Trajectory Optimization using Reinforcement Learning for Map Exploration

Thomas Kollar and Nicholas Roy,

***Abstract*—Automatically building maps from sensor data is a necessary and fundamental skill for mobile robots; as a result, considerable research attention has focused on the technical challenges inherent in the mapping problem. While statistical inference techniques have led to computationally efficient mapping algorithms, the next major challenge in robotic mapping is to automate the data collection process.**

In this paper, we address the problem of how a robot should plan to explore an unknown environment and collect data in order to maximize the accuracy of the resulting map. We formulate exploration as a constrained optimization problem and use reinforcement learning to find trajectories that lead to accurate maps. We demonstrate this process in simulation and show that the learned policy not only results in improved map-building, but that the learned policy also transfers successfully to a real robot exploring on MIT campus.

1. Introduction

BUILDING maps has become a fundamental skill that most autonomous mobile robots must have. In order to properly interact with their environment, robots need to track their position over time and plan deliberate motions through the world. A robot that is able to build and update its own maps autonomously will have a tremendous advantage over those robots that rely on external positioning systems and static, pre-made maps. As a result, considerable research attention has been focused on the technical challenges inherent in the mapping problem.

Thomas Kollar is with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, 02139. E-mail: tkollar@mit.edu.

Nicholas Roy is with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA. 02139. E-mail: nickroy@mit.edu.

Early mapping algorithms have addressed the problems of using high-quality range data taken at known positions to build floor plans of indoor environments (Moravec and Martin 1994; Thrun 1998). More recent mapping research has focused on building maps with robots that do not have access to global position information (Smith et al. 1990; Lu and Milios 1997; Hähnel et al. 2003; Dellaert and Kaess 2006; Eustice et al. 2005; Thrun et al. 2005; Olson et al. 2006). Without a global position sensor, the sensor data must be used to simultaneously determine the robot position and update the map as the robot moves around. This inference problem, known as the Simultaneous Localization and Mapping problem (or SLAM), may initially appear computationally intractable, but modern statistical inference and machine learning techniques have led to computationally efficient mechanisms for accurate mapping without global position information. Further, these innovations have led to robots that can map large-scale, outdoor environments with a variety of different sensors and to the creation of commodity open-source community around mapping software (Haehnel et al. 2003; Thrun and Montemerlo 2005; Martinez-Cantin and Castellanos 2005; Martinelli et al. 2007; Pfaff et al. 2007; Davison et al. June 2007).

The next major challenge in robotic mapping is to automate the data collection process in order to build the highest-quality map with the least time and cost. In almost all operational map building systems, the robot records the data that will be used to build the map, but the collection process

has limited autonomy. Path-planning and control are sometimes employed, but human supervision is often required to ensure that the collected data is complete (that is, the robot has observed all relevant parts of the world), and that the robot has collected the most useful data for maximizing the accuracy of the map. Once the data is collected, a map is built during an off-line phase; this map can then be used for localization and planning.

There are clearly multiple possible trajectories that constitute a complete exploration of the environment, but some trajectories are preferable to others. As a simple example, point turns for a holonomic robot are a canonically "bad" motion as most mapping algorithms are sensitive to errors in rotational measurements. The responsibility for avoiding such trajectories almost always falls to the human driver; data collected by a bad (or novice) robot driver will lead to the creation of erroneous maps. Avoiding such problems will require robot control algorithms that can reliably explore their environments and generate accurate maps without human intervention. This will enable exploration in distant environments where closed-loop control by a remote human is impractical.

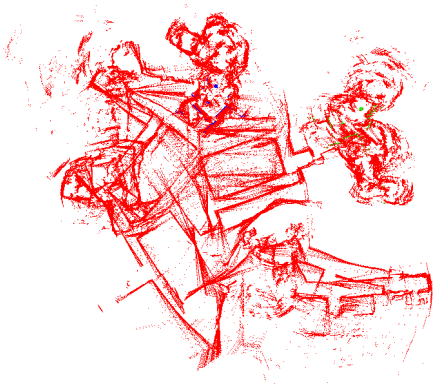
The problem of how to choose sensing trajectories to obtain the best map will be referred to as the *exploration* problem. A good robot exploration strategy must both cover the space sufficiently and collect data that leads to an accurate map. These constraints are to some extent in opposition. A planner that maximizes the environmental coverage will spend relatively little time revisiting previous locations, whereas a controller that minimizes the map error will revisit previous locations regularly to eliminate inconsistencies between different parts of the map. The optimal exploration controller will balance these constraints.

The map-building process for a robot equipped with a laser range scanner shown in Figures 1(a-d) illustrates the need for good exploration. The pose of the robot during each range scan is calculated from the (noisy) odometry measurements without using a SLAM algorithm, and the map is constructed assuming these pose estimates are correct (Figure 1(a)). The red dots in Figures 1(a-c) correspond to measured obstacle positions, and the resulting map is completely unintelligible. If a SLAM procedure is used to estimate the correct robot poses from the sensor data, then a better estimate of the map can be inferred, as shown in Figure 1(b). However, this inferred map contains errors, in particular, the two black arrows are pointing to robot poses in the data where large-scale mapping errors occurred, resulting in map doubling. Notice that outside of these two specific misalignments, most of the map is relatively correct; straight hallways and corners can be clearly seen. For reference, Figure 1(c) shows the corrected map where a human user manually edited the pose information so that the scans aligned, and Figure 1(d) shows the final occupancy grid map that can be used for planning.

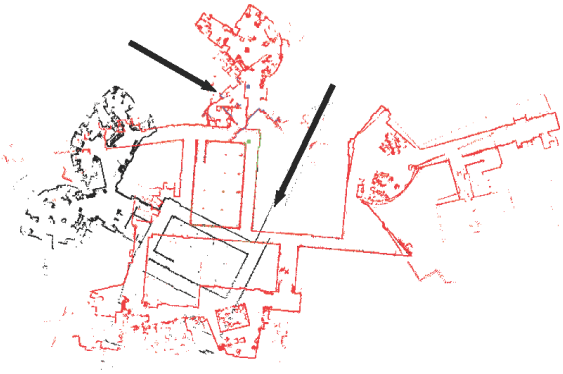
A formal model of autonomous exploration is unfortunately not easy to solve. From a decision-theoretic standpoint, the sequential decision-making problem of exploration is an instance of a continuous-state partially observable Markov decision process (POMDP). Although there has been work in solving continuous POMDPs (Porta et al. 2006; Thrun 2000), the difficulty in scaling these algorithms makes them ineffective in addressing the exploration problem. Even expressing the model parameters of the exploration problem as a continuous POMDP is impractical in real-world domains, never mind solving for a policy.

Without a closed form solution to the objective function, we cannot write down an analytic control law. One alternative strategy would be to use numerical optimization such as model-predictive control, but the computational cost of map prediction eliminates the use of predictive control optimization. This idea has been pursued with some success (Sim et al. 2004; Stachniss et al. 2005), but even a one-step optimization is slow to compute.

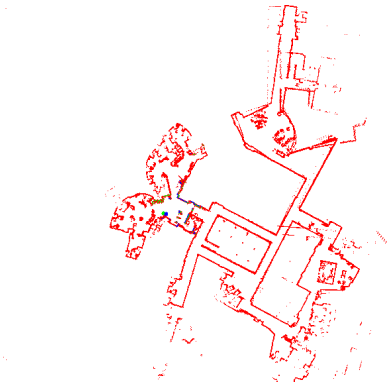
Reinforcement learning has been used successfully to solve large sequential decision making problems in both fully observable and partially observable domains. Many reinforcement learning algorithms rely on the idea that even when the optimal policy cannot be solved analytically, using knowledge of where good policies lie allows the learner to optimize its own performance from experience. It is precisely this idea that we use to find good exploration controllers; although we



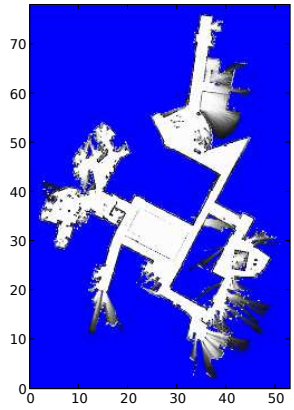
(a) Raw Measurements



(b) Mapping Errors



(c) Corrected Measurements



(d) Occupancy Grid

Fig. 1. Simultaneous Localization and Mapping. In (a)-(c), the red dots are laser range points projected from the estimated or actual robot pose. In (a), the robot odometry is used to calculate the robot pose. The errors in the pose estimate lead to a completely useless map. In (b) a SLAM algorithm is used to estimate the robot trajectory, but the robot trajectory at two locations (shown by the black arrows) puts the true robot pose outside the bounded location assumed by the inference algorithm, causing the mapping to fail. The map is correct on either side of that motion, but the completed map is inconsistent. In (c) and (d), the robot pose and measurements were corrected by hand to generate the globally consistent set of range scans (c) and the corresponding occupancy grid (d). In (d), the black pixel are obstacles, white pixels are free space and blue are unknown.

may not be able to directly optimize the robot trajectory, we can learn from trajectories that have minimized the map error in the past. The advantage to the reinforcement learning approach is that we can produce control policies for an otherwise intractable optimization; additionally, by learning a true policy as opposed to computing a plan, we can amortize the cost of learning over the life-time of the agent. Assuming the agent dynamics do not change substantially, the policy, once learned, can be used to choose control actions in real-time.

In this paper, we frame the exploration problem as a constrained optimization, where the goal is to choose a trajectory that maximizes the map accuracy within the constraints of planned map coverage. Firstly, we describe the mapping problem

and choose an appropriate representation of the inference problem. We then formulate the exploration problem as a sequential decision making problem and describe the control architecture. We demonstrate how the Policy Search Dynamic Programming algorithm (Bagnell et al. 2003) can be used to learn good policies by decomposing the reinforcement learning problem into a sequence of one-step learning problems. Finally, we give experimental results for both simulation and real-world exploration problems.

2. MAP BUILDING

The mapping problem can be described formally as inference of a map $\mathbf{m}$. The environment is assumed to be stationary, such that $\mathbf{m}$ does not change with time, and that the map consists of a set of features $\{m^0, \ldots, m^{|\mathbf{m}|}\}$, such as grid cells or landmark positions. We wish to infer the map from a sequence of observations, $z_{0:T} = \{z_0, \ldots, z_T\}$, where z_t is the observation at time t. These observations are generally measurements of the features m^i such as range and bearing to the feature in the robot's frame of reference. The subscript $0:T$ denotes a sequence of measurements taken at discrete intervals from time $t = 0$ to some end time $t = T$. Since sensing is a noisy and potentially ambiguous process, each observation z_t is modeled as a stochastic function $p(z_t|\mathbf{x}_t, \mathbf{m})$ of both the map and the vehicle pose at time t, denoted as $\mathbf{x}_t = (x_t, y_t, \theta_t)$, and we refer to this distribution as the observation function or measurement model. A distribution over maps, $p(\mathbf{m}|z_{0:T}, \mathbf{x}_{0:T})$, can be inferred from observations and known robot poses as

$$p(\mathbf{m}|z_{0:T}, \mathbf{x}_{0:T}) = \alpha \prod_{t=0}^{T} p(z_t|\mathbf{m}, \mathbf{x}_t)p(\mathbf{m}), \quad (1)$$

where α is a normalizer. Given the distribution over $\mathbf{m}$, a single map estimate $\hat{\mathbf{m}}$ can be extracted by using a statistic of the distribution, such as the mean or the maximum likelihood estimate.

2.1. Mapping and Localization

It is rarely the case that mapping is carried out in an environment where the robot pose is known. Without knowledge of the robot pose, the joint distribution of the map and robot poses must be represented, a formulation known as the SLAM problem (Smith et al. 1990; Moutarlier and Chatila 1989; Leonard and Durrant-Whyte 1991). If the robot poses are correlated by the control actions u_t taken by the robot using the transition function $p(\mathbf{x}_{t+1}|\mathbf{x}_t, u_t)$, then the joint distribution over the map and robot poses is $p(\mathbf{m}, \mathbf{x}_{0:T}|z_{0:T}, u_{0:T})$

If many features m^i in the map $\mathbf{m}$ are correlated by observations taken from a single pose, then inferring the true posterior can become computationally demanding. A common approximation is to filter the observations, such that the inference is over the map and just the most recent robot pose, $\mathbf{x}_T$, rather than the complete robot trajectory, $\mathbf{x}_{0:T}$. Using some statistical identities, we can compute the joint posterior as

$$\begin{aligned} p(\mathbf{m}, \mathbf{x}_T|z_{0:T}, u_{0:T}) = \quad (2) \\ \alpha p(z_T|\mathbf{x}_T, \mathbf{m}) \int p(\mathbf{x}_T|u_T, \mathbf{x}_{T-1}) \cdot \\ p(\mathbf{m}, \mathbf{x}_{T-1}|z_{0:T-1}, u_{0:T-1}) d(\mathbf{x}_{T-1}). \end{aligned}$$

Equation (2) is known as the Bayes filter, and is a popular form of map inference for its recursive formulation, allowing additional observations to be incorporated into the posterior efficiently. Implementing the Bayes' filter requires committing to a specific distribution of the distribution $p(\mathbf{m}, \mathbf{x}_T)$, and this choice of representation will have consequences for the representation of the transition $p(\mathbf{x}_t|\mathbf{x}_{t-1}, u_t)$ and observation $p(z_t|\mathbf{m}, \mathbf{x}_t)$ functions.

2.2. Extended Kalman Filter SLAM

One of the most popular representations for mapping is the Kalman filter (Kalman 1960), in which the state distribution is assumed to be Gaussian, such that

$$p(\xi) = \mathcal{N}(\mu, \Sigma), \quad (3)$$

where $\xi = (\mathbf{x}, \mathbf{m})$ is the joint state space of the map and robot pose, μ is the mean estimate of the state, and Σ is the covariance. The original Kalman filter formulation of the general tracking problem assumes that the transition and observation

functions are linear functions of the state with Gaussian noise. If the transition and observation functions are non-linear as in the SLAM problem, the extended Kalman filter (EKF) (Smith et al. 1990) approximates the transition and observation functions by linearizing them around the mean state at each time step.

Formally, the EKF models the transition and observation functions as general non-linear functions g and h, such that

$$\xi_t = g(\xi_{t-1}, u_t, w_t) \quad w_t \sim \mathcal{N}(0, W_t), \quad (4)$$

$$\text{and } z_t = h(\xi_t, q_t), \quad q_t \sim \mathcal{N}(0, Q_t), \quad (5)$$

where w_t and q_t are random, unobservable noise variables. In the presence of this unobserved noise, the EKF estimates the state at time T from the estimate at time $T-1$ in two separate steps: a process step based only on the control input u_T leading to an estimate $p(\overline{\xi}_T) = \mathcal{N}(\overline{\mu}_T, \overline{\Sigma}_T)$, and a measurement step to complete the estimate of $p(\xi_T) = p(\mathbf{m}, \mathbf{x}_T)$[1]. The process step is given by

$$\overline{\mu}_T = g(\mu_{T-1}, u_t, 0) \quad (6)$$

$$\overline{\Sigma}_T = G_T \Sigma_{T-1} G_T^T + V_T W_T V_T^T, \quad (7)$$

where G_T is the Jacobian of g with respect to ξ evaluated at μ_{T-1}, and V_T is the Jacobian of g with respect to w. For convenience, we use $R \triangleq VWV^T$. Similarly, the measurement step follows as

$$\mu_T = \overline{\mu_T} + K_T(h(\overline{\mu}_T, 0) - z_T). \quad (8)$$

K_T is the Kalman gain and is given as

$$K_T = \overline{\Sigma}_T H_T^T \left(H_T \overline{\Sigma}_T H_T^T + Q_T^T\right)^{-1}, \quad (9)$$

where H_T is the Jacobian of h with respect to ξ evaluated at $\overline{\mu}_T$. Finally, the covariance follows as

$$\Sigma_T = (I - K_T H_T)\overline{\Sigma}_T. \quad (10)$$

The complete EKF update algorithm is given in Algorithm 1.

The EKF approach to SLAM and mapping is attractive for a number of reasons. Firstly, when the state vector μ is not too large, the inference process is efficient; the most computationally demanding step is the matrix inversion required to calculate the Kalman gain. When the state vector does become large, this matrix inversion can become unwieldy, but there exist a number of approximation algorithms for factoring the map model to ensure constant upper bounds on the map inference process (Thrun et al. 2004; Bosse et al. 2005; Walter et al. 2005) while minimizing the error induced in the approximation. Secondly, when the linearization of the transition and observation functions is a good approximation, the EKF mean has been shown to be an unbiased estimate that minimizes the expected squared error (Newman and Leonard 2003). Finally, by representing the distribution over the map and robot pose parametrically, statistics of the map estimate can be used in planning exploration.

[1] We denote $\overline{\xi}, \overline{\mathbf{m}}, \overline{\mathbf{x}}_t, \overline{\mu}_t$ and $\overline{\Sigma}_t$ as the variables after the control u_t but before the measurement z_t has been integrated. For brevity we will also omit conditioning variables such that $p(\xi_t) = p(\mathbf{m}, \mathbf{x}_T) = p(\mathbf{m}, \mathbf{x}_T | u_{0:T}, z_{0:T})$.

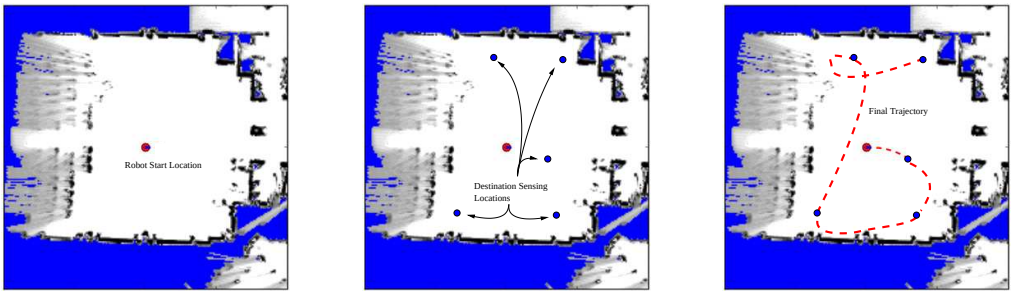
Algorithm 1 The Extended Kalman Filter

Require: Prior mean and covariance μ_{t-1}, Σ_{t-1}, control u_t and observation z_t.

1: Compute process mean,
$$\overline{\mu}_t = g(\mu_{t-1}, u_t, 0)$$
2: Compute process Jacobian G_t.
3: Compute process covariance,
$$\overline{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t.$$
4: Compute measurement Jacobian H_t.
5: Compute Kalman gain,
$$K_t = \overline{\Sigma}_t H_t^T \left(H_t \overline{\Sigma}_t H_t^T + Q_t\right)^{-1}.$$
6: Compute new mean,
$$\mu_t = \overline{\mu_t} + K_t(h(\overline{\mu}_t, 0) - z_t).$$
7: Compute new covariance,
$$\Sigma_T = (I - K_T H_T)\overline{\Sigma}_T.$$
8: **return** (μ_t, Σ_t).

2.3. Map Representations

Throughout the remainder of this paper, we will assume without loss of generality that the map $\mathbf{m}$ is a set of features with specific locations, $m^i \in \mathcal{R}^2$. While our treatment of mapping and exploration will be independent of the map representation, we will rely on an accurate model of the joint posterior, including statistics of the posterior such as the covariance of the map and robot trajectory, in order



(a) Partial map estimate (b) Selected Coverage Constraints (c) Constrained Exploration Trajectory

Fig. 2. The constrained optimization of the exploration process. (a) The current partial map estimate. (b) The sensing constraints chosen by the map coverage process. (c) The resulting trajectory that maximizes map accuracy while satisfying the coverage constraints.

to minimize the expected error. If range sensor data is available to capture the boundary between free space and obstacles, a common approach is to first use a feature-based map to infer the mean estimate of the vehicle trajectory $\mathbf{x}_{0:T}$ and then use equation (1) to build a grid map from the range measurements and the estimated mean poses. In practice, the occupancy grid contains a strong independence assumption between observations of map features (known as the "Naive Bayes" assumption) that is not consistent with the true generative model. Using Naive Bayesian independence assumptions can often allow computationally efficient and accurate inference of the mean of a distribution but the higher order moments are generally biased and over-confident, leading to problems in active learning and exploration (Roy and McCallum 2001).

3. Exploration and Active Map Building

The SLAM problem in the previous sections describes how a robot can build a map: by collecting a series of measurements, inferring a distribution over the map and using the mean of the distribution as an unbiased estimate. If the extended Kalman filter is to compute this distribution then the posterior mean will minimize the error of the distribution and, given enough data, the mean will converge to the true map. Unfortunately, few if any guarantees are given for the rate at which the map converges. The mapping process is highly dependent on the Gaussian assumption, the robot trajectory, the acquired sensor measurements, and the distinctiveness of map features (e.g. to perform data association). The goal of the algorithm in this paper is to choose control actions that lead to sensor data that results in the best map. There are, however, competing concerns in determining the "best map" and how a robot should explore its environment. Sensor measurements are expensive with respect to time, power and certainty, and map inference is only possible for the sensed parts of the environment; if the data only covers part of the environment, then the mapping process will be incomplete and the map $\mathbf{m}$ will not contain all m^i features. Therefore, exploration planning should be designed to minimize the number of sensor measurements required to sense every feature of the map $\mathbf{m}$. Conversely, complete but erroneous maps as in Figures 1(a) and 1(b) are not useful for planning, therefore exploration should be designed to minimize the error of the map.

There are different ways to reconcile these objectives of coverage and accuracy. The most common approach is to optimize the robot trajectory with respect to an overall objective that is a weighted sum of individual terms (Bourgault et al. 2002), but this raises the question of how to choose the individual weights. Our approach is instead to define the coverage problem as choosing the n best sensing locations that maximize the likelihood the complete environment will be observed. If these n sensing locations serve as *constraints* on a trajectory

planner which maximizes the map accuracy subject to these constraints, then we have a principled way to ensure that every map feature is contained in the map estimate $\mathbf{m}$, and that the map itself is accurate. Figure 2 shows how this process can be decomposed and both objectives optimized separately. The trajectory constraints based on coverage can be calculated first, and then the constrained trajectory can be calculated to maximize map accuracy; the following two sections describe how to optimize each separately.

3.1. Map Coverage

Let us first consider the problem of coverage. We define a "visibility" function $\delta(m^i, z_{0:T})$,

$$\delta(m^i, z_{0:T}) = \begin{cases} 1 & \text{if map feature } m^i \text{ appears at least once in the observations } z_{0:T}, \\ 0 & \text{otherwise.} \end{cases} \quad (11)$$

The goal of a coverage planner can then be given as a search for the trajectory that maximizes the number of sensed features,

$$\underset{z_{0:T}}{\operatorname{argmax}} \sum_{i=1}^{|\mathbf{m}|} \delta(m^i, z_{0:T}). \quad (12)$$

We cannot control or predict *a priori* what observations z_t will be received. We can control the robot poses $\mathbf{x}_{0:T}$, but the observations are generated stochastically according to the map and the robot pose, $p(z_{0:T}|\mathbf{x}_{0:T}, \mathbf{m})$. We do not know the map $\mathbf{m}$, so equation (12) must be computed in expectation over the map prior distribution $p(\mathbf{m})$ and observation likelihoods, such that

$$\underset{\mathbf{x}_{0:T}}{\operatorname{argmax}} E_{p(\mathbf{m})} E_{p(z_{0:T}|\mathbf{x}_{0:T},\mathbf{m})} \left[\sum_{i=1}^{|\mathbf{m}|} \delta(m^i, z_{0:T}) \right]. \quad (13)$$

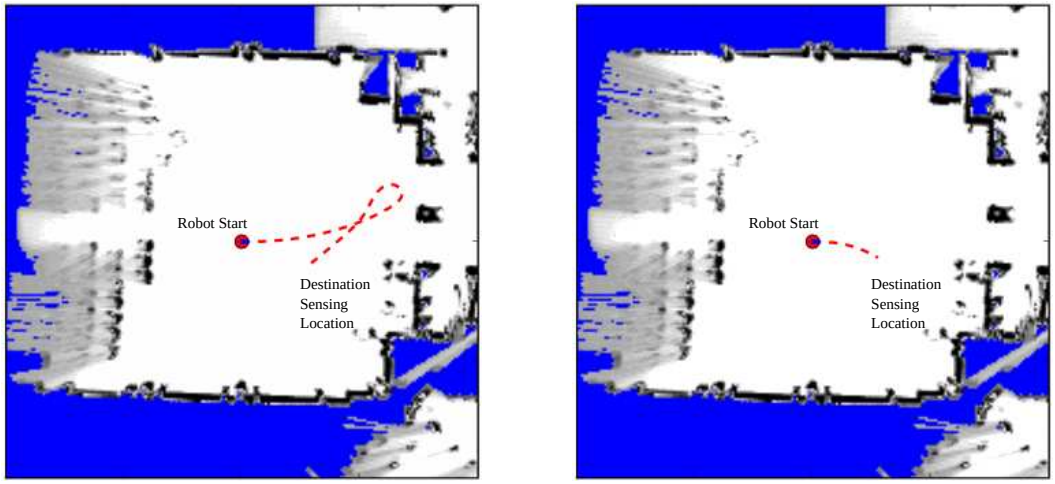
This raises two problems. Firstly, computing the double expectation over both map prior and observations is computationally infeasible. Secondly, the utility of the plan depends strongly on how informative the map prior is. At the beginning of the mapping process, the map prior will be very uncertain, which will cause all trajectories to have approximately the same expected performance, as all maps are equally likely. As more observations are made, the map distribution will begin to converge to the true map, but the mean map estimate $\hat{\mathbf{m}}$ will become an increasingly good approximation to the complete distribution. In either case, we can approximate equation (13) as

$$\underset{\mathbf{x}_{0:T}}{\operatorname{argmax}} E_{p(z_{0:T}|\mathbf{x}_{0:T},\hat{\mathbf{m}})} \left[\sum_{i=1}^{|\mathbf{m}|} \delta(m^i, z_{0:T}) \right]. \quad (14)$$

Therefore, a robot coverage algorithm that chooses a set of destination points to maximize the likelihood of completing the map based on the current estimate, and then follows a tour through these points will have performance comparable to any other sequential decision making algorithm. Solving this optimization problem is outside the scope of this paper, but there are a number of geometric coverage planners in the literature including *art gallery*-type results (Bourque and Dudek 2000; Ganguli et al. 2006). In the remainder of this paper we will assume the availability of such an algorithm for determining sensor locations for coverage.

3.2. Map Accuracy

The coverage exploration optimization does not depend on the robot motion, but only on the robot perception at discrete points in the environment; the optimization is independent of the motion between these discrete sensing locations. However, the trajectory between sensor measurements can have a significant effect on the accuracy of the map. When the robot motion is noisy, substantial uncertainty is introduced into the robot pose; when few or no observations are expected (e.g., because of sensor range limits or environmental sparsity), this uncertainty is propagated throughout the SLAM filter into future measurements, increasing the expected error of the posterior. By avoiding trajectories with large motion noise and trajectories with limited observations of map features, the robot can minimize the error introduced by its own motion. Similarly, 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 controlling the number of measurements or choosing trajectories that maximize the number of observations of specific landmarks, even



(a) Worst Trajectory (b) Best Trajectory

Fig. 3. Example trajectories. (a) An example bad trajectory, where the long looping trajectory induces considerable positional error. (b) The optimal trajectory between the start and goal location. The short motion and large radius of curvature minimizes the overall error.

when traveling through already-mapped areas of the environment.

In Figure 3, two trajectories are shown. The robot does not have access to the complete map in either case, although the map is drawn in for reference. In Figure 3(a) the robot trajectory is long and contains a tight loop. This would clearly be a very suboptimal way of arriving at the destination point. In contrast, Figure 3(b) shows a much more reasonable trajectory both in terms of length and in terms of the kind of path.

Assuming a standard error function such as sum of squared errors, an exploration planner that is to generate the most accurate map must optimize its trajectory $\mathbf{x}_{0:T}$ as follows

$$\underset{\mathbf{x}_0:T}{\operatorname{argmin}} E_{p(\mathbf{m}|\mathbf{x}_{0:T})}||\mathbf{m} - \hat{\mathbf{m}}||^2. \qquad (15)$$

where again $\hat{\mathbf{m}}$ is the mean of the map marginal distribution. In the SLAM problem, we do not have direct control of the robot pose but can only issue controls u_t, so we change the objective function to

$$\underset{u_{0:T}}{\operatorname{argmin}} E_{p(\mathbf{m}|u_{0:T})}||\mathbf{m} - \hat{\mathbf{m}}||^2, \qquad (16)$$

implicitly marginalizing over states and observations. The constraints of the sensing locations are not present in the objective of equation (16); they are constraints on the solution that the optimizer must satisfy. Just as in equation (13), this expectation of equation (16) is computationally intractable without a fairly accurate map prior. Furthermore, since the expectation is taken with respect to the map at the *end* of the trajectory, the same approximation as in equation (14) cannot be used. Without the ability to optimize the trajectory exactly or find a good approximation, we turn to reinforcement learning in the next section to find trajectories to minimize equation (16) subject to the constraints of equation (14).

4. Reinforcement Learning for Exploration

Solving for the policy in equation (16) requires a sequential decision-making algorithm, where the choice of action u_t depends on the current posterior distribution $p(u_t, \mathbf{m})$. The most general framework for choosing actions based on the current distribution is the partially observable Markov decision process (POMDP) (Sondik 1971). A POMDP is described by the following:

- a set of states S,

- a set of actions A,
- a set of observations Z,
- a transition function $T : S \times A \times S \rightarrow \mathcal{R}$ such that $T(s^i, a^j, s^k) = p(s^i|a^j, s^k)$,
- an emission function $O : S \times A \times Z \rightarrow \mathcal{R}$ such that $O(s^i, a^j, z^k) = p(z^k|a^j, s^i)$,
- a reward function $r : S \times A \times S \rightarrow \mathcal{R}$,
- an initial state distribution $p_0(S)$
- a discount factor $\gamma \in [0, 1]$.

The state at each time t is assumed to be unobservable. Assuming a known action a, a received observation z and the corresponding transition and action functions, the Bayes' filter of equation (2) can be used to estimate the distribution $p(s)$. The standard optimal policy for a POMDP model is one that chooses an action a based on the current distribution $p(s)$ to maximize the expected discounted reward over the life of the agent. Given appropriate choices of the state, action and observation sets and transition, observation and reward functions, the exploration problem can be modeled as a POMDP.

The state space trivially seem to be the current robot pose $\mathbf{x}$ and the map $\mathbf{m}$. Almost all algorithms for solving POMDPs optimally require a discrete state space, but rather than solving for the POMDP policy, the use of an appropriate reinforcement learning algorithm will allow us to maintain a continuous representation of the state. However, recall that the trajectory must satisfy a set of constraints in terms of visiting the sensing locations selected by the coverage planner. We therefore augment the state with the set of remaining locations $\{\mathbf{L}_1, \ldots, \mathbf{L}_n\}$ that are to be visited, such that

$$S = \mathbf{X} \times \mathbf{M} \times \{\mathbf{L}_i\} \tag{17}$$

where the last term is a set of sensing locations that the policy must visit, and n will decrease as the trajectory visits each location in turn. By putting the active constraints directly into the state space, there is a danger of computational intractability, but we will again rely on the reinforcement learning algorithm to avoid this problem. It should be noted that this is a hybrid state space, where the first two components (robot pose, $\mathbf{x}$, and map $\mathbf{m}$) are unobserved but the sensing locations $\{\mathbf{L}_i\}$ are chosen by the exploration planner and are therefore fully observable.

The observations are exactly the observations z described in Section 2. The observation emission function is given by $p(z|s) = p(z|\mathbf{x}(s), \mathbf{m}(s))$ where $\mathbf{x}(s)$ is the robot pose component of state s, $\mathbf{m}(s)$ is the map component of state s and $p(z|\mathbf{x}(s), \mathbf{m}(s))$ is the observation function used for mapping in Section 2.

4.1. The Action Space

The action space may initially appear to be the parameterized control space of the robot, which for most ground robots is a bearing and velocity, $u = (\theta, v)$. This action has the unfortunate effect of requiring extremely long plans in terms of the number of actions required; solving for a POMDP policy has complexity that is roughly double exponential in the horizon length (a loose bound on the complexity is $O(|A|^{|Z|^t})$ for discrete and finite state, action and observation spaces), and learning a policy will also scale in complexity with the horizon length. In order to have any hope of finding good exploration policies, choosing a different policy class (and action parameterization) that can express the optimal policy more compactly is essential.

In choosing a more appropriate policy class, we can take advantage of structure in our problem. Firstly, we can use the policy class to impose a smoothness on the trajectories. Instead of requiring the exploration planner to select fresh control parameters at each time t, we can induce a bias in the controls by representing the action as a smooth parametric curve. Secondly, if the policy class is a set of trajectories, we can also use this policy class to enforce the constraints imposed by the coverage planner, which simplifies the optimization process. As a result, we define the action for state s to be a parameterized trajectory from the robot's current pose (position and orientation) to the next sensing location in the state.

The simplest polynomial that both interpolates the robot pose and the next sensing location and allows us to constrain the start orientation is a cubic polynomial, that is, the curve parameterized as

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

with $t \in [0, 1]$. The curve is fixed such that the derivative at the start matches the orientation of the robot at its current position, such that

$$atan2\left(\frac{\partial y}{\partial t}\Big|_{t=0}, \frac{\partial x}{\partial t}\Big|_{t=0}\right) = \theta.$$

Using this parameterization 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, $(x, y, \theta)_{t=0}$ is the start pose of the robot and $(x, y)_{t=1}$ is the destination position (x_g, y_g). 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, to give a continuous action space in $\mathcal{R}^3$. Figure 4 shows a set of sample action parameterizations, with a destination sensing point that is 3 meters in front of the robot.

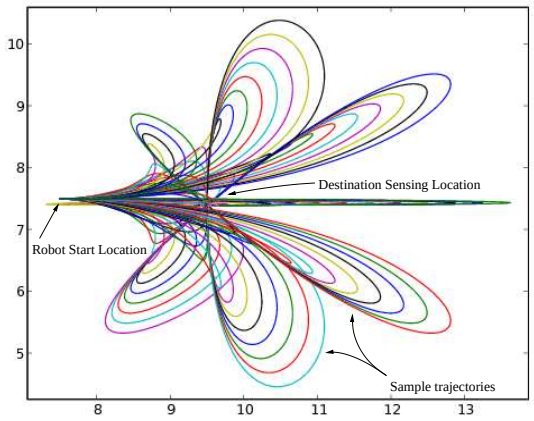


Fig. 4. A sample set of possible actions for a destination sensing point that is 3 meters in front of the robot. Some of the actions are clearly sub-optimal for this start and goal, but may be useful for achieving a particular orientation at the destination. The reinforcement learning problem is to identify which parameterization should be chosen given the current robot state, map and sensing points.

There are three potential issues in this policy class that could lead to sub-optimality in the performance. Firstly, the set of motion strategies that can be expressed by the exploration planner has been considerably reduced, and the true optimal exploration policy may no longer be feasible with this action parameterization. However, this bias allows us to dramatically increase the computational tractability of the problem; in preliminary experiments using richer policy classes, we experienced difficulties in scalability and learnability, and could not demonstrate satisfactory convergence for learning exploration policies outside of toy examples. The choice of policy class is one of the key enablers of our optimization, and we will see experimental results that suggest that this policy class results in good (if not guaranteed optimal) exploration performance.

A second issue is that the curves are not constrained to a fixed length and will take varying amounts of time to execute. The assumption of a standard discounted POMDP is that the actions are all of some fixed length. If this assumption is violated for problems with a discount factor $\gamma < 1$, then the planner may not be able to correctly compute the expected discounted reward. A complete plan comprised of fewer but longer actions will appear to be less valuable compared to a plan of the same overall performance but comprised of more, shorter actions. The correct planning formalism would be the semi-Markov decision process, which adds an explicit representation of the duration of each action. Experimentally, we found that the variability in the length of the actions was not sufficient to justify the additional computational overhead of a semi-Markov model.

It should be noted that the actions are kinematic trajectories and do not directly provide motor control commands. To convert each action into executable motor controls $u_{0:T}$, we use pure-pursuit control (Ollero and Heredia 1995) on the current robot position estimate and a point on the curve a distance d in front of the robot. This pure-pursuit controller also gives rise to the transition function corresponding to the action a. Each action a gives rise to a distribution of possible control sequences required to follow the curve a. The probability of arriving at a particular state s_i (for s_i defined by Equation 17) is

$$p(\mathbf{x}(s_i)|\mathbf{x}(s_j), a) = p(\mathbf{x}(s_i)|\mathbf{x}(s_j), u_{0:T}). \quad (18)$$

This probability is the result of a series of EKF

updates, and can not easily be evaluated in closed form, but can be approximated using Monte Carlo integration. The map component $\mathbf{m}(s)$ remains constant. Finally, the sensing location sequence component of the state changes deterministically in that once a sensing location is visited at the end of executing action a, that sensing location is deterministically removed from the coverage plan.

4.2. The Reward Function

Given access to training data with ground truth information of the map ($\hat{\mathbf{m}}$), the reward function is the squared error of the map as in equation (15). For all actions, the reward function is

$$r(\cdot, a, \cdot) = ||\mathbf{m}(s) - \hat{\mathbf{m}}|| \tag{19}$$

The squared error reward function is our fundamental quantity of interest. This reward function is not a function of the specific SLAM algorithm, and allows the controller to predict the effects of unmodeled SLAM filter failures. Over-confidence, bad data-associations, and filter divergence account for a number of problems that may occur and are not modeled explicitly by an extended Kalman filter mapper; these nevertheless will affect the squared error.

In circumstances where the squared error cannot be computed due to lack of ground-truth data, an alternate reward function based on the uncertainty of the distribution can be considered. There is an abundance of related work using information gain for choosing actions, among which the most closely related comes from (Makarenko et al. 2002; Stachniss 2006; Feder et al. 1999; Sim 2006). Information-theoretic reward functions have also been used to choose the next best measurement in sensor networks. If the estimator is unbiased, then as the entropy of the distribution converges to 0, the mean of the estimator will converge to the true distribution. The literature on optimal experimentation (Fedorov 1972) defines two principal information-theoretic objective functions in terms of information:

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

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

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

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

For information gain, $\Sigma_{V|A}$ is the covariance after a set of observations A have been made of variables V and Σ_V is the covariance before the they were made.

There is some intuition that underlies these metrics. The determinant of a square, symmetric matrix is the product of the eigenvalues, and so the determinant of the covariance roughly corresponds to the volume of the uncertainty ellipse at a contour of constant likelihood. The determinant is directly proportional to the entropy of a Gaussian. Reducing one of the dimensions of uncertainty to zero corresponds to driving the determinant of the covariance matrix to zero. Similarly, the trace has an interpretation as the sum of the eigenvalues of the covariance. The trace therefore roughly corresponds to the circumference of the uncertainty ellipse. We will compare the policies for these reward functions in the next section.

5. Policy Search Dynamic Programming

Given the model parameters described in the previous section, it remains to find the optimal policy mapping distributions over states $p(s)$ to actions a. The POMDP model of the exploration problem described above includes large and continuous state, action and observation spaces. There are no known algorithms for finding the exact optimal policy for continuous POMDPs, only approximation algorithms (Porta et al. 2006) and these algorithms are extremely unlikely to scale, either by approximating the value function or by approximating the continuous parameters with discretizations. Finally the model is sufficiently complex that even generating an explicit representation of the model parameters may be computationally intractable.

Although an exact computational solution is infeasible, we do have access to high-quality simulators and real-world data that can be used to characterize good solutions. By learning good policies, we avoid the need to represent the model and solve it.

Algorithm 2 Policy Search by Dynamic Programming

Require: Simulator with state sampling function P

1: $\xi = \left\{p^{(1)}(s), \ldots, p^{(m_1)}(s) | p(s) \sim P\right\}$

2: Where $v_t^{a_j, \pi_{t+1}, \ldots, \pi_N}(s)$ is a trace through the MDP sampled m_2 times using the previously computed policies:

$$v_t^{A, \pi_{t+1}, \ldots, \pi_N}(\xi) = \begin{pmatrix} v_t^{a_1, \pi_t+1, \ldots \pi_N}(p^{(1)}(s) & \ldots & v_t^{a_{|A|}, \pi_t+1, \ldots \pi_N}(p^{(1)}(s) \\ \vdots & \vdots & \vdots \\ v_t^{a_1, \pi_t+1, \ldots \pi_N}(p^{(m_1)}(s)) & \ldots & v_t^{a_{|A|}, \pi_t+1, \ldots \pi_N}(p^{(m_1)}(s)) \end{pmatrix}$$

3: $\hat{\pi}_t(\xi) = \text{argmax}_{a \in A} \left[v_t^{A, \pi_{t+1}, \ldots, \pi_N}(\xi) \right]$, where argmax is taken with respect to the rows of the matrix.

4: Train a classifier with input ξ and output $\pi_t^{'}(\xi)$ and return the classifier as the policy:

$$\pi_t(p(s)) : P(s) \rightarrow A$$

5: $\Pi = \{\pi_t \ldots \pi_N\}$; $t = t - 1$; goto step 1 until convergence.

We instead use reinforcement learning to find good policies from data.

The decision to use reinforcement learning does bring with it the question of which algorithm to use in learning. There are a number of different learning algorithms, with different representational and learning power. The process of gathering data during exploration learning can be expensive, and the input data (the state distributions) are high-dimensional. Policy Search Dynamic Programming (Bagnell et al. 2003) is a form of reinforcement learning that decomposes the learning process into a series of one-step learning algorithms, essentially turning the learning of sequential decision making into a form of dynamic programming. The Policy Search by Dyanamic Programming (PSDP) algorithm can be seen in Algorithm 2.

The algorithm operates by first learning a one-step policy at time T (e.g., the end time) by generating a sample distribution $p^{(i)}(s)$ and action a and propagating each sampled distribution $p^{(i)}(s)$ forward. Each distribution-action pair is labeled with immediate reward. The best distribution-action pair is kept as a training instance for a supervised learning problem at time t. The result is a classifier that provides an optimal action a for any distribution $p(s)$. A new one-step policy for the previous time step $T-1$ is then learned by sampling distributions and actions as before. Using the policy previously obtained at time T, a two-step value is obtained for each distribution and action at $T-1$: the value associated with each distribution and action is accumulated from the immediate reward at time $T-1$ *and* the reward received by running the learned policy for time T, and a new policy is learned for time $T-1$. The learner then iterates at each time t through to time T using the policies π_i for $i \in \{t \ldots T\}$. After sampling all of the actions for distribution $p(s)$ at time t, the best action a' is selected.

5.1. Support Vector Machine Policy Learning

The primary advantage to PSDP for our purposes is that each one-step policy can be found using any learning algorithm. This allows us to take advantage of machine learning techniques that are effective in generalizing from high-dimensional data. There are a number of classifiers that could be chosen for use in PSDP such as neural networks, Gaussian processes, and support vector machines (SVMs). We have chosen to leverage a classifier known to generalize well in high-dimensional spaces, has few parameters to tune and is known to perform well on a variety of problems: a Support Vector Machine (Burges 1998; Duan and Keerthi 2005).

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 has a linear decision boundary. However, by using the "kernel trick", it is possible to extend support vector machines to non-linear decision boundaries (Haykin 1994). The optimization performed to find the best decision boundary can be formulated as a quadratic program that is relaxed to penalize errors. The parameters left free include a term to account for the number of errors that will be allowed as well as the kernel parameters.

To learn a policy, we take training data consisting of labeled distribution-action pairs and learn to predict the best action for each distribution. Although we can maintain a continuous input space, an SVM requires a discrete action space. The basic support vector machine performs binary classification and must be extended to the n-ary classification problem of choosing actions. A set of classifiers is typically trained in a *one-versus-all* or *one-versus-one* manner (Duan and Keerthi 2005; Hsu and Lin 2001). In the one-versus-all framework, each binary classifier is trained using the data from one class against the data from all of the other classes, generating n binary classifiers for each of the n classes. At prediction time, each of the n binary classifiers return either a class or *other*, referring to the fact that it could be any of the other classes. When there are multiple available classes, the one with the highest value is selected as the correct class. In the one-versus-one framework, each binary classifier is trained using data from only two classes, generating $O(n^2)$ classifiers. At prediction time, each pairwise binary classifier returns the most likely of the two classes and casts a vote for this class. The class with the most votes wins the n-ary classification and is taken to be the true class. In (Hsu and Lin 2001), it was shown that the one-versus-one framework is most appropriate for n-ary classification problems such as ours. In our optimization, we will also want to rule out trajectories that collide with obstacles. We simply ignore votes for infeasible trajectories to get the next optimal trajectory while avoiding the need to retrain for particular environmental constraints, allowing the learner to generate policies that generalize across environments.

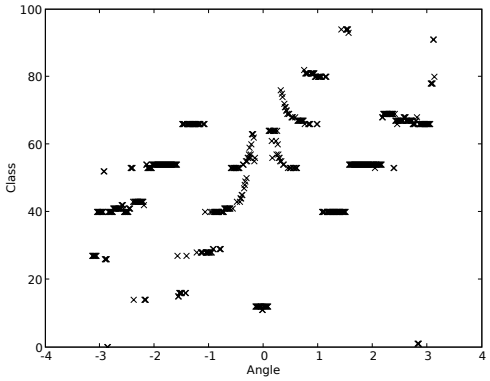
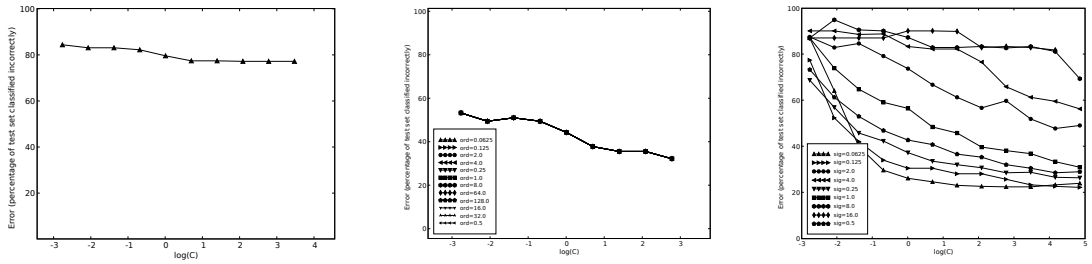


Fig. 5. A slice through the training data along the destination angle, for destinations at a fixed distance. The x axis is angle to destination training instances, for training instances with the next sensing point at a fixed distances from the start robot pose. The y axis is the training label of each instance (i.e., the optimal action). It is impossible to draw a definite conclusion, but the data does not appear linearly separable.

In order to ensure that the SVM can learn well, an appropriate kernel must be chosen. Figure 5 shows a slice through a training data set along the destination bearing direction at a fixed distance to the next destination sensing point. The training data has some clear structure, but does not appear linearly separable, suggesting the need for a non-linear kernel.

Cross-validation was performed with the learned classifier on the test set to determine how well the trajectory exploration function was learned for a linear kernel, a polynomial kernel and a Gaussian kernel. A parameter search over misclassification penalty C and covariance width σ for $C, \sigma \in \{2^i | [-5 \leq i \leq 7] \subset Z\}$ was additionally performed to find the best values. A graph of the parameter search is shown in Figure 6(a)-(c). The y-axis is the classification error rate on the test dataset for different values of C and the x-axis are values of σ. The best values over a range of datasets were $C = 4$, $\sigma = 0.125$.

We can visualize the learned trajectories to the destination to confirm that the data matches our expectations. Figure 7(a) shows a set of training



(a) Linear Kernel (b) Polynomial Kernel (c) Gaussian Kernel

Fig. 6. In these graphs we can see the effect of search for good parameters. (a) Search for good linear kernel, (b) a polynomial kernel, and (c) a Gaussian kernel. Graph (b) contains data for multiple polynomial kernels, but all appeared to have identical performance. The graphs confirm the suggestion of Figure 5, that the training data is not linearly separable in that the fastest graph to converge is the Gaussian kernel. This is the kernel that was used in this paper.

data consisting of a fixed robot start pose and different destination sensing locations sampled around a 3 meter radius circle centered on the robot. The training label on each sample was found by simulating each trajectory and retaining the maximum reward action, a process that is subject to noise. Therefore, some of the training examples appear to have incorrect labels. Figure 7(b) shows the policy learned from this data. The learner has generalized enough to avoid many of the spurious noisy labels, but classification error does persist. The classification error in general was 20% in 100 class problems.

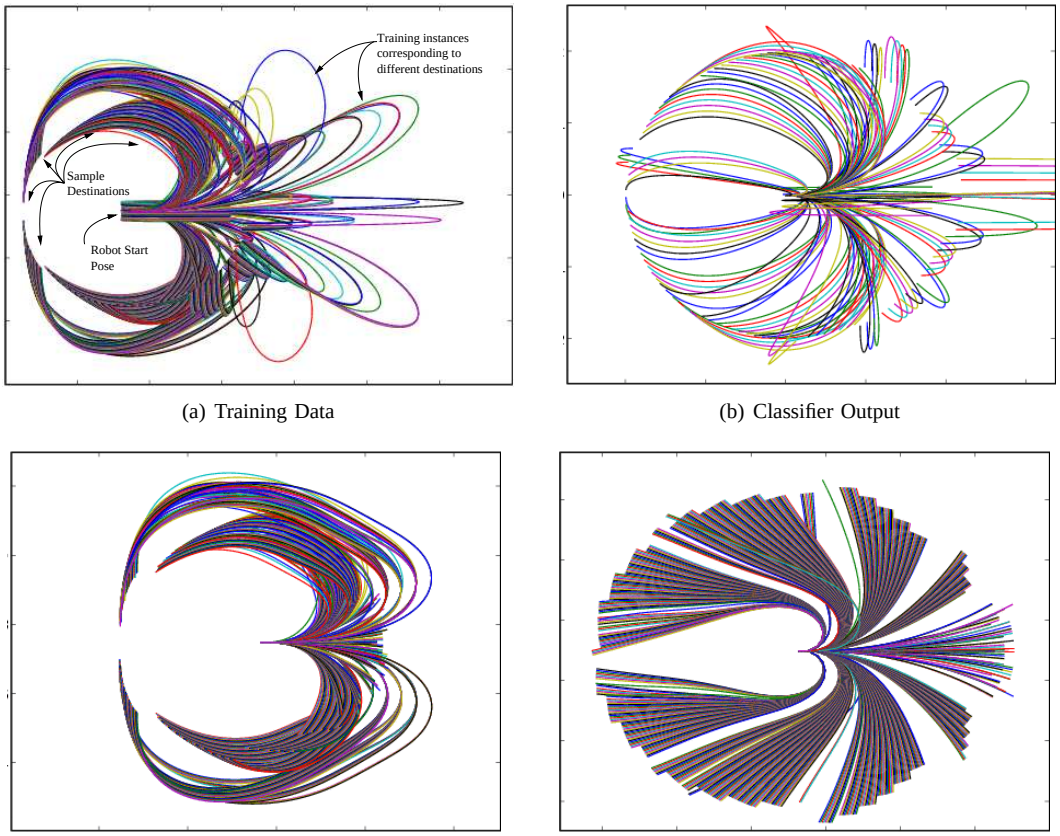
As a comparison, we examined the learner on the reward functions $trace(\Sigma)$ and the entropy of the distribution, $ent(\Sigma)$, which would have the effect of optimizing the policy for different norms on the *predicted* filter covariance, rather than the experienced filter error. The trace controller leads to trajectories that were more reflective of the desired motion with much lower curvature. The controller trained on $ent(\Sigma)$ reward led to a policy that was much closer to a shortest-path controller, suggesting that the $ent(\Sigma)$ reward function is not a good substitute for squared error.

6. Experimental Results and Analysis

Having demonstrated that the classifier was capable of learning the target concept, we trained a control policy and evaluated the performance in both simulated and real experiments. In both cases, the resulting map was significantly improved with the learned trajectories.

6.1. Policy Training

Collecting training data consisted of four phases. Firstly, a model of the robot motion and sensing was built for generating simulated training data. The simulator is part of the CARMEN robot control package, and assumes a quasi-holonomic robot with three degrees of freedom, such that $\mathbf{x} = (x, y, \theta)$. The robot was also been equipped with a 180° field-of-view laser range sensor. The motion model followed (Eliazar and Parr 2004), such that the robot control was in terms of a forward translation and a rotation. Each control consisted of a rotation $d\theta_1$, a translation dr along the direction θ, and a final rotation $d\theta_2$. The motion error had a translation (down-range), rotation and slip (cross-range) component, where slip corresponds to a translation in the direction perpendicular to the expected direction of motion. The specific parameters of the motion model were generated by collecting example controls and sensor data from the robot. The Expectation-Maximization model learning algorithm of (Eliazar and Parr 2004) was used to fit the motion model parameters to the data. The complete equations of motion and the fitted model parameters are given in Appendix A. The sensing model was a laser range finder, with a fea-



(a) Training Data

(b) Classifier Output

(c) Learned Policy Using $trace(\Sigma)$

(d) Learned Policy Using $ent(\Sigma)$

Fig. 7. (a) Example training data. The robot start pose is fixed, and plotted are the optimal actions for different destinations sampled on the circumference of a circle 3 meters in radius centered at the robot pose. (b) Example learned labels for a test set. The robot start pose is again fixed and the samples are again drawn from a circle of radius 3 meters around the robot. (c) and (d) Outcomes of the controller trained on $trace(\Sigma)$ and $ent(\Sigma)$.

ture extraction algorithm that follows the algorithm in (Walter and Leonard 2004).

In order to train each one-step policy, 1,000 pairs of start pose and destination sensing locations $(\mathbf{x}, \mathbf{L}_i)$ were sampled randomly in a simulated map. These poses were sampled from an empty map containing only pre-constructed landmarks. For each action $a \in A$, the robot was placed at the start pose, the action was executed, and an EKF was updated. The action space consisted of 100 different parameterizations of a cubic spline interpolating $\mathbf{x}$ and $\mathbf{L}_i$ and the robot followed this spline using a pure-pursuit controller. The reward (trace or entropy) was computed and the trajectory was labeled with its reward. The best instance of $(\mathbf{x}, \mathbf{L}_i)$ was labeled and returned to be included in the training data.

Third, a support vector machine classifier was trained using the location of the destination(s) as the state and an integer action number as the class. A Gaussian kernel was used as discussed in Section 5.1. The kernel parameters used to train this classifier were $C = 4$ and $\sigma = 0.125$.

Finally, the support vector machine was loaded at runtime and novel destinations given to the robot. In real-time the classifier classified the current state

to generate a trajectory, which the robot followed from its current pose to the destination.

6.2. Artificial Simulated Environment

To understand the significance of the difference between the learned controller and a conventional Shortest-Path coverage controller touring the destination sensing locations, a set of simulated experiments were analyzed. The first of these was in an artificial world that was created to be difficult to map. In particular, when using a scan-matching based approach, matching the right spokes to one another can be difficult since many different alignments may look the same.

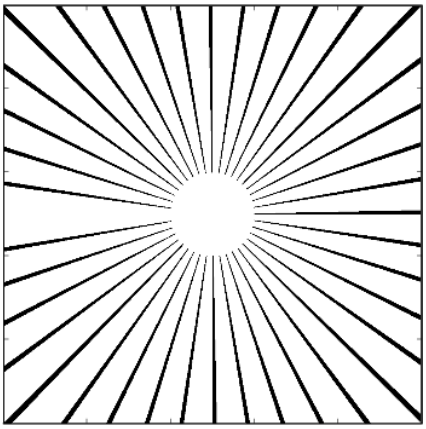


Fig. 8. An artificial simulated map used for initial evaluation in simulation. The white pixels are free space and the black pixels are obstacles.

Figure 8 shows the artificial test map that was used for an initial evaluation of the learned exploration policy. The robot was placed initially in the center of the (unknown) map and given a fixed set of destination sensing locations. The free space of the map is a star polygon with a kernel in the center of the map, which meant that a coverage trajectory was a short loop around the inner circle. Figure 9(a) shows a Shortest-Path trajectory computed using a standard shortest-path motion planner (Barraquand et al. 1992), and Figure 9(d) shows the trajectory of the Learned controller.

Figure 9(b) and Figure 9(e) show maps created by using the recorded (simulated) odometry to directly compute the environmental model from the (simulated) laser range data. In both cases the odometry is noisy, and the maps contain substantial errors and misalignments. However, using a robust mapping algorithm to correct the maps, the trajectory of the Learned controller led to a substantially better map compared to the Shortest-Path controller.

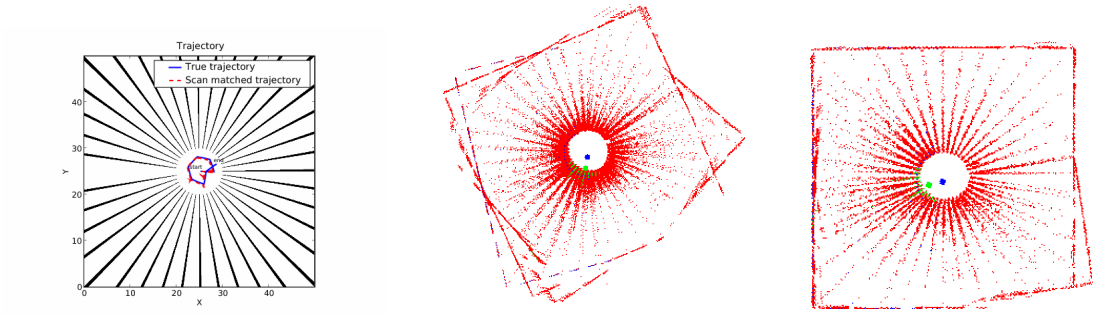
For the Shortest-Path controller, we can clearly see a number of mapping errors. Referring to Figure 8, we can see an opening on one end of the map. This opening is not present in the map from the Shortest-Path controller. Further, a number of the passageways are blocked off, whereas in the map acquired by the Learned controller, this is not the case. Finally, the environment was square; examining the bottom right of the Shortest-Path map, we can see that some of the scans have been misaligned with respect to the square.

Since we performed these experiments in simulation, the true position of the robot was available. After recovering the estimated pose of the robot over the entire trajectory using a SLAM algorithm, the distance of the estimated trajectory from the true trajectory was computed and this error has been plotted in Figures 10(a) and 10(b). The robot pose error has been substantially reduced by the Learned controller, and moreover, the error in the Shortest-Path controller appears to be growing. In contrast, in repeated trials, the error resulting from the Learned controller remained boundedly small in repeated trials.

Overall, in this environment, we can conclude that the Learned controller provided sensor data that was much easier to integrate at a much higher quality than the Shortest-Path controller. Map errors were reduced and the true position of the robot was able to be estimated much more accurately. The use of a robust mapping algorithm for these experiments and the repeatability of the mapping errors leads us to believe that these results will generalize across mapping algorithms.

6.3. Realistic Simulated Environments

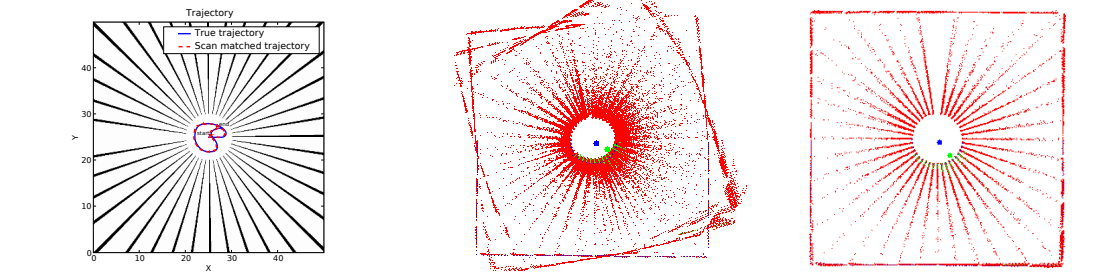
Following these results, further tests were run in simulations of real-world environments. We evaluated the performance in exploring and mapping eight different environmental models using maps



(a) True/Estimated trajectories (Shortest-Path)

(b) Raw Measurements (Shortest-Path)

(c) Corrected Measurements (Shortest-Path)



(d) True/Estimated trajectories (Learned)

(e) Raw Measurements (Learned)

(f) Corrected Measurements (Learned)

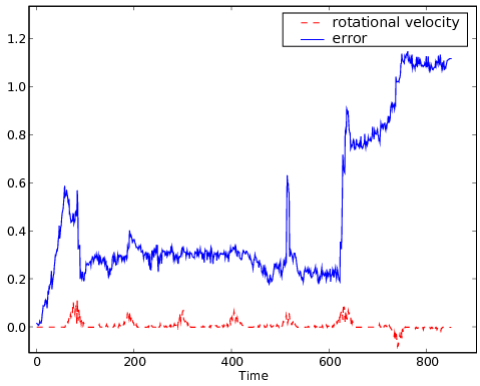
Fig. 9. Comparison of Shortest-Path exploration controller vs. Learned controller. (a) Exploration trajectory using Shortest-Path controller. (b) Raw data plotted using odometry estimates for Shortest-Path trajectory. The red dots are laser endpoints calculated from the robot pose based on dead reckoning (odometry) and the laser range data. (c) Final map inferred by a SLAM algorithm from data collected by Shortest-Path controller. (d) Exploration trajectory using Learned controller. (e) Raw data plotted using odometry estimates for Shortest-Path trajectory. (f) Final map inferred by SLAM algorithm from data collected by Learned controller.

available for the Radish data set (Howard and Roy 2003). Figures 11(a) and 11(c) show example trajectories. Figures 11(d) and 11(b) show that the Shortest-Path controller often has large deviations from the true position, but the learned controller matches the true trajectory very closely. The error in the Learned controller was only briefly above $.1m$, where the error in the Shortest-Path controller grew to $.3m$. When the robot revisited these locations to remap the area, the errors of $.3m$ could have been catastrophic.

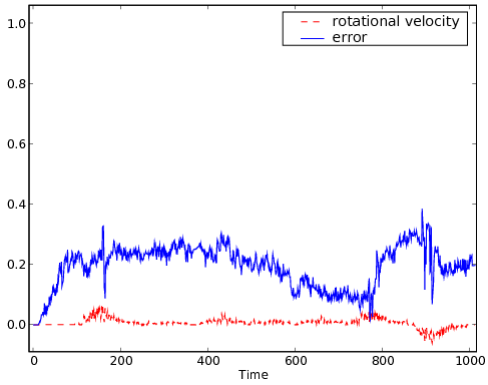
Finally, the average error between the estimated and ground truth robot trajectories in each map was computed. These results are shown in Figure 12. The difference between controllers across maps is variable; for maps 1, 3, 4, 5 and 7, the Learned controller provided a significant improvement. Note that the error bars are plotted at 1,000 standard errors[2] in order to be visible, demonstrating substantial statistical significance to the differences in the mean values.

In map 0, a comparatively short trajectory was followed, so it is reasonable that this was due to the uncertainty in the mapping process. In map 6, the point turn controller performed better largely because the environment was very tight. Further, even though the average map error was reduced by the Shortest-Path controller, the mapping process diverged at the end, and had the data collection not been terminated, the error induced by the Learned

[2]Standard error is defined as $\sigma/\sqrt{N}$.



(a) Error in estimated position (Shortest-Path)



(b) Error in estimated position (Learned)

Fig. 10. (a) The squared error in robot position between the estimated robot position and the ground truth position for the trajectory of the Shortest-Path controller. (b) The squared error in robot position between the estimated robot position and the ground truth position for the trajectory of the Learned controller.

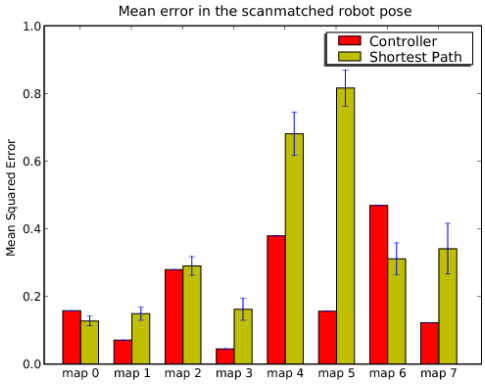


Fig. 12. The average mean squared error between the estimated and true robot pose over a variety of trajectories. The error bars are at 1000 standard errors, which demonstrates that there is substantial statistical significance between the control methods.

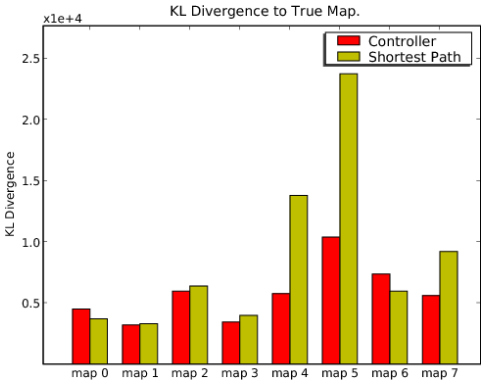
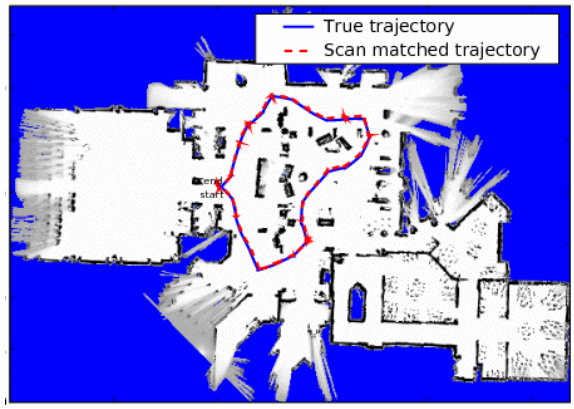


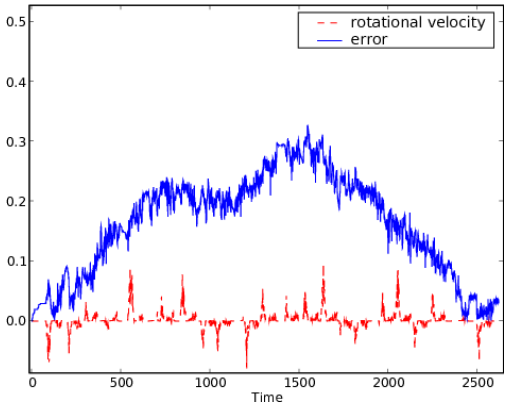
Fig. 13. The KL divergence between the occupancy grid computed from the corrected scans and the ground truth occupancy grid.

controller appeared to be improving compared to the Shortest-Path controller. In conclusion, in five of the eight trials, the Learned controller was considerably better than the Shortest-Path controller in terms of squared error and in others they were equivalent.

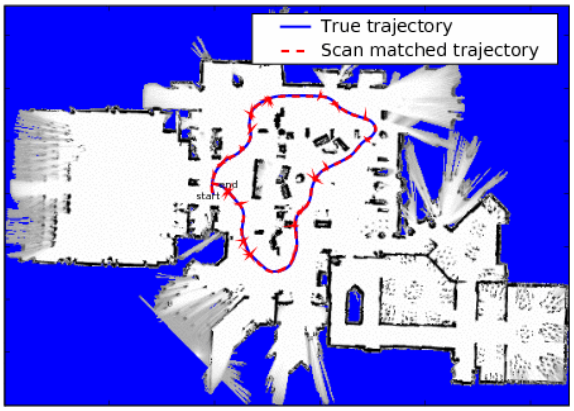
As a final analysis, Figure 12 plots an estimate of the error in the occupancy grid maps created by the mapper. The difference between the estimated and true robot pose shown in Figure 12 was the most reliable estimate of the accuracy of the mapping process, but as an additional verification, we compared the Kullback-Leibler divergence between the occupancy grid computed by the SLAM algorithm and the ground truth occupancy grid computed from the true position of the robot in the simulation (Cover and Thomas 1991). Computing the KL divergence required searching for the minimum error alignment between occupancy grids, as the mapping process can result in a rigid transformation between the inferred map and the



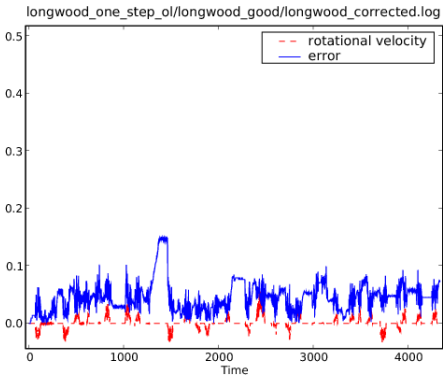
(a) Trajectory (Shortest-Path)



(b) Pose Error (Shortest-Path)



(c) Trajectory (Learned)



(d) Pose Error (Learned)

Fig. 11. Learned trajectories for a simulated experiment using the Longwood map from the Radish data set. (a) The trajectory of the Shortest-Path controller. (b) The robot pose error for the Shortest-Path controller. (c) The trajectory of the Learned controller. (d) The robot pose error for the Learned controller.

corrected map, and this alignment process is subject to local minima problems. Additionally, neither the mapping process nor the exploration process is designed explicitly to minimize this error measure. However, the error in the maps as measured by the KL divergence (Figure 13) was consistent with the analysis of the error in robot pose (Figure 12), which confirmed the performance of the Learned controller.

7. Real-World Exploration

As a final validation, we evaluated the learned controller in a real exploration task. The learned controller was trained using simulated data as before, but the simulator model was learned itself from real data. There are numerous instances of reinforcement learning algorithms transferring well from simulation to real-world experiments when the simulator is a high-fidelity physical model of the world (Ng. et al. 2004). As a result, we had high confidence that our learned policy would perform well in real exploration tasks.

Fig. 14. The robot used both for the simulator model and to test the learned controller.

The robot used for the experiments is shown in Figure 14. It was equipped with a single SICK laser range finder on the front and odometry encoders in each wheel. This robot was particularly interesting because its construction and differential drive mechanism generate noisy odometry.

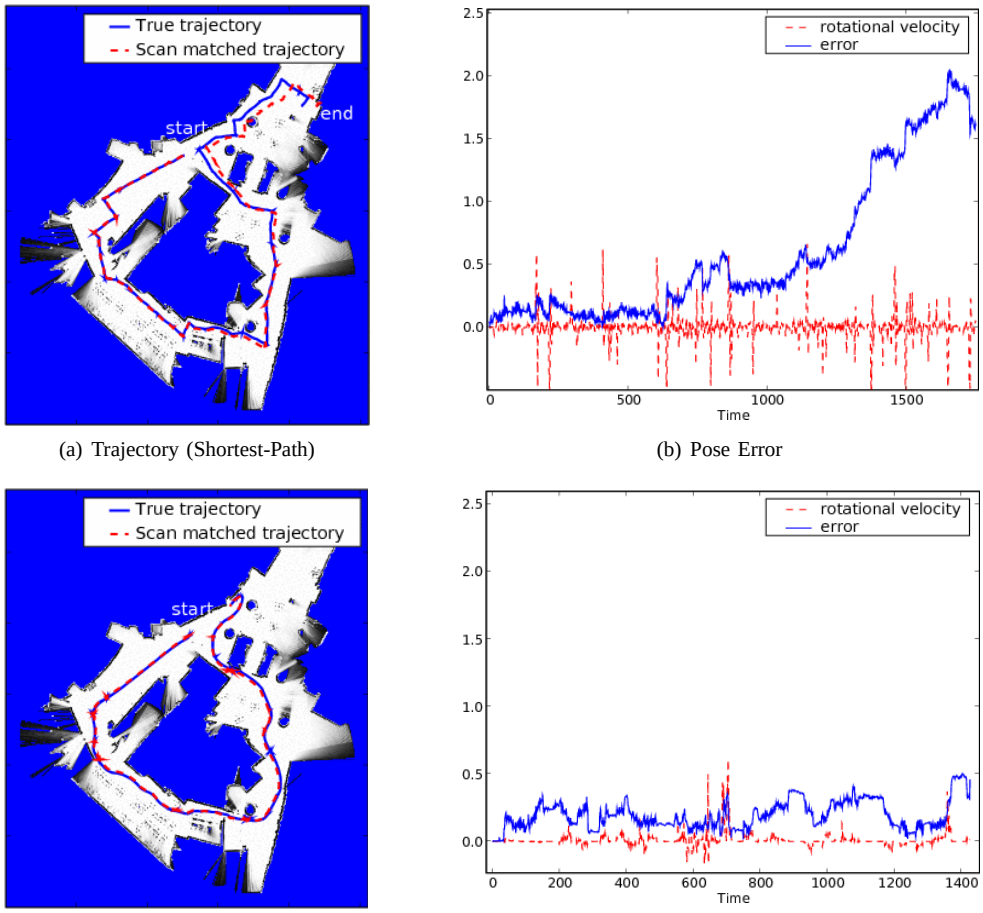
The experiment was performed on the first floor of the Stata Center at the Massachusetts Institute of Technology. No completely accurate "ground truth" map of a real world environment exists due to non-permanent objects, and architectural drawings for the Stata Center are also inaccurate due to permanent changes to the environment structure. Thus, before running experiments, a map of the area was pre-generated by hand, and a process running Monte Carlo Localization onboard provided an estimate of ground truth. During the experiment, the robot chose actions in real-time and drove around a large loop that had been identified during simulation as difficult to map.

Figures 15(a) and 15(c) show the estimated trajectories for the Shortest-Path and Learned controllers. It is clear that the Learned controller matches the estimated path quite well. As was seen in simulation, the Shortest-Path controller diverges as it comes around the loop.

Figures 15(b) and 15(d) show the error between the estimated robot pose during mapping and the estimated true pose. These plots confirm that data collected by the Shortest-Path controller resulted in much more inaccurate map. By the end of the trajectory it was nearly 1.5 meters off course, while the Learned controller was no more than half a meter off of the true trajectory at any given time. These results indicate not only that the trajectories can have a large influence on map creation, but also that learning the best trajectories from simulation helps immensely during the mapping process.

8. Related Work

This is not the first work to consider the problem of active exploration for building the most accurate, lowest-uncertainty map. One popular approach in the literature, described in several places (Whaite and Ferrie 1997; Feder et al. 1999; Bourgault et al. 2002), is gradient ascent in information gain. This has substantial computational efficiency, in that the information gain need only be computed for the 8-connected neighboring states. The major disadvantage to the gradient ascent approach is that it is subject to local minima. While it has been argued that repeated observations taken from a local minimum will eventually flatten out the information gain at that pose, it may take a considerable amount of time for this process to occur, as is evidenced by the slow convergence of this approach in the experimental results. The gradient ascent approach also has no



(a) Trajectory (Shortest-Path)

(b) Pose Error

(c) Trajectory (Learned)

(d) Pose Error (Learned)

Fig. 15. Learned trajectories for a real experiment in the first floor of the Stata Center at MIT. (a) The trajectory of the Shortest-Path controller. (b) The robot pose error for the Shortest-Path controller. (c) The trajectory of the Learned controller. (d) The robot pose error for the Learned controller.

notion of "closing the loop" and cannot model long paths with large payoffs in map certainty at the end of the path.

Stachniss et al. (Stachniss and Burgard 2003) describe one of the few global exploration algorithms where they explicitly compute the information available at all locations in the environment. Their algorithm is closest in spirit to our algorithm, although they integrate an additional notion of travel costs to gather data at different locations in the environment. The disadvantage to their approach is that they are limited by computational costs to searching for the maximally informative single location, rather than a maximally informative trajectory through a series of locations in the environment.

Some attention has already been paid to the problem of completeness during automatic data collection, or coverage (Choset 2000; Acar et al. 2003), especially in the sensor network and computational geometry communities. Coverage exploration can often be defined in terms of sensor placement, that is, determining point locations to minimize the number of sensor measurements required to sense the entire space. With the exception of some

work in coverage behaviors (Choset 2001), these exploration strategies generally ignore the effect of motion between measurement points. However, data from a poorly chosen motion and sensing trajectory between measurement points can substantially interfere with the overall inference.

The problem of active exploration is related to a number of other sequential decision making problems. The active localization problem is one where the robot's position is unknown but the map *is* known. Fox et al. (Fox et al. 1998) use the same criterion of entropy minimization to find a trajectory that localizes the robot as quickly as possible. The authors choose a purely local gradient ascent approach, although they augment the information gain with a notion of relative costs of different actions.

Active exploration and localization are both specific instances of a more general planning framework, known as the Partially Observable Markov Decision Process (Sondik 1971), in which the controller makes decision based on the complete state distribution, rather than the mean estimate. Exact solutions for the optimal POMDP policy are computationally intractable, but good approximation algorithms have been emerged recently. The Augmented MDP (AMDP, also known as "Coastal Navigation") algorithm (Roy and Thrun 1999) in particular assumes that the state distribution is Gaussian, and uses this assumption to find efficient approximations to the POMDP policy. The AMDP approach could be applied to the active exploration problem, if the information gain were represented by an "information volume"; each voxel in the information volume would represent the current robot pose and map entropy. Given a transition function relating a voxel and action to some posterior voxel, the planning problem would be just be a shortest-path problem to the "lowest" reachable voxel in the information volume. The disadvantage to this approach is that computing the expected transitions in the information volume is infeasible for reasonably-sized maps.

More recently, non-myopic exploration algorithms have been developed by the sensor network community. The sub-modularity properties of information gain have been used to demonstrate efficient algorithms for planning in single (Meliou et al. 2007) and multi-robot domains (Singh et al. 2007). In contrast to this work, however, these algorithms do not model the information loss due to motion and are essentially forms of coverage. The closest algorithm in spirit to our reinforcement learning algorithm uses Gaussian processes to model the predicted reward of an exploring robot (Martinez-Cantin et al. 2007). The Gaussian process learner is able to learn multi-step trajectories and incorporate both predicted information gain from observations and information loss due to motion. However, the learning is performed on a per-step basis. Given a new destination for the robot, a new GP value function must be learned. Ours is the only work we are aware of that learns a general purpose policy for exploration that is independent of the environment.

Finally, active learning and statistical experiment design solve very related problems, addressing the question of, given a space of possible data (or queries, or experiments), what is the one additional measurement that would maximally improve the existing model (Cohn et al. 1996; Freund et al. 1997). The active learning approaches typically maximize the information gain criterion as we do in this work, but the disadvantage to these approaches is that they are not physically motivated; that is, there is no notion of "traveling" to acquire more data, there is no notion of a sequential decision making process that gathers a stream of data, and there is no notion of the relative costs of different queries.

9. Conclusion

We have demonstrated in this paper a novel approach for trajectory selection to explore unknown environments. We have shown that reinforcement learning is an effective way to identify control policies that can explore an unknown environment and collect data to minimize map error. The technical contributions are a problem formulation that allows the exploration for coverage to be balanced with the exploration for map accuracy using a constrained optimization representation, and a formulation of the exploration problem as a tractable partially observable Markov decision process. In particular, we demonstrate that the use of a policy class based

on smooth trajectories interpolating way-point constraints leads to efficient learning. By learning with this policy class of smooth trajectories, the robot is biased towards controllers that minimize error induced by its own motion. Additionally, by using a policy class where each action has considerable duration, the effective horizon length of the problem is dramatically reduced.

Our method relies on a physically accurate simulation of robot kinematics, but the parameters of the simulation were fitted to real data, allowing us to transfer policies learned in simulation seamlessly onto a real robot. We demonstrated that our algorithm was able to generate trajectories for exploration in large-scale environments resulting in much more accurate maps than those maps resulting from a conventional shortest-path controller. The learned exploration controller not only minimized map error but also prevented unbounded growth in error over the course of the exploration. While other forms of reinforcement learning are often environment-specific, our algorithm can generalize to many real-world environments efficiently. As a result, once trained, our learned policy was able to be executed in real time. We believe this is the only real-time current exploration algorithm capable of maximizing map accuracy.

A number of interesting questions are still to be addressed in the exploration literature. We relied on a conventional coverage planner to generate destination sensing locations, focusing only on the problem of map accuracy. A globally optimal solution would involve a coverage planner that can utilize the learned policy to make decisions about where to move. Secondly, our current formulation used the EKF formulation of mapping; as a result, our exploration algorithm largely focused on trajectories that minimized the Gaussian noise captured by the EKF model. We would like to extend our model to allow the learner to better predict trajectories that may violate the assumptions of the mapper, such as linearity or accurate data association.

REFERENCES

Acar, E. U., Choset, H., Zhang, Y., and Schervish, M. (2003). Path planning for robotic demining: Robust sensor-based coverage of unstructured environments and probabilistic methods. *International Jounal of Robotics Research* 22(7-8): 441–466.

Bagnell, J. D., Kakade, S., Ng, A., and Schneider, J. (2003). Policy search by dynamic programming. In *Neural Information Processing Systems*, volume 16. MIT Press.

Barraquand, J., Langlois, B., and Latombe, J. (1992). Numerical potential field techniques for robot path planning. *IEEE Transactions on Systems, Man and Cybernetics* 22(2): 224–241.

Bosse, M., Newman, P. M., Leonard, J. J., and Teller, S. (2005). Slam in large-scale cyclic environments using the atlas framework. *International Journal of Robotics Research* 23(12): 1113–1139.

Bourgault, F., Makarenko, A., Williams, S., Grocholsky, B., and Durrant-Whyte, H. (2002). Information based adaptive robotic exploration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. EPFL, Lausanne.

Bourque, E. and Dudek, G. (2000). On-line construction of iconic maps. In *Proc. IEEE International Conference in Robotics and Automation*. San Francisco, CA.

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

Choset, H. (2000). Coverage of known spaces: The boustrophedon cellular decomposition. *Autonomous Robots* 9: 247–253.

Choset, H. (2001). Coverage for robotics–A survey of recent results. *Annals of Mathematics and Artificial Intelligence* 31(1): 113–126.

Cohn, D. A., Ghahramani, Z., and Jordan, M. I. (1996). Active learning with statistical models. *Journal of Artificial Intelligence Research* 4: 129–145.

Cover, T. M. and Thomas, J. A. (1991). *Elements of information theory*. Wiley-Interscience, New York, NY, USA. ISBN 0471062596.

Davison, A. J., Reid, I. D., Molton, N. D., and Stasse, O. (June 2007). Monoslam: Real-time single camera slam. *Transactions on Pattern Analysis and Machine Intelligence* 29(6): 1052–1067.

Dellaert, F. and Kaess, M. (2006). Square root

sam: Simultaneous localization and mapping via square root information smoothing. *Int. J. Rob. Res.* 25(12): 1181–1203. ISSN 0278-3649.

Duan, K.-B. and Keerthi, S. S. (2005). Which is the best multiclass svm method? An empirical study. In *Multiple Classifier Systems*, volume 3541, 278–285.

Eliazar, A. and Parr, R. (2004). Learning probabilistic motion models for mobile robots. In *Proceedings of the International Conference on Machine Learning*, volume 69, 32.

Eustice, R., Walter, M., and Leonard, J. (2005). Sparse extended information filters: Insights into sparsification. In *Proceedings of the International Conference on Intelligent Robots and Systems*, 3281–3288.

Feder, H. J. S., Leonard, J. J., and Smith, C. M. (1999). Adaptive mobile robot navigation and mapping. *International Journal of Robotics Research, Special Issue on Field and Service Robotics* 18(7): 650–668.

Fedorov, V. (1972). *Theory of optimal experiments*. Academic press, New York.

Fox, D., Burgard, W., and Thrun, S. (1998). Active Markov localization for mobile robots. *Robotics and Autonomous Systems* 25(3-4): 195–207.

Freund, Y., Seung, H., Shamir, E., and Tishby, N. (1997). Selective sampling using the Query By Committee algorithm. *Machine Learning* 28: 133–168.

Ganguli, A., Cortes, J., and Bullo, F. (2006). Distributed deployment of asynchronous guards in art galleries. In *American Control Conference*, 1416–1421. Minneapolis, MN.

Haehnel, D., Burgard, W., and Thrun, S. (2003). Learning compact 3D models of indoor and outdoor environments with a mobile robot. *Robotics and Autonomous Systems* 44(1): 15–27.

Hähnel, D., Burgard, W., Fox, D., and Thrun, S. (2003). 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)*.

Haykin, S. (1994). *Neural networks: a comprehensive foundation*. MacMillan, Toronto, Canada.

Howard, A. and Roy, N. (2003). The robotics data set repository (Radish). URL `http://radish.sourceforge.net/`.

Hsu, C. and Lin, C. (2001). A comparison of methods for multi-class support vector machines. In *Technical report, National Taiwan University*. Taipei, Taiwan.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME-Journal of Basic Engineering* 82(Series D): 35–45.

Leonard, J. and Durrant-Whyte, H. F. (1991). Simultaneous map building and localization for an autonomous mobile robot. In *Proceedings of the IEEE International Workshop on Intelligent Robots and Systems*, 1442–1447. Osaka, Japan.

Lu, F. and Milios, E. (1997). Globally consistent range scan alignment for environment mapping. In *Autonomous Robots*, volume 4, 333–349.

Makarenko, A., Williams, S., Bourgault, F., and Durrant-Whyte, H. (2002). An experiment in integrated exploration.

Martinelli, A., Nguyen, V., Tomatis, N., and Siegwart, R. (2007). A relative map approach to SLAM based on shift and rotation invariants. *Robotics and Autonomous Systems* 55(1): 50–61.

Martinez-Cantin, R. and Castellanos, J. (2005). Unscented SLAM for Large-Scale Outdoor Environments. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 328–333.

Martinez-Cantin, R., de Freitas, N., Doucet, A., and Castellanos, J. (2007). Active policy learning for robot planning and exploration under uncertainty. In *Proc. Robotics: Science and Systems (RSS)*. Atlanta, GA, USA.

Meliou, A., Krause, A., Guestrin, C., and Hellerstein, J. (2007). Nonmyopic informative path planning in spatio-temporal models. In *Proc. of 22nd Conference on Artificial Intelligence (AAAI)*, 602–607.

Moravec, H. and Martin, M. (1994). Robot navigation by 3D spatial evidence grids. Technical report, Mobile Robot Laboratory, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Moutarlier, P. and Chatila, R. (1989). An experimental system for incremental environment modeling by an autonomous mobile robot. In *Proc. 5th International Symposium on Robotics*

Research. Tokyo.

Newman, P. M. and Leonard, J. J. (2003). Consistent convergent constant time slam. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-03)*, 1143–1150. Acapulco Mexico.

Ng., A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., and Liang, E. (2004). Inverted autonomous helicopter flight via reinforcement learning. *International Symposium on Experimental Robotics* .

Ollero, A. and Heredia, G. (1995). Stability Analysis of Mobile Robot Path Tracking. *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems* 3: 461–466.

Olson, E., Leonard, J., and Teller, S. (2006). Fast iterative optimization of pose graphs with poor initial estimates. In *International Conference on Robotics and Automation*, 2262–2269.

Pfaff, P., Triebel, R., Stachniss, C., Lamon, P., Burgard, W., and Siegwart, R. (2007). Towards mapping of cities. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*. Rome, Italy.

Porta, J., Spaan, M., Vlassis, N., and Poupart, P. (2006). Point-based value iteration for continuous POMDPs. *Journal of Machine Learning Research* 7: 2329–2367.

Roy, N. and McCallum, A. (2001). Toward optimal active learning through Monte Carlo estimation of error reduction. In *Proceedings of the International Conference on Machine Learning (ICML 2001)*. Williamstown, MA.

Roy, N. and Thrun, S. (1999). Coastal navigation with mobile robots. In *Advances in Neural Processing Systems 12*, volume 12, 1043–1049.

Sim, R. (2006). *On Visual Maps and their Automatic Construction.* Ph.D. thesis, McGill University.

Sim, R., Dudek, G., and Roy, N. (2004). Online control policy optimization for minimizing map uncertainty during exploration. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2004)*, 1758 – 1763. New Orleans, LA.

Singh, A., Krause, A., Guestrin, C., Kaiser, W., and Batalin, M. (2007). Efficient planning of informative paths for multiple robots. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*.

Smith, R., Self, M., and Cheeseman, P. (1990). *Autonomous Robot Vehicles*, chapter Estimating uncertain spatial relationships in robotics, 167–193. Springer-Verlag, Orlando, FL. ISBN 0-387-97240-4.

Sondik, E. (1971). *The Optimal Control of Partially Observable Markov Decision Processes*. Ph.D. thesis, Stanford University, Stanford, California.

Stachniss, C. (2006). *Exploration and Mapping with Mobile Robots*. Ph.D. thesis, University of Freiburg, Department of Computer Science.

Stachniss, C. and Burgard, W. (2003). Exploring unknown environments with mobile robots using coverage maps. In *Proc. of the International Conference on Artificial Intelligence (IJCAI)*. Acapulco, Mexico.

Stachniss, C., Grisetti, G., and Burgard, W. (2005). Information gain-based exploration using rao-blackwellized particle filters. In *Proc. of Robotics: Science and Systems (RSS)*, 65–72. Cambridge, MA, USA.

Thrun, S. (1998). Bayesian landmark learning for mobile robot localization. *Machine Learning* 33.

Thrun, S. (2000). Monte Carlo POMDPs. In *Advances in Neural Information Processing Systems*, volume 12, 1064–1070. MIT Press.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. The MIT Press, Cambridge, MA.

Thrun, S., Liu, Y., Koller, D., Ng, A., Ghahramani, Z., and Durrant-Whyte, H. (2004). Simultaneous localization and mapping with sparse extended information filters. *International Journal of Robotics Research* 23(7-8).

Thrun, S. and Montemerlo, M. (2005). The graph-slam algorithm with applications to large-scale mapping of urban structures. *International Journal on Robotics Research* 25(5/6): 403–430.

Walter, M., Eustice, R., and Leonard, J. (2005). A provably consistent method for imposing exact sparsity in feature-based slam information filters. In *Proceedings of the 12th International Symposium of Robotics Research (ISRR)*. San Francisco, CA.

Walter, M. and Leonard, J. (2004). An experimental investigation of cooperative SLAM. In *Proceedings of the Fifth IFAC Symposium on Intelligent Autonomous Vehicles*. Lisbon, Portugal.

Whaite, P. and Ferrie, F. (1997). Autonomous exploration: Driven by uncertainty. *IEEE Transactions on Patten Analysis and Machine Intelligence* 19(3): 193–205.

APPENDIX

The motion model with slip is given as $g(\xi_t, u_t, w_t)$ where the control consists of an initial rotation $d\theta_1$, a forward translation dr in the current heading direction of the robot θ and a final rotation $d\theta_2$, such that $u_t = (d\theta_1, dr, d\theta_2)$. An *effective* control $\tilde{u}_t = (d\tilde{\theta}_1, \tilde{dr}, d\tilde{\theta}_2, \tilde{ds})$ is generated, where each effective control component is a random variable depending on both the control and the noise process $w = (w_{d\theta_1}, w_{dr}, w_{d\theta_2}, w_{ds})$. An additional component $\tilde{ds}$ is included in the effective control to represent the slip motion, that is, cross-range error. The effective control variables are calculated as

$$\begin{aligned} w_{d\theta_1} &\sim \mathcal{N}(0,1) \\ d\tilde{\theta}_1 &= \mu_{\theta_1} + w_{d\theta_1} \cdot \sigma_{\theta_1} \\ \mu_{\theta_1} &= \alpha_{\theta,\theta} d\theta_1 + \alpha_{\theta,r} dr \\ \sigma^2_{\theta_1} &= \beta^2_{\theta,\theta} ||d\theta_1||^2 + \beta_{\theta,r} ||dr||^2 \end{aligned}$$

$$\begin{aligned} w_{dr} &\sim \mathcal{N}(0,1) \\ \tilde{dr} &= \mu_r + w_{dr} \cdot \sigma_r \\ \mu_r &= \alpha_{r,r} dr + \alpha_{r,\theta} d\theta \\ \sigma^2_r &= \beta^2_{r,r} ||dr||^2 + \beta_{r,\theta} ||d\theta||^2 \end{aligned}$$

$$\begin{aligned} w_{d\theta_2} &\sim \mathcal{N}(0,1) \\ d\tilde{\theta}_2 &= \mu_{\theta_2} + w_{d\theta_2} \cdot \sigma_{\theta_2} \\ \mu_{\theta_2} &= \alpha_{\theta,\theta} d\theta_2 + \alpha_{\theta,r} dr \\ \sigma^2_{\theta_2} &= \beta^2_{\theta,\theta} ||d\theta_2||^2 + \beta_{\theta,r} ||dr||^2 \end{aligned}$$

$$\begin{aligned} w_{ds} &\sim \mathcal{N}(0,1) \\ \tilde{ds} &= \mu_s + w_{dr} \cdot \sigma_s \\ \mu_s &= \alpha_{s,r} dr + \alpha_{s,\theta} d\theta \\ \sigma^2_s &= \beta^2_{s,r} ||dr||^2 + \beta_{s,\theta} ||d\theta||^2, \end{aligned}$$

where $d\theta = d\theta_1 + d\theta_2$. The means $\alpha_{i,j}$ and covariances $\beta_{i,j}$ describe the contribution of each control j to the effective control i, e.g., $\alpha_{\theta,r}$ is the mean contribution of the translation to the rotation. The controls $d\theta_1$ and $d\theta_2$ share model parameters, such that $\alpha_{\theta_1,r} = \alpha_{\theta_2,r} = \alpha_{\theta,r}$, so the subscripts 1 and 2 are omitted from the model parameters.

Each effective control has a dependence on the commanded translation and rotation, and each dependence consists of a bias and random variance, leading to a total of 16 parameters that must be estimated. An Expectation-Maximization procedure (Eliazar and Parr 2004) was used to estimate the parameters for the robot in Figure 14, and the parameters are given in table I.

A translational and rotational displacement ζ_t is then calculated from the effective controls as

$$\tilde{\zeta}_t = \begin{bmatrix} \tilde{dr}\cos(\theta_{t-1} + d\tilde{\theta}_1) + \tilde{ds}\cos(\theta_{t-1} + d\tilde{\theta}_1 + \frac{\pi}{2}) \\ \tilde{dr}\sin(\theta_{t-1} + d\tilde{\theta}_1) + \tilde{ds}\sin(\theta_{t-1} + d\tilde{\theta}_1 + \frac{\pi}{2}) \\ d\tilde{\theta}_1 + d\tilde{\theta}_2 \end{bmatrix}.$$

The complete motion can then be calculated as

$$\mathbf{x}_t = \mathbf{x}_{t-1} + \tilde{\zeta}_t. \tag{20}$$

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

TABLE I
THE SIXTEEN PARAMETERS FOR THE STOCHASTIC MOTION MODEL OF THE ROBOT.