might be the case. Note that $\Sigma_t = G\Sigma_{t-1}G^T + VWV^T$. Here, we will examine the 3x3 pose submatrix after projection using the Jacobian G: $G\Sigma_{t-1}G^T$. Without loss of generality, we assume that G, Σ_{t-1} and G^T are all $3x3$ matrices. Thus, the following computation is performed:

Theorem 3.6.7. *The trace and robot motion For A as defined above for robot motion and Σ a covariance matrix, in certain situations $tr(A\Sigma A^T) \leq tr(\Sigma)$.*

Proof. First we expand the Jacobian to find the new values for the diagonal elements of the covariance matrix.

$$
\begin{aligned}
G\Sigma_{t-1}G^T &= \begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} | & | & | \\ \sigma_1 & \sigma_2 & \sigma_3 \\ | & | & | \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & 1 \end{pmatrix} \\
&= \begin{pmatrix} \sigma_{11}+\sigma_{13}a & \sigma_{21}+\sigma_{23}a & \sigma_{31}+\sigma_{33}a \\ \sigma_{12}+b\sigma_{13} & \sigma_{22}+b\sigma_{23} & \sigma_{32}+b\sigma_{33} \\ \sigma_{13} & \sigma_{23} & \sigma_{33} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ a & b & 1 \end{pmatrix} \\
&= \begin{pmatrix} \sigma_{11}+\sigma_{13}a+(\sigma_{31}+\sigma_{33}a)a & \sigma_{21}+\sigma_{23}a+(\sigma_{31}+\sigma_{33}a)b & \sigma_{31}+\sigma_{33}a \\ \sigma_{12}+b\sigma_{13}+(\sigma_{32}+b\sigma_{33})a & \sigma_{22}+b\sigma_{23}+b(\sigma_{32}+b\sigma_{33}) & \sigma_{32}+b\sigma_{33} \\ \sigma_{13}+\sigma_{33}a & \sigma_{23}+\sigma_{33}b & \sigma_{33} \end{pmatrix} \\
&= \begin{pmatrix} | & | & | \\ \sigma_1^{'} & \sigma_2^{'} & \sigma_3^{'} \\ | & | & | \end{pmatrix}
\end{aligned}
$$

Using the trace of the covariance matrix as a measure of uncertainty, we find the situations in which the pose covariance goes down between two timesteps. This happens when:

$$
\begin{aligned}
\sigma_{11}^{'}+\sigma_{22}^{'}+\sigma_{33}^{'} &< \sigma_{11}+\sigma_{22}+\sigma_{33} \\
\sigma_{11}+\sigma_{13}a+(\sigma_{31}+\sigma_{33}a)a+\sigma_{22}+b\sigma_{23}+b(\sigma_{32}+b\sigma_{33})+\sigma_{33} &< \sigma_{11}+\sigma_{22}+\sigma_{33} \\
\sigma_{13}a+(\sigma_{31}+\sigma_{33}a)a+b\sigma_{23}+b(\sigma_{32}+b\sigma_{33}) &< 0 \\
2a\sigma_{13}+2b\sigma_{23}+\sigma_{33}(a^2+b^2) &< 0 \\
\sigma_{33}(a^2+b^2) &< -2a\sigma_{13}-2b\sigma_{23}
\end{aligned}
$$

Note that a and b can larger or smaller than 0, as can the covariance of x or y with θ. These particular values are defined in Chapter 2. Thus, the pose uncertainty will at times become smaller overall, when the noise is smaller than the reduction after the projection of the Jacobian. This is an important and unexpected property to note and identifies one dissatisfying aspect of the trace as a measure of uncertainty. $\square$

3.7 Conclusions

In this section we have defined the trajectory exploration problem, presented a number of ways that it could be useful for generating high-quality maps, and elaborated the information-theoretic and squared-error objective functions. It remains for us to show how we might solve the trajectory exploration problem and to give results for real problems. These will be discussed in the following chapters.

Chapter 4

Markov Decision Processes

As discussed in the last chapter, most other approaches to exploration are greedy with respect to the next action. In this section, we will present an alternative approach that is not greedy and will select the sequence of actions to maximize the accuracy of the resulting map. Our approach will require a large amount of offline computation but can be quickly performed online. In this chapter, we will first present the Markov Decision Process formalism [29], how to obtain a policy from a MDP and finally, how to formulate robot exploration as a belief-space MDP.

4.1 Markov Decision Processes

There are a number of elements to a MDP; these are the notions of *state*, *action*, *reward* and *uncertainty*. Decisions in a MDP are made at specified intervals called *decision epochs* based on the current state. The *state* in a MDP describes everything that an agent needs to know to make a decision. Some examples of state variables are angles, velocities, accelerations, locations or inventory.

The notion of *reward* provides an objective function for the agent. A *policy* is a mapping from states to actions and should maximize the expected future reward over N decision steps, or potentially over an infinite discounted horizon. While the agent could take the action that maximizes its current reward, this is may often be a poor choice, since the agent may be forced into a situation where only increasingly worse choices exist, even when the robot could be greedy with respect to the next action.

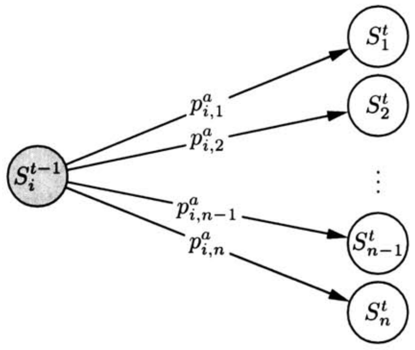
The idea of *uncertainty* is incorporated via the transition from state to state under an action. The result of an action may not be certain, so that translating a robot 3 meters may actually result in a translation of 2.9 meters or it may result in a translation of 3.1 meters. After each action, our agent will fall into some new state $s^{'}$, where it must make a new decision.

Formally, a MDP is a tuple $\langle T, A, S, R, s_0 \rangle$ where S is the state space, s_0 is the start state, A is the set of actions available to the agent, and $R(s, a, s^{'})$ is the reward for taking action a in state s and ending in state $s^{'}$. $p(s^{'}|s, a) = T(s, a, s^{'})$ is the probability of transitioning to state s' given a prior state s and action a.

Figure 4.1.1 illustrates a Markov Decision Process at a particular decision epoch $t - 1$. At time $t - 1$, the agent is in state s_i and decides to take action a in this state. With some probability the agent will transition to any of the other states s_j after taking this action. For any state and every action, there is a different graph. Note that the probability of any of these transitions may be zero, thus removing some of the links. For each of these edges, there is also a reward r.

When a problem can be formulated as a MDP, the goal is then to find a policy that will prescribe the best action for the agent to take in any state. Algorithms to find policies will be discussed in Sections 4.2 and 4.2.1.

Figure 4.1.1: Abstraction of a Markov Decision Process



4.1.1 State and Action Spaces

State and action spaces may be continuous or discrete. Much work has focused on the case where both the state and action spaces are discrete. In a MDP, the current state tells the agent everything necessary to decide the best action. This contrasts to the case where some notion of history is required to uniquely determine the state. This also contrasts to partially observable Markov decision processes (POMDPs), where only an observation is seen, but from which a unique state may be unrecoverable.

For example, imagine that the agent sees a particular hallway that is indistinguishable from other identical hallways on other floors of the same building. In this situation, the agent cannot be sure which floor he is on until he sees the floor numbers, or some other observation which gives a unique identifier of state. In this situation, the state is unknown: the agent has a belief or pdf about its state.

Though this problem could be formulated as a POMDP, the approximations that would need to be made and the remaining difficulty of solving it would outweigh the benefits. Thus, we use a belief-state MDP in this work.

4.1.2 Transition probabilities and Rewards

The transitions indicate the probability of transitioning to each of the other states given knowledge of the current state and action. The transition probabilities must be learned from historical data or otherwise known beforehand. When the state and actions are discrete, the transition probabilities take the form of a discrete distribution over the posterior state:

$$p(s^{'}|s, a)$$

The rewards often must be known a priori and must be bounded and stationary. However, there are few other requirements on the reward, as it can be a continuous function over the real numbers. When the rewards depend on the next state an expectation is taken so that the reward only depends on the current state and action, but not on the posterior state:

$$r(s,a) = \sum_{s_j \in S} p_t(s_j|s_i,a) r_t(s_i,a,s_j)$$

While for the most part $p(s_j|s_i,a)$ tends to be a probability distribution, a wider range of systems can be modeled when not requiring $\sum_j p(s_j|s_i,a) = 1$ [29].

4.1.3 Optimization Criteria

A number of optimization criteria are available for Markov decision processes and are divided into finite and infinite horizon Markov decision processes. The finite-horizon criteria will be discussed here, and are similar to the results for the infinite horizon MDP. Our objective function will be the expected reward over a particular time horizon:

$$v_N^{\pi}(s) = E\left[\sum_{t=1}^{N-1} r_t(X_t,Y_t) + r_N(X_N)\right]$$

4.2 Dynamic Programming (Backwards Induction)

The standard technique to solve a finite-horizon MDP is by using dynamic programming, which will calculate the optimal value of each state at each time horizon and then use this optimal value to create a policy. Time is iteratively decreased until the wanted horizon is achieved. This algorithm can be seen in Figure 4.2.1.

The expected future reward is often called the *value function.* This function v is computed in the first part of step 2. From the value function, a policy π, mapping states to actions, is directly derived at the current time in the second part of step 2.

Dynamic programming starts at the end and works backwards. Since we have full knowledge of the reward, transitions, states and actions, we can store a large table with time as the x-axis and the states as the y-axis. We compute a column of this table with the first part of step 2, only using the previous column in the table and then we use the action from which the optimal value was computed to create the optimal policy.

For real-world applications of dynamic programming, a transition model must be available to the algorithm as well as a fully specified reward function. Computation of the transition function can be computationally and memory intensive and the specification of the reward function for all states can be difficult. Policy search by dynamic programming (PSDP), a modification to dynamic programming to deal with continuous state, neither requires the full transition model nor does it require full specification of the reward function. It avoids these problems by sampling traces through the state space at the cost of full optimality.

Figure 4.2.1: Policy Evaluation and Improvement

1. $t = N$ and $v_N^*(s_N) = r_N(s_N)$
2. $t = t - 1$

$$\begin{aligned} v_t^*(s_t) &= \max_{a \in A} \left[r_t(s_t, a) + \sum_{s_j \in S} p_t(s_j^{t+1} | s_i^t, a) v_{t+1}^*(s_j) \right] \\ \pi^*(s_t) &= argmax_{a \in A} \left[r_t(s_t, a) + \sum_{s_j \in S} p_t(s_j^{t+1} | s_i^t, a) v_{t+1}^*(s_j) \right] \end{aligned}$$

3. If $t = 1$ stop. Otherwise, go to 3.

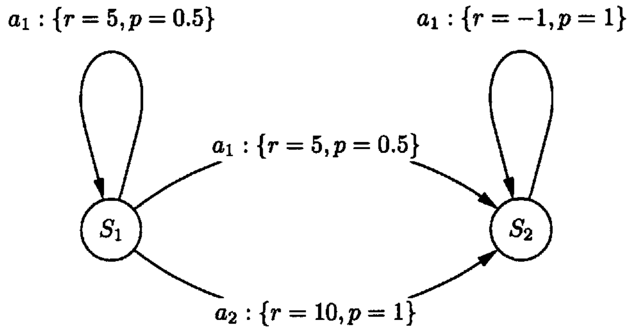
To make things clear, we will now indulge ourselves in a very simple Markov decision process that includes two states. This example is taken from [29], and is reproduced here for clarity.

Example 4.2.1. A simple MDP is shown in Figure 4.2.2. It has two states, s_1 and s_2 and two actions, a_1 and a_2. The arcs correspond to outcomes of a particular action. If action a_1 is taken in state s_1, then we may either end up in state s_1 or s_2 with 0.5 probability and reward of 5. Taking action a_2 in s_1 or action a_1 in s_2 are deterministic and will both transition to s_2 with probability 1.

We will be looking for an N-step policy for this MDP. The actual policy should be obvious: take action a_1 unless in the next to last timestep, in which case if we are in s_1, then take a_2. Dynamic programming will give us this solution; let us see how.

On the first step, we want to maximize the reward from any given state. The agent's life will be over in one timestep, so he should want to live life to the fullest. Clearly in this case, from s_1, he will take action a_2 and only has one option from s_2, which is to take action a_1. Thus, $v_1(s_1) = 10$ under a_2, $v_1(s_1) = 5$ under a_1 and $v_1(s_2) = -1$ under a_1. Clearly the action which maximizes the

Figure 4.2.2: An abstraction of a simple MDP


$a_1 : \{r = 5, p = 0.5\}$
$a_1 : \{r = -1, p = 1\}$
$a_1 : \{r = 5, p = 0.5\}$
S_1
S_2
$a_2 : \{r = 10, p = 1\}$

instantaneous reward is $\pi_1(s_1) = a_2$ and $\pi_1(s_2) = a_1$.

For, $t = 2$, we have the following for s_1:

$$\begin{aligned}
v_2^{a_1}(s_1) &= 5 + [0.5(-1) + 0.5(10)] &&= 9.5\\
v_2^{a_2}(s_1) &= 10 + [1(-1)] &&= 9.0\\
v_2^{a_1}(s_2) &= -1 + [1(-1)] &&= -2.0
\end{aligned}$$

Now, clearly the correct action is a_1 in state s_1. This action will continue to be optimal indefinitely after the first timestep for state s_1. State s_2 only has one action available, so a_1 will always be taken. Note that if the agent is greedy in s_1, then over a long horizon, this corresponds to a very low value. The value for a number of horizons is shown in the following table:

	$t=1$	$t=2$	$t=3$	$t=4$	...	$t=N-1$	$t=N$
s_1	10	9.5	8.75	7.875	...	$\approx 12-(N-1)$	$\approx 12-N$
s_2	-1	-2	-3	-4	...	$-(N-1)$	$-N$

4.2.1 Policy Search by Dynamic Programming (PSDP)

Policy Search Dynamic Programming is a form of dynamic programming that uses samples to explore a continuous state space. A value is computed for each state using the backed-up policies to evaluate each sample. After enough samples are evaluated, the action which maximizes the value

Figure 4.2.3: Policy Search by Dynamic Programming

1. $S = \{s_1, \ldots, s_{m_1} | s_i \sim P_i\}$
2. Where $v_t^{a_j, \pi_{t+1}, \ldots, \pi_N}(s_i)$ is a trace through the MDP sampled m_2 times using the previously computed policies:

$$v_t^{A; \pi_{t+1}, \ldots, \pi_N}(S) = \begin{pmatrix} v_t^{a_1, \pi_t+1, \ldots \pi_N}(s_1) & \ldots & v_t^{a_{|A|}, \pi_t+1, \ldots \pi_N}(s_1) \\ \vdots & \vdots & \vdots \\ v_t^{a_1; \pi_t+1, \ldots \pi_N}(s_{m_1}) & \ldots & v_t^{a_{|A|}, \pi_t+1, \ldots \pi_N}(s_{m_1}) \end{pmatrix}$$

3. $\pi_t'(S) = \arg\max_{a \in A} \left[v_t^{A, \pi_{t+1}, \ldots, \pi_N}(S) \right]$, where $\arg\max$ is taken with respect to the rows of the matrix.
4. Train a classifier with input S and output $\pi_t'(S)$:

$$\pi_t(S) : S \to \pi_t'(S)$$

5. $\Pi = \{\pi_t \ldots \pi_N\}$; $t = t - 1$; goto step 1 unless t is the horizon that we care about.

of a state is taken as a training example: $S_t \to A$, and a classifier is trained that maps states to actions at this horizon.

The advantage here will be that we do not need to know the transition model or the reward function, we simply need to be able to sample from the transition model and evaluate the reward function. Further, when the classifier can use continuous values, then PSDP can deal with continuous state.

The PSDP algorithm can be seen in Figure 4.2.3. PSDP learns policies by decomposing the problem into a series of one-step policies, as in dynamic programming. The algorithm operates by first learning a one-step policy at time T (e.g., the end time) by sampling states and actions, and learning to predict an action that maximizes the immediate one-step reward. A new one-step policy for the previous time step $T - 1$ is then learned by sampling states and actions, obtaining an immediate reward as before. Using the policy previously obtained at time T, a two-step value is obtained for each state and action at $T - 1$. Thus, the value associated with each state and action is accumulated from the immediate reward at time $T - 1$ *and* the reward received by running the learned policy for time T.

The learner iterates at each time t generating a sample state $s^{(i)}$ and action a, propagating each sample $s^{(i)}$ forward through to time T using the policies π_i for $i \in \{t \dots T\}$. After sampling all of the actions in state s at time t, the best action a' is selected at state s. For each state s, there is thus a best action a'. These are used as training pairs in a supervised learning problem at time t and the result is a classifier that, for any state s, provides an optimal action a.

Guarantees of PSDP

The guarantees of PSDP rely on the quality of the policy generated and the variational distance between the initial probability distribution from which the initial samples are generated and the true distribution seen in practice. While these guarantees are not extremely strong theoretical guarantees, the algorithm has been demonstrated to work well in practice on a number of problems [1]. The particular result cited is stated as follows:

Theorem 4.2.2 (Value Function Approximability). *For $\Pi = (\pi_0, \dots, \pi_{N-1})$ be a policy that is ϵ approximate, such that:*

$$E_{s \sim \mu_t}\left[V_{\pi_t, \dots, \pi_N}(s)\right] \geq \max_{\pi'} E_{s \sim \mu_t}\left[V_{\pi', \dots, \pi_N}(s)\right] - \epsilon$$

Then for all $\pi_{ref} \in \Pi^N$:

$$V_\pi(s_0) \geq V_{\pi_{ref}}(s_0) - N\epsilon - N d_{var}(\mu, \mu_{\pi_{ref}})$$

where, d_{var} is the variational distance between two probability distributions:

$$d_{var}(\mu, \mu') = \frac{1}{N} \sum_{t=0}^{N-1} \int_{s \in S} \left|\mu_t(s) - \mu'_t(s)\right|$$

Overall, this is a fairly weak result for a couple of reasons. First, the value function is an encoding of the policy, but it is really the relative values of the value function that define the policy. Thus, knowing a relation to the value function only tells part of the story. Second, in practice there is no way to tell how good the approximation to the actual value function you have.

Classifier Choice

There are a number of classifiers that could be chosen for use in PSDP: neural networks, Gaussian processes, and support vector machines (SVMs) are a few that we might consider. For our purposes in this work, we chose to leverage the classifier that has the least parameters to tune and seems to perform the best on a variety of problems: a Support Vector Machine [3, 6].

Support vector machines start from the principle that the decision boundary which maximizes the space between classes is likely to be the best one. In its simplest form it simply has a linear decision boundary. However, by using a *Kernel trick* it is possible to extend support vector machines to non-linear decision boundaries [19].

The optimization performed to find the best decision boundary can be formulated as a quadratic program that is relaxed to penalize errors. Among the parameters that are left free includes a term to account for the number of errors that will be allowed as well as the kernel parameters.

The basic support vector machine performs classification for two classes, and must from this basis, be extended to many classes. Though it would be nice to perform this optimization jointly over all the classes, how to perform this optimization remains an open question. In lieu of such an optimization, a cascade of classifiers is typically trained in a *one-versus-all* or *one-versus-one* manner. In the one-versus-all framework, one class is trained against all of the others. The one with the highest value is taken to be the correct class. In the one-versus-one framework, each class is trained against the set of inputs that only contain one output class. In this way, each classifier will vote for which class it has most support for. The class with the most votes wins the classification and is taken to be the true class.

In [15], it was shown that the one-versus-one framework is more suitable for most problems, so we use it here. In our optimization, we will also want to rule out trajectories that collide with obstacles. To do this we simply will skip over votes that are from infeasible trajectories. In such a way we can get the next optimal trajectory without having to retrain for each particular circumstance.

4.3 Policy Search for SLAM Trajectory Optimization

At this point, we have the tools necessary to formalize the SLAM exploration problem as an MDP, and to solve it using PSDP. We will start with the trajectory optimization problem and work towards a solution to the general form of the exploration problem, both in information theoretic terms and in sse terms. In this chapter we will start with the problem of finding trajectories given that you know the locations you want to visit. In the second half of the chapter, we will progress to the problem of finding the locations to visit as well.

4.3.1 Overview

The first control problem we wish to solve is how to generate a motion trajectory for the robot from its current estimated pose (x, y, θ) to a destination position (x_g, y_g) (or sequence of destinations, $(x_g^0, y_g^0, x_g^1 y_g^1 \ldots, x_g^n, y_g^n)$. We therefore need to define a state space, action space and reward function.

During exploration, a planner can only observe the current estimate of the state, rather than the true state of the world, which is a form of partial observability that often leads to computational intractability in finding good plans. However, we will assume a high-level planner has chosen waypoints for information gathering, and will focus on the problem of controlling the robot's uncertainty during travel between each waypoint. Because we can directly observe the state of the exploration process at all times, we can phrase this control problem as a Markov Decision Process (MDP).

For the trajectory control problem, the robot has a continuous state space (poses) and action space (motor commands). These issues are not necessarily fatal in that using intelligent discretization techniques for continuous state and action spaces have been used with success in other MDP settings. Most importantly, however, we do not have a closed form representation of the reward function, as the reward of each state and action is effectively a predicted EKF update from the robot pose for the specific action. Instead of solving for the optimal policy directly, we turn to reinforcement learning to map the current state to an appropriate action. We will show that our policy optimizes the multi-step robot trajectory and reduces the overall map error.

4.3.2 State Space

The state space of the controller is the current robot pose (x, y, θ), the EKF filter state μ and Σ, a sequence of destinations $(g_x^0, g_y^0, g_x^1 g_y^1 \ldots, g_x^n, g_y^n)$ chosen by some high-level exploration planner

Table 4.1: State space

r	$g_1 \dots g_k$	$l_1 \dots l_n$

to maximize information gain [1]. Note that by our choice of PSDP, we will decompose the larger reinforcement learning problem into a series of horizon-one problems and rely on PSDP to learn a controller that generates trajectories that move smoothly across a sequence of waypoints.

4.3.3 Action Space

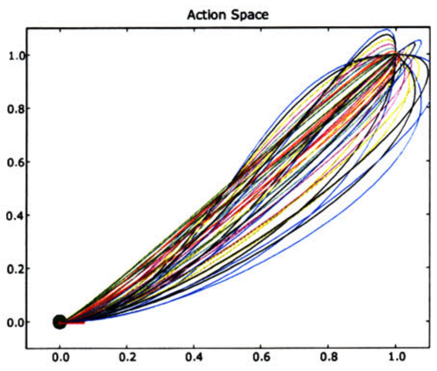
In order to express motion between waypoints as a one-step learning problem, we cannot use direct motor commands as the actions. We instead use as the action space the simplest polynomial that both interpolates the start and end goal and allows us to constrain the start orientation. This is the space is of cubic splines between the current location of the robot and the sequence of information waypoints. (A cubic spline consists of a set of cubic polynomials interpolating a set of control points.) The spline is fixed such that the derivative of the spline at the start control point matches the orientation of the robot at its current position. The orientation with which the robot arrives at the goal point and the magnitude at the start and end locations are not fixed. The spline is parametrized as follows ($t \in [0, 1]$):

$$\begin{aligned} x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x \\ &= a_y t^3 + b_y t^2 + c_y t + d_y \end{aligned}$$

Using this parametrization and providing values for the x and y positions $(x, y)_{t=0}$, $(x, y)_{t=1}$ and derivatives $(\frac{\partial x}{\partial t}, \frac{\partial y}{\partial t})_{t=0}$, and $(\frac{\partial x}{\partial t}, \frac{\partial y}{\partial t})_{t=1}$ at $t = 0$ and $t = 1$, one can solve for the specific constants. Although there are nominally eight total parameters, five of these parameters are fixed. $(x, y)_{t=0}$ is the start position of the robot and $(x, y)_{t=1}$ is the destination position (x_g, y_g). Further, we maintain the orientation constraint of the robot by requiring that

[1]For the purposes of clarity, and without loss of generality, we describe the system in Euclidean co-ordinates. In practice, we implemented the state representation in polar co-ordinates in the robot frame of reference, reducing the number of variables required to express the joint robot pose and goal position.

Figure 4.3.1: The action space



$$\frac{\frac{\partial y}{\partial t}_{t=0}}{\frac{\partial x}{\partial t}_{t=0}} = \tan(\theta)$$

Thus, we have one free parameter at the start point (the magnitude of the derivative at $t = 0$) and two free parameters (the orientation of arrival and the magnitude of this ratio) at the destination. The kinematic trajectory of the spline does not directly provide motor control commands. To convert the spline into an actual motor sequence, we use a simple $P - D$ control loop on the current robot position estimate and a point on the spline a distance d in front of the robot.

4.3.4 Reward

The reward for each action is either the information gain or the squared error as discussed earlier. These objective functions are computed with respect to all of the state variables, including the robot pose. For example, if $u_{t=k}$ is the original uncertainty and $u_{t=k+1}$ is the final uncertainty, then $r_{t=k} = u_{t=k} - u_{t=k+1}$, encouraging the policy to introduce as little uncertainty as possible through the motion of the robot.

Since we are using an Extended Kalman filter, the posterior is given by a Gaussian with covariance Σ and mean μ, and a good measure of uncertainty in a Gaussian distribution is information gain; the use of the trace (in contrast to the determinant) for information gathering in exploration problems has been shown elsewhere to improve performance [31].

4.3.5 Classifier

The specific policy learner we use for each one-step policy is the multi-class Support Vector Machine. This algorithm is a discriminative learner that assigns a label (i.e., optimal action) to each instance to be classified. We use an SVM to learn our one-step policy for two reasons: the SVM allows us to learn a classifier over an infinite input space, which allows us to use a continuous state space, and the SVM is generally an efficient learner of large input spaces with a small number of samples. However, the SVM can only learn to label instances from a discrete set of labels; since we are associating optimal actions with each state, we must discretize our action space appropriately.

PSDP also requires that we have a proposal distribution from which to sample initial states. For our purposes here, we sample uniformly over the possible destinations. It should be noted that the PSDP algorithm produces a non-stationary deterministic policy, which is a more general class of policy than is usually used and PSDP is known to minimize approximation errors compared to value function techniques. The complete PSDP algorithm is given in figure 4.2.3.

4.3.6 Obstacles

Because of the way that multiclass-SVMs are implemented, we are able to efficiently exclude actions that would otherwise run into obstacles. As discussed before, a multiclass SVM is usually created from a number of binary SVMs and the two most commonly implemented strategies are the *one-versus-all* and *one-versus-one*.

For each time, a multiclass SVM is queried for the best action to take. When the best action is not able to be taken because it will run into an obstacle, it is eliminated and its binary classifiers are excluded from voting. Actions are eliminated in order of best to worst until some action is found that does not run into the obstacle. It is of course assumed that the action set is rich enough to capture the space of actions needed to move from one point to another for every world. While this is may the case, in practice we had few problems with obstacles using this strategy.

4.4 Conclusions

In this section, we introduced the Markov Decision Process formalism. We discussed Dynamic Programming as one method to solve a Markov Decision Process. Finally we discussed a policy search method that is very similar to dynamic programming, but can be used for reinforcement learning as well. We then discussed what PSDP can guarantee and we discussed which classifier to choose. In the next chapter we will we will discuss experimental results.

Chapter 5

Experimental Setup and Results

For these experimental results we perform an iterative set of experiments with progressively real simulations. To understand what the best objective function is, we perform a one-step comparison of the trace, determinant, and sum squared error. In general, there is no reason to believe the trace and determinant will be significantly different. One would expect the squared-error to remove bad data associations and divergences of the map, and therefore will likely be slightly better.

It should be noted that all of these results are compared to a hand controller that performs a point turn to face the destination and then translates there. In particular, we will perform three different simulated experiments, in order of increasing model, mapping, and simulator sophistication:

1. **Simulation 1:** Simple simulator of feature measurements, simple EKF, simple motion model, libsvm.

2. **Simulation 2:** Simple simulator of feature measurements, complex EKF, non-standard motion model, libsvm.

3. **Simulation 3:** Complex simulator for raw range readings, feature extraction and complex EKF, non-standard motion model, custom svm.

The specific assumptions of each experimental setup will be discussed in their respective sections.

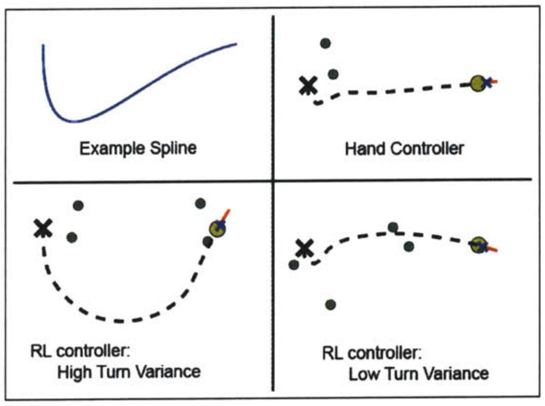

Example Spline
Hand Controller
RL controller: High Turn Variance
RL controller: Low Turn Variance

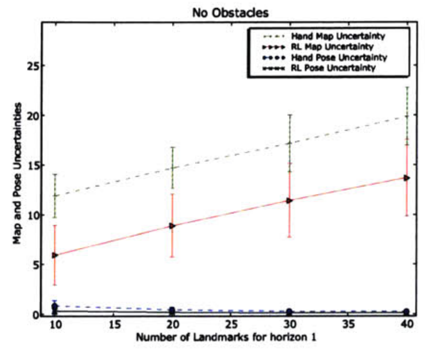
Figure 5.0.1: Examples of controllers for different dynamics. Top left: is an example spline action. Bottom left: an example trajectory chosen by the controller when the error introduced by rotations is large. Top right: an example distance-optimal (hand) controller. Bottom right: an example trajectory chosen when there is little error introduced by rotations.

5.1 Simulation one

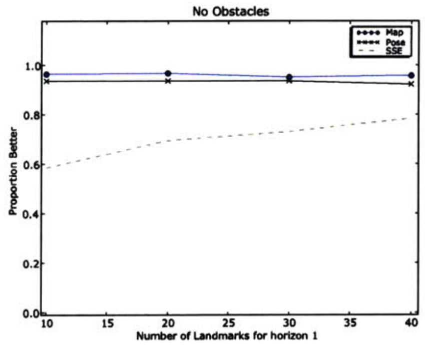
This is the simplest experiment that was performed. In this set of experiments, features were simulated and directly associated in the Kalman filter. The motion model described in previous sections was not implemented in lieu of a simpler model that is identical to the one described in Chapter 2, but without slip. The parameters of the motion and measurement models for the simulator and the EKF were chosen by hand arbitrarily. Further, it was assumed that all of the features could be seen all the time.

Policy search by dynamic programming was used to learn both single and multi-step policies. The world was assumed to be of size 35×35 meters, so that features would be no more than this distance away from the robot. Each policy was learned using a support vector machine with a Gaussian kernel, $C = 5$, and $\gamma = 0.2$. These parameters were computed by doing a parameter search with one step input/output pairs. Each learning problem used $m_1 = 1,000$ samples with $m2 = 1$ roll-out samples for each start sample.

A total of 160 actions for each sample were considered by sampling from the space of possible cubic splines. Both the landmark locations and the goal positions were chosen at random. Training each horizon-one problem with 1,000 samples takes about an hour. The time to classify actions using an SVM takes milliseconds, making it viable for use in online exploration.



(a) Map and Pose Uncertainties



(b) Proportion of times RL is better than the Hand-controller

Figure 5.1.1: A comparison over all destinations of the RL controller and the hand controller (a) in terms of pose and map uncertainty and (b) in terms of the proportion of time that the RL controller is better for various numbers of randomly selected landmarks.

5.1.1 Policy Adaptation

In Figure 5.0.1 we can see two trajectories learned using reinforcement learning. Note here that only the destination was a part of the state space, so that there are no confounding factors from the landmarks. The difference between these figures is the robot model that the EKF and simulator use. In the bottom left figure, a motion model with high rotational variance was used, thereby discouraging fast turns. In the bottom right figure a model is used that has high translational variance in the noise, thereby encouraging short travel distance and high rotational velocity. Comparing these, it becomes clear that the policy is being adapted to the motion model.

Comparing the distance-optimal controller (top right) and the low rotational variance controller (bottom right), there is little difference. This indicates that when high rotational velocity is not penalized by the simulator, the policy converges to the distance-optimal solution.

5.1.2 One-step policies

Looking at figure 5.1.1(a), we can see the improvement over the distance-minimizing hand controller in terms of the overall map and pose uncertainties. In figure 5.1.1(b), it is clear that the map and pose uncertainties are better than the hand controller nearly 98% of the time. Further, up to

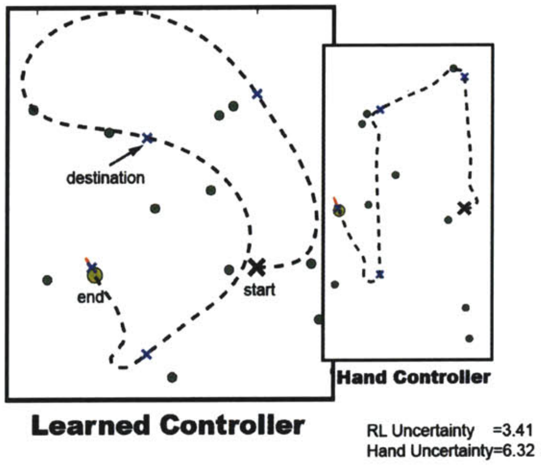


Figure 5.1.2: Exploration trajectories: a sequence of 4 exploration points are given by a high-level planner for exploration. On the left is the true four-step trajectory generated from the learned controller. On the right is the trajectory of the distance-minimizing controller.

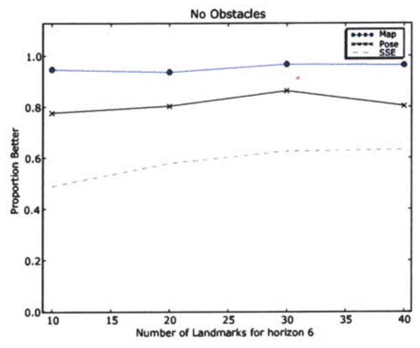
80% of the time the RL controller will obtain better overall sum squared error in the maps. Note that figure 5.1.1 reflects destinations at least 7 meters from the current location of the robot and randomly selected landmarks.

These results show that minimizing the one-step pose uncertainty indeed provides trajectories that will help to minimize the total map uncertainty as well as the sum squared error. Further, as the number of landmarks increases, the proportion of times the sum squared error is better increases. The salient feature of figure 5.1.1 is the difference between controllers.

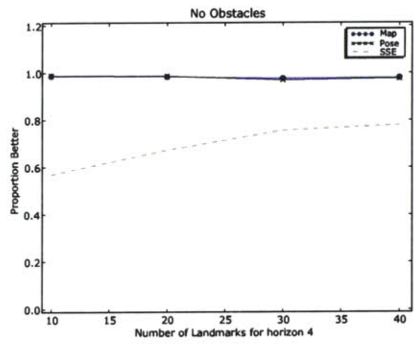
5.1.3 Multistep policies

The creation of multi-step policies takes polynomial time in the number of steps that we want to optimize. In particular, the number of simulation steps that we have to perform is $O(\frac{1}{2}n(n+1)akv)$, for n the horizon, a the number of actions, m_1 the number of samples at a given timestep and m_2 the number of approximations of the value function that we need to perform.

Since the computation of the t^{th} policy takes t^2 simulation steps for a fixed problem, the bottleneck of generating a multistep policy resides in the training. Computing a one-step policy corresponds to simply setting $t = 1$, and therefore is significantly faster than generating a true multi-step



(a) Iterated one-step policy for horizon 6



(b) True four-step policy for horizon 4

Figure 5.1.3: Comparison of a (a) one-step policy run for 6 destinations and (b) a true 4-step policy.

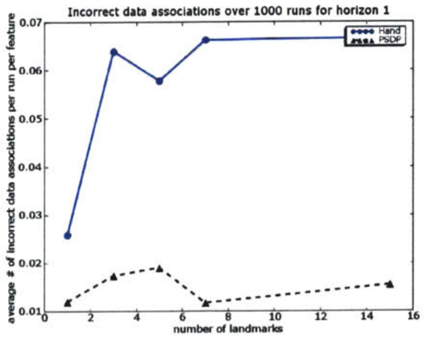
policy. In our experiments, the total training time for a one-step policy is about 30 minutes, while a four-step policy can take 5 hours. The training of each SVM is trivial; in our experiments it takes well less than a minute per SVM. Finally, when we need to evaluate a state (perform classification using an SVM), the computational time remains at milliseconds, making it viable for use in online exploration.

As a proxy for investing the computational time into creating an n step policy for an n step problem, we decided to compare the benefits of using a true four-step policy over repeating the one-step policy six times. Further, we perform a comparison directly to the hand controller for each set of landmarks and destinations. Figure 5.1 shows an example trajectory chosen by the controller, and the corresponding trajectory generated by the hand controller. In this example, the uncertainty of the map learned by the RL controller compared favorably to the map learned by the hand controller.

In Figure 5.1.3(a) we used a 1 step policy over 6 steps and compared this to the hand controller. We can see a reduction in the sum squared error some of the time, the pose uncertainty about 80% of the time and the map uncertainty almost all of the time. This shows that a one-step controller can improve overall uncertainty somewhat.

For the true four-step policy, we performed a quantitative comparison of the learned policies and hand-crafted controllers. We used $m_1 = 1000$ samples and we test on 300 trials consisting of four steps each, where actions are chosen either by the hand controller or by the learned RL policy.

Figure 5.2.1: Simulation two number of bad data associations for the squared error objective function



Here we used a true four step policy and not a four step policy that is generated via chaining four one-step policies. The results are presented in Figure 5.1.3(b). Clearly, the RL controller is doing better than the hand controller.

Comparing Figure 5.1.3(a) and 5.1.3(b) we see that the true 4 step policy results in a reduced map uncertainty nearly all of the time and lower sum squared error in the map up to 80% of the time. In contrast to the iterated one-step policy, we notice that the true 4-step policy significantly outperforms the iterated 1-step policy.

These results indicate that the SLAM problem can greatly benefit from the use of trajectory planning as we have presented it here. In particular, when taking into account the next n destinations, the RL will produce a policy much better than a standard hand tuned policy.

5.2 Simulation two

In the second simulation, we improved the EKF with all of the details discussed in Chapter 2. The particular elements which make this a significant improvement when estimating the covariance. Here the motion model was learned for our robot and therefore the controller that we get out will be directly usable on our real robot. Further, the addition of new features into the filter was performed in a more accurate way.

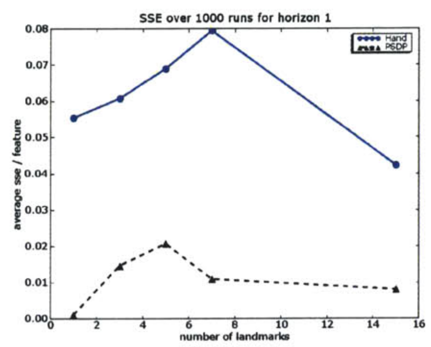
In this set of experiments, the SVM parameters were tuned via a parameter search over C and σ^2

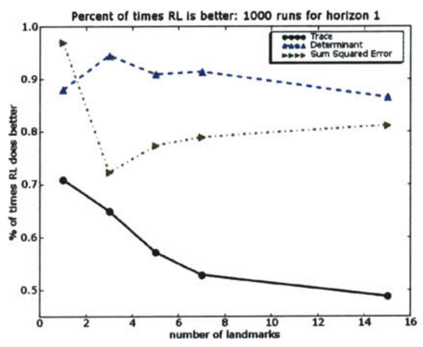
over one-step samples. There were 500 samples for training and around 300 monte carlo simulations for the experiments. The number of actions was reduced to 50, due to increased computational complexity.

The one-step results are summarized in Figures 5-3(a), 5-3(c), and 5-3(b). We can draw the conclusion that the learned one-step policy tends to minimize the squared error. This corroborates the results of the simulation one. As expected, it appears from Figure5-3(c) that the squared error objective performs slightly better than the information-based metrics in terms of raw squared error and also the percentage of times it provides a better solution.

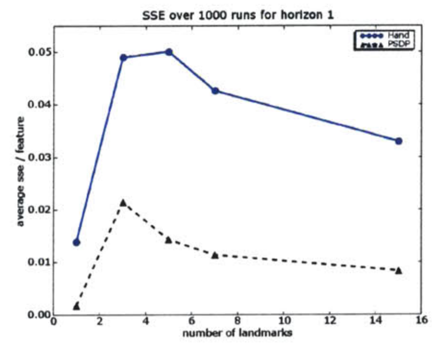
It was also discovered that there are fewer data association errors with the learned controller. By looking at Figure 5.2, we can see that the number of bad data associations was reduced by a significant amount. This particular graph is for the squared-error objective function, but the information-based objective functions look similar.

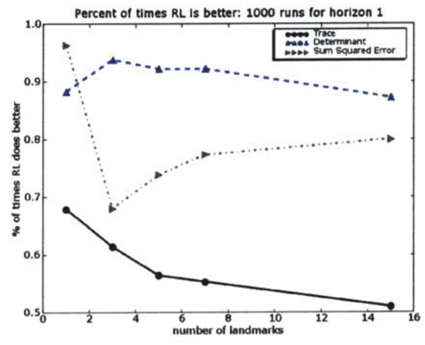
Figure 5.2.2: Simulation two one-step results



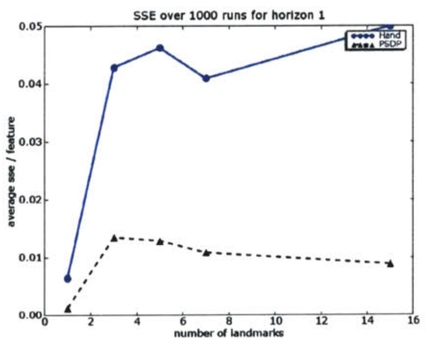


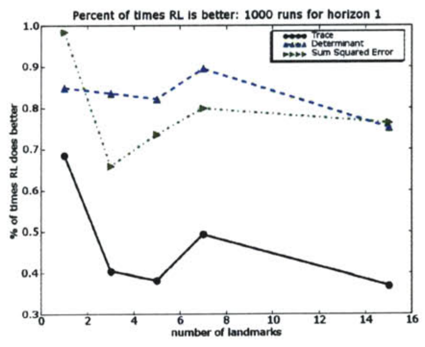
(a) Trace as the objective function





(b) Determinant as the objective function





(c) squared error as the objective function

5.3 Simulation three

The third simulation is the most realistic among the simulations, though some details still need to be improved. In this experiment, gridmaps of actual office buildings are loaded and hurdle landmarks are automatically added. A simulator uses this map to ray trace the expected measurements for each pose in the map, and motion and measurement updates are performed with respect to the model presented in Chapter 2. The experimental setup can be seen in Figure 5.3.

An EKF with the non-standard motion model is used for tracking and point features are extracted from the raw range measurements using a reliable *hurdle extractor*. Data association is performed, and measurements correctly as described in Chapter 2. A gridmapper was implemented using CARMEN and trajectories are planned with respect to this map.

Other parts of the subsystems were upgraded as well, including a multiclass one-versus-one ν-SVM to generate trajectories that do not collide with obstacles. Alternative actions may thus be selected without retraining the SVM, as described in 4.3.6. There were again 50 actions to select among.

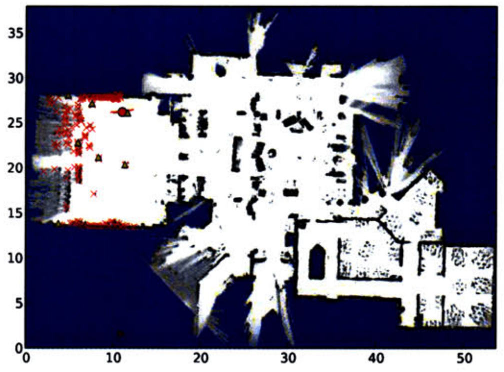
The preliminary results for the third simulation are shown in Figures 5-3(a) 5-3(b) and 5-3(c). The results are inconclusive, though it does appear that the squared error of the hand-controller is about the same as the squared error of the reinforcement learner. The spikes are from one or two divergences in the EKF mapper. Most of the time there is a 50 percent chance that the RL controller will be better than the hand controller.

There are a few reasons why these results may not be positive. First, the computation of squared error needs improvement, since there will be features added to the EKF that do not correspond to the map and there are features in the map that the EKF may not see. A one-to-one matching needs to be performed between the two. Second, a parameter search for the ν-SVM needs to be performed.

5.3.1 Conclusions

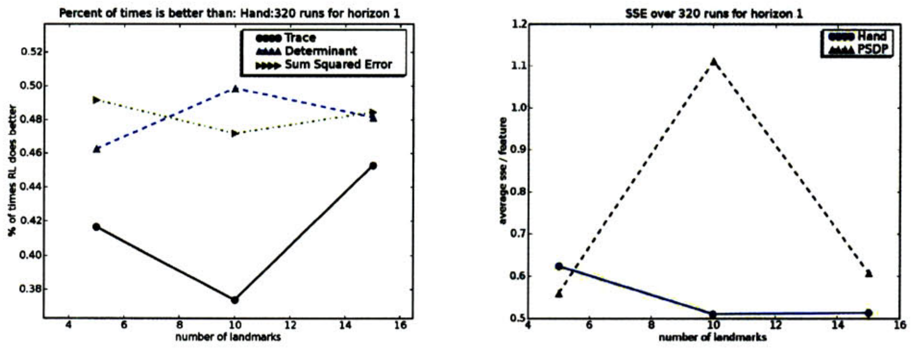
In summary, from the first two experiments it is clear that trajectory optimization definitely causes the policy to adapt based on the motion model of the robot, that this adaptation will reduce the squared error in the map, and that better data associations are made with the learned policy. There remains much work to be done, however, to show that this will work in the real world.

Figure 5.3.1: Simulation example trajectory: red xs are simulated measurements, yellow triangles are EKF landmark locations, red circle is the pose of the robot. Hurdles have been added as small dots on the map.

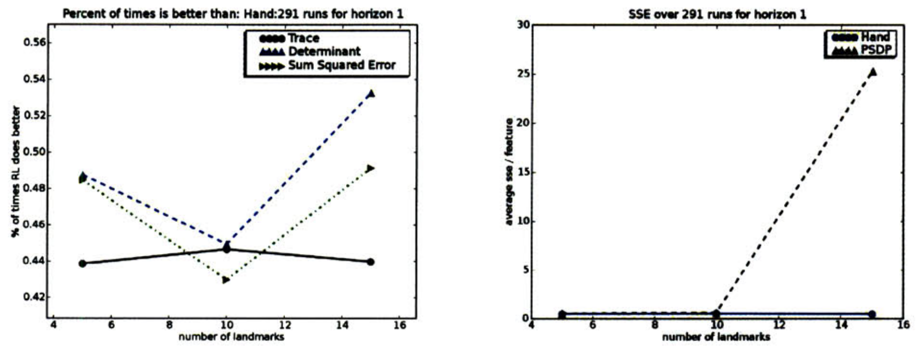


Repeating the single and multi-step experiments of simulations one and two in the third experimental setup remains one of the current difficulties. Given the results of simulation three, it is yet unclear whether the adaptation of the policy causes significant improvement in the map in realistic environments. There are a number of specific improvements that, to date, have yet to be implemented and tried that may correct this shortcoming.

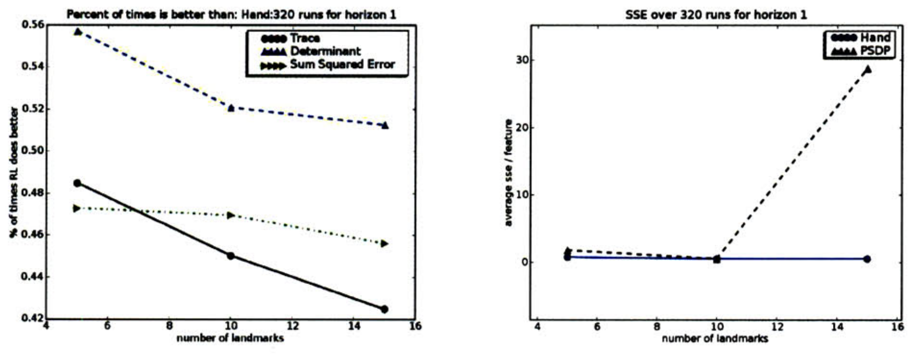
Figure 5.3.2: Simulation three one-step results



(a) Squared error



(b) Trace



(c) Determinant

Chapter 6

Conclusions and Future Work

In this work a novel approach for integrated action selection for trajectory planning has been shown. We have performed accurate integration of measurement and motion uncertainty into the covariance and have explored the posterior trace and covariance of robot motion. Further, we have defined the trajectory exploration problem and formalized it as a Markov Decision process and shown how to solve it using policy search. In solving the problem this way, we have presented a novel solution to generate k-step trajectories for a mobile robot.

Given destination points for information gathering, we have shown that reinforcement learning can be used to learn a control policy that finds actions to minimize the uncertainty and squared error in the map. These control policies currently work only in a simple simulator, but we hope to extend them to much more complex environments.

The results for the simpler simulations show quite a bit of promise. The map showed significant improvement in these situations, indicating that the underlying problem exists in simulation and in the real world. The third simulation still needs more analysis, and though the results are currently negative, we hope to find and overcome the problems.

Though squared error was shown to be a slightly better objective function, information-based metrics tend to be fairly robust and can be computed without a simulator. All reward functions were found to be good heuristics for generating better maps.

Data association is often a problem when building large maps. Often many landmarks will become associated incorrectly into the state. In our experiments, data association errors decreased significantly given only a difference in the robot controller, indicating that our optimization has improved the measurements of the robot and thereby improved the data association.

While other forms of reinforcement learning are often environment specific, the algorithm we have presented can generalize to many real-world environments. Given a new world, and the assumption that at least one of the actions in the action space is feasible, then an optimal motion plan given those actions may be performed. Further, by using machine learning, and given a new robot with different sensor and motion models, we can run these algorithms directly and recover an optimal policy.

6.1 Future Work

By far the biggest drawback is that our results are only in simulation. It is our goal in the near future to add the capability of performing this trajectory optimization offline and running the policy online on real robots to optimize the map-building process.

Further, the results shown in Section 5.3 are very preliminary and need to be confirmed. Though they indicate that the learning may help little, it is entirely possible that certain aspects of the implementation can be improved significantly. There is a very real phenomenon of point turns causing problems with mappers, and if the RL does not improve the estimates, then we hope to find why this might be the case.

More broadly, we hope to extend these techniques to the full exploration problem and begin to perform destination selection together with trajectory selection. We believe this to be possible, but have not yet obtained results toward this end. Finally, new domains will be explored, from helicopters to legged robots.

Chapter 7

Appendix

7.1 Alternative Motion Models

Definition 7.1.1. The *Velocity Motion Model* described directly in terms of the translational velocity v, rotational velocity ω, and the change in time dt.

$$\begin{pmatrix} x_e \\ y_e \\ \theta_e \end{pmatrix} = \begin{pmatrix} x_s - \frac{v}{\omega} sin(\theta_s) + \frac{v}{\omega} sin(\theta_s + \omega\, dt) \\ y_s + \frac{v}{\omega} cos(\theta_s) - \frac{v}{\omega} cos(\theta_s + \omega\, dt) \\ \theta_s + \omega\, dt \end{pmatrix}$$

Alternatively, this can be written in terms of the above equations as:

$$\begin{pmatrix} x_e \\ y_e \\ \theta_e \end{pmatrix} = \begin{pmatrix} x_s - \frac{dr}{d\theta_1 + d\theta_2} sin(\theta_s) + \frac{dr}{d\theta_1 + d\theta_2} sin(\theta_s + d\theta_1 + d\theta_2) \\ y_s + \frac{dr}{d\theta_1 + d\theta_2} cos(\theta_s) - \frac{dr}{d\theta_1 + d\theta_2} cos(\theta_s + d\theta_1 + d\theta_2) \\ \theta_s + d\theta_1 + d\theta_2 \end{pmatrix}$$

Definition 7.1.2. Extending the previous motion model with a term for slip, the *Velocity Motion Model with slip* is described by the following equations:

$$\begin{pmatrix} x_e \\ y_e \\ \theta_e \end{pmatrix} = \begin{pmatrix} x_s - \frac{dr}{d\theta_1+d\theta_2} sin(\theta) + \frac{dr}{d\theta_1+d\theta_2} sin(\theta + d\theta_1 + d\theta_2) + ds\, cos(\theta + d\theta_1 + \frac{\pi}{2}) \\ y_s + \frac{dr}{d\theta_1+d\theta_2} cos(\theta) - \frac{dr}{d\theta_1+d\theta_2} cos(\theta + d\theta_1 + d\theta_2) + ds\, sin(\theta + d\theta_1 + \frac{\pi}{2}) \\ \theta_s + d\theta_1 + d\theta_2 \end{pmatrix}$$

7.2 Matrix Properties

These theorems about matrices are mostly reproduced from [11]. Please refer there for more details.

Definition 7.2.1. Let $A, B \in M_{nxn}$. B is similar to A if there exists an invertible matrix Q such that $B = Q^{-1}AQ^T$.

Knowing that A and B are similar matrices gives us a condition for when their traces are equivalent.

Theorem 7.2.2. *If A and B are similar matrices, then $tr(A) = tr(B)$.*

Further, the trace and the determinant of the covariance matrix can be related to the eigenvalues of the matrix.

Theorem 7.2.3. *Let A be an n by n real symmetric matrix, then $tr(A) = \sum_{i=1}^{n} \lambda_i$ and $tr(A^*A) = \sum_{i=1}^{n} |\lambda_i|^2$, where the λ_i are not necessarily distinct eigenvalues of A.*

The determinant is characterized by the product of eigenvalues in the next theorem.

Theorem 7.2.4. *Let A be an n by n symmetric normal matrix, then $det(A) = \prod_{i=1}^{n} \lambda_i$, for λ_i not necessarily distinct eigenvalues of A.*

Theorem 7.2.5. *From [13], the* Sherman-Morrison-Woodbury formula *or* Matrix Inversion lemma *is stated as follows. For any invertible quadratic matrices A and C, then:*

$$(A + UCV)^{-1} = A^{-1} - A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}$$

7.3 Sampling from a Gaussian

Assume we have a Gaussian random variable x with mean μ and covariance Σ. For now, assume that μ is 0. Then, we want $x = Ry$, for R some matrix and y a Normal random vector with variance 1 and correlations 0. We want to discover what R should be. The following is a necessary and sufficient condition to find R such that $x = Ry$:

$$\begin{aligned} x^T\Sigma^{-1}x &= y^TIy \\ x^T(RR^*)^{-1}x &= y^TIy \\ x^T(R^*)^{-1}R^{-1}x &= y^TIy \end{aligned}$$

Thus, it is clear that $R^{-1}x = Iy$ such that:

$$x = Ry$$

Note that R and R^*can be computed by taking the Cholesky Decomposition of the covariance matrix. R should be lower diagonal. An alternative derivation with the starting point at $x = Ry$ is as follows (noting that the conjugate $*$ of a real matrix is simply the transpose):

$$\begin{aligned} x &= Ry \\ R^{-1}x &= y \\ R^{-1}x &= Iy \\ x^T(R^*)^{-1}R^{-1}x &= x^T(R^*)^{-1}Iy \\ x^T(RR^*)^{-1}x &= x^T(R^{-1})^TIy \\ x^T\Sigma^{-1}x &= (R^{-1}x)^TIy \\ x^T\Sigma^{-1}x &= y^TIy \end{aligned}$$

Thus, it should now be clear that, since we know that $x^T\Sigma^{-1}x = y^TIy$, then R must come from the Cholesky Decomposition of the covariance matrix Σ.

7.4 Extended Kalman Filter

Theorem 7.4.1. *If $P(X_t|X_{t-1})$, $P(Y_t|Y_{t-1})$ and the prior distribution over X_0 are all Gaussian and the X_t is a* linear *function of X_{t-1} for all t, then bel(X_t) for all t will be Gaussian. Further, the updates are as shown as in Figure 2.4.2.*

Proof. We again prove this by induction on the proposition that: If $bel(X_t)$ is Gaussian given the above conditions. We also assume that the prior distribution $P(X_0)$ is Gaussian, therefore the base case holds. At this point we assume that $bel(X_{t-1})$ is Gaussian and we need to prove that $bel(X_t)$ is Gaussian.

We must engage ourselves in two steps, each corresponding to a step in the Bayes filter and prove to ourselves that the result remains Gaussian. First, we must show that after marginalization, the result remains Gaussian. Thus, we assume that $bel(X_{t-1})$ is Gaussian and we need to show that $\overline{bel}(X_t)$ is Gaussian.

For the time projection step, we want $\overline{bel}(X_t) = \int_{X_{t-1}} P(X_t|X_{t-1})bel(X_{t-1})$. By the inductive hypothesis, $bel(X_{t-1})$ is Gaussian and we parametrize it by Σ_{t-1} and mean μ_{t-1}. The transition probability $P(X_t|X_{t-1})$ is parametrized by $\mu_t = g(\mu_{t-1})$. We will call G_t the Jacobian of $g()$.

Similar to [34], the trick we will employ is to break up the exponent of the probability distribution into two pieces: L_R which depends on both X_t and X_{t-1}, but is a manageable integral, and L_P that depends only on X_t, but that we can factor outside the integral. We will then find the moments of this quadratic function and we will have the solution.

$$\begin{aligned}
\overline{bel}(X_{t-1}) &= Z_1 \int_{X_{t-1}} e^{-\frac{1}{2}(L_P+L_R)} \\
&= Z_1\, e^{-\frac{1}{2}(L_P)} \int_{X_{t-1}} e^{-\frac{1}{2}(L_R)} \\
&= Z_2\, e^{-\frac{1}{2}(L_P)}
\end{aligned}$$

To start, we write the exponent of the probability distribution that we are after.

$$\begin{aligned} L &= (x_{t-1}-\mu_{t-1})^T\Sigma_{t-1}^{-1}(x_{t-1}-\mu_{t-1}) \\ &\quad +(x_t-g(\mu_{t-1})-G_t(x_{t-1}-\mu_{t-1}))^T W_t^{-1}(x_t-g(\mu_{t-1})-G_t(x_{t-1}-\mu_{t-1})) \end{aligned}$$

We will choose L_R rather using the first and second moments of this distribution, creating another Gaussian distribution from which we can easily integrate out the interior to a constant. Through a bit of work that is shown in the Appendix and the benign insight that for quadratic forms in the exponential the first derivative is the mean and the second derivative is the inverse covariance matrix, it can be shown the relevant partial derivatives become:

$$\frac{\partial L}{\partial x_{t-1}} = \mu_\xi = \Sigma_\xi[G^T W^{-1}(x_t - g(\mu_{t-1}) + G\mu_{t-1}) + \Sigma_{t-1}^{-1}\mu_{t-1}]$$
$$\frac{\partial^2 L}{\partial x_{t-1}^2} = \Sigma_\xi^{-1} = \Sigma_{t-1}^{-1} + G^T W_t^{-1} G$$

This corresponds to the Gaussian distribution associated for a fixed X_{t-1}. Let this distribution be L_R. Note that we can easily integrate over the extents of a Gaussian distribution to get and integral of 1, or if unnormalized, an integral of $det(2\pi\Sigma)^{1/2}$. After a bit more math, we can obtain $L_P = L - L_R$, which is:

$$\begin{aligned} L_P = \;& (x_t-g(\mu_{t-1})-G_t(x_{t-1}-\mu_{t-1}))^T W_t^{-1}(x_t-g(\mu_{t-1})-G_t(x_{t-1}-\mu_{t-1})) \\ & +\mu_\xi^T\Sigma_\xi^{-1}\mu_\xi + \mu_{t-1}\Sigma_{t-1}^{-1}\mu_{t-1} \end{aligned}$$

Thus, we now have L_D, which we can integrate out and L_P, which we can factor out of the integral. It should be clear that L_P is a quadratic form, and therefore the posterior is a Gaussian. We now find the form of the Gaussian by taking the partial derivatives. Using the Inversion Lemma, stated in the Appendix as Equation 7.2.5, we obtain the first partial derivative, from which the second order partial should be obvious:

$$\frac{\partial L_P}{\partial x_t} = [W + G\Sigma_{t-1}G^T]^{-1}(x_t - g(\mu_{t-1}) + G\mu_{t-1}) - W^{-1}G\Sigma_\xi\Sigma_{t-1}^{-1}\mu_{t-1}$$
$$\frac{\partial^2 L_P}{\partial x_t^2} = [W + G_t\Sigma_{t-1}G_t^T]^{-1}$$

Setting the first equation equal to zero, we find that $x_t = g(\mu_{t-1})$. Clearly these values correspond to the few steps in the Extended Kalman filter of Figure 2.4.2. We still need to shoq that the second step of the Extended Kalman filter, where we multiply by the measurement probability, is Gaussian. It is not difficult to see, since the exponent remains quadratic, but let us compute the values.

The measurement update is derived similarly. We will first form the probability distribution and then maximize the log likelihood by taking the first and second order partial derivatives. The log likelihood is:

$$\begin{aligned} L_z &= -\frac{1}{2}[(x_t - \overline{\mu}_t)^T \overline{\Sigma}_t^{-1}(x_t - \overline{\mu}_t) \\ &\qquad + (z_t - h(\overline{\mu}_{t-1}) - H_t(x_t - \overline{\mu}_t))^T Q_t^{-1}(z_t - h(\overline{\mu}_{t-1}) - H_t(x_t - \overline{\mu}_t)) + log(Z)] \end{aligned}$$

In what follows, H_t is the Jacobian of the measurement function $h()$, z_t is the measurement and $\overline{\Sigma}_t$, $\overline{\mu}_t$ are the sufficient statistics of $\overline{bel}_t$. Thus, we can take the partial derivatives, obtaining:

$$\frac{\partial L_z}{\partial x_t} = \overline{\Sigma}_t^{-1}(x_t - \mu_t) - H_t^T Q^{-1}(z_t - h(\overline{\mu}_t) - H_t(x_t - \overline{\mu_t}))$$
$$\frac{\partial^2 L_z}{\partial x_t^2} = \Sigma_t^{-1} = [\overline{\Sigma}_{t-1}^{-1} + H_t^T Q_t^{-1} H_t]^{-1}$$

Setting the first derivative equal to 0, we obtain the first relation directly, calling $K_t = \Sigma_t H_t^T Q_t^{-1}$. However, it turns out using a bit of matrix algebra that we can directly rewrite K_t as shown in Equation 7.4.1. By using the Matrix Inversion lemma and plugging K_t into Σ_t, it can easily be checked that the relation is true.

$$\mu_t = \overline{\mu}_t + K_t(z_t - h_(\overline{\mu}_t))$$
$$\Sigma_t = [\overline{\Sigma}_{t-1}^{-1} + H_t^T Q_t^{-1} H_t]^{-1} = (I - K_t H_t)\overline{\Sigma}_t$$
$$K_t = \overline{\Sigma}_t H_t^T Q^{-1} = \overline{\Sigma}_t H_t^T (H_t \overline{\Sigma_t} H_t^T + Q_t)^{-1}$$

Thus, we have shown that given a Gaussian belief, the new belief is Gaussian as well, as long as the update is linear. Thus, by the inductive hypothesis, the belief is Gaussian for all timesteps and the update is given by Equation 7.4.1. Q.E.D. $\square$

For the above algorithm it is assumed that the measurement has already been associated with a state variable. However, associating a measurement to a state variable is nevertheless a difficult problem. In the case of

Bibliography

[1] James Bagnell, Sham Kakade, Andrew Ng, and Jeff Schneider. Policy search by dynamic programming. In *Neural Information Processing Systems*, volume 16. MIT Press, December 2003.

[2] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, 1992.

[3] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[4] Georges A. Darbellay and Igor Vajda. Entropy expressions for multivariate continuous distributions. *IEEE Transactions on Information Theory*, 46(2):709–712, 2000.

[5] Frank Dellaert. Square Root SAM: Simultaneous location and mapping via square root information smoothing. In *RSS*, 2005.

[6] Kai-Bo Duan and S. Sathiya Keerthi. Which is the best multiclass svm method? an empirical study. In *Multiple Classifier Systems*, pages 278–285, 2005.

[7] T. Duckett, S. Marsland, and J. Shapiro. Learning globally consistent maps by relaxation, 2000.

[8] A. Eliazar and R. Parr. Learning probabilistic motion models for mobile robots, 2004.

[9] R. Eustice, M. Walter, and J. Leonard. Sparse extended information filters: Insights into sparsification. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 641–648, August 2005.

[10] H. Feder, J. Leonard, and C. Smith. Adaptive mobile robot navigation and mapping, 1999.

[11] S. Friedberg, A. Insel, and L. Spence. *Linear Algebra: 3rd Edition.* Prentice Hall, Upper Saddle River, NJ, 1997.

[12] Hector Gonzalez-Banos and Jean-Claude Latombe. Planning robot motions for range-image acquisition and automatic 3d model construction.

[13] G. Goulb and C. Loan. *Matrix Computations: 3rd edition.* The Johns Hopkins University Press, Baltimore, MD, 1996.

[14] D. Hähnel, W. Burgard, D. Fox, and S. Thrun. A highly efficient FastSLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2003.

[15] C. Hsu and C. Lin. A comparison of methods for multi-class support vector machines, 2001.

[16] Michael Jordan. *An Introduction to Probabilistic Graphical Models.* Unpublished.

[17] SungJoon Kim. *Efficient Simultaneous Localization and Mapping Algorithms using submap networks.* PhD thesis, Massachusetts Institute Of Technology, 2004.

[18] Andreas Krause, Carlos Guestrin, Anupam Gupta, and Jon Kleinberg. Near-optimal sensor placements: maximizing information while minimizing communication cost. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 2–10, New York, NY, USA, 2006. ACM Press.

[19] Miroslav Kubat. Neural networks: a comprehensive foundation by simon haykin, macmillan, 1994, isbn 0-02-352781-7. *Knowl. Eng. Rev.*, 13(4):409–412, 1999.

[20] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping, 1997.

[21] A. Makarenko, S. Williams, F. Bourgault, and H. Durrant-Whyte. An experiment in integrated exploration, 2002.

[22] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing.* The MIT Press, Cambridge, Massachusetts, 1999.

[23] Peter S. Maybeck. Stochastics models, estimation, and control: Introduction, 1994.

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

[25] Michael Montemerlo, Nicholas Roy, and Sebastian Thrun. Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (carmen) toolkit. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, volume 3, pages 2436–2441, Las Vegas, NV, October 2003.

[26] J. Neira and J. Tardos. Data association in stochastic mapping using the joint compatibility test, 2001.

[27] Edwin Olson, John Leonard, and Seth Teller. Fast iterative optimization of pose graphs with poor initial estimates. pages 2262–2269, 2006.

[28] M. Paskin. Thin junction tree filters for simultaneous localization and mapping.

[29] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley, Hoboken, NJ, 2005.

[30] Robert Sim. *On Visual Maps and their Automatic Construction.* PhD Thesis, 2006.

[31] Robert Sim and Nicholas Roy. Global a-optimal robot exploration in slam. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Barcelona, Spain, 2005.

[32] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. pages 167–193, 1990.

[33] C. Stachniss. *Exploration and Mapping with Mobile Robots.* PhD thesis, University of Freiburg, Department of Computer Science, April 2006.

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

[35] S. Thrun, D. Koller, Z. Ghahramani, H. Durrant-Whyte, and A. Ng. Simultaneous mapping and localization with sparse extended information filters, 2002.

[36] Sebastian Thrun, Yufeng Liu, Daphne Koller, Andrew Y. Ng, Zoubin Ghahramani, and Hugh Durrant-Whyte. Simultaneous Localization and Mapping with Sparse Extended Information Filters. *The International Journal of Robotics Research*, 23(7-8):693–716, 2004.

[37] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(4):376–380, 1991.

[38] E. Wan and R. van der Merwe. The unscented kalman filter for nonlinear estimation, 2000.

[39] Yair Weiss and William T. Freeman. Correctness of belief propagation in gaussian graphical models of arbitrary topology. *Neural Computation*, 13(10):2173–2200, 2001.

[40] B. Yamauchi. A frontier based approach for autonomous exploration, 1997.

[41] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Understanding belief propagation and its generalizations. pages 239–269, 2003.