Planning to learn: Integrating model learning into a trajectory planner for mobile robots

Proceedings of the 2009 IEEE
International Conference on Information and Automation
June 22 -25, 2009, Zhuhai/Macau, China

Planning to Learn: Integrating Model Learning into a Trajectory Planner for Mobile Robots

Matthew Greytak and Franz Hover

***Abstract*— For a mobile robot that performs online model learning, the learning rate is a function of the robot's trajectory. The tracking errors that arise when the robot executes a motion plan depend on how well the robot has learned its own model. Therefore a planner that seeks to minimize collisions with obstacles will choose plans that decrease modeling errors if it can predict the learning rate for each plan. In this paper we present an integrated planning and learning algorithm for a simple mobile robot that finds safe, efficient plans through a grid world to a goal point using a standard optimal planner, A*. Simulation results show that with this algorithm the robot practices maneuvers in the open regions of the configuration space, if necessary, before entering the constrained regions of the space. The robot performs mission-specific learning, acquiring only the information it needs to complete the task safely.**

I. INTRODUCTION

The exploration vs. exploitation problem is fundamental to many robot planning tasks: is it better to explore the state space and learn about the robot's environment, or is it better to complete the mission quickly using *a priori* knowledge? This tradeoff is a main component of reinforcement learning [1], [2] and planning in uncertain environments [3]. In trajectory planning for simultaneous localization and mapping (SLAM), the robot chooses a path through an unknown environment to reduce the uncertainty of the map features and the robot's own location [2], [4]. The multi-armed bandit is a standard exploration vs. exploitation problem in which an agent seeks to learn the reward distributions from different actions in order to maximize the total reward [5].

Instead of exploring the physical environment or a set of reward distributions, we consider the exploration of the vehicle's state. Learning algorithms have a faster convergence when presented with input data whose signal-to-noise ratio is large. Exploration of the state space can improve the convergence rate by presenting the learning algorithm with novel data; as the robot learns more about its own plant model, it can perform tasks more efficiently and with less error [6]. However, such exploration may delay the robot's mission. Therefore planning and learning are tightly linked for a robot that performs online learning. In this paper we study how to use this relationship to include learning in a well-established planning framework, A* search, by modifying the cost function to include learning-dependent information.

M. Greytak and F. Hover are with the Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, USA. mgreytak@mit.edu, hover@mit.edu.

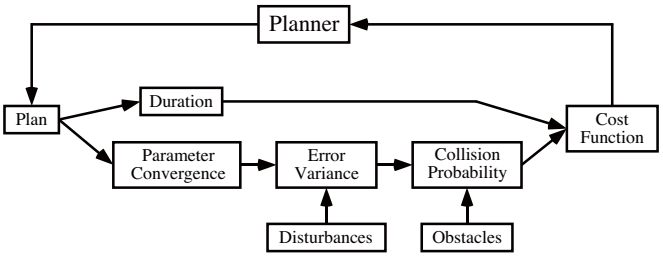

Fig. 1. The cost of each plan includes the collision probability, which is a function of the learning algorithm and external disturbances.

A. Integrated Strategy

Integrating learning and planning in a single framework requires two key abilities for the system:

1) *Predict the knowledge evolution for each plan.* – If a robot performs online learning, then different plans will result in different sets of knowledge that the robot will acquire. For example, as the robot makes more left turns, it will get better at making left turns.
2) *Predict the tangible cost associated with each plan, given the knowledge evolution.* – In general the execution errors when following the plan will be larger at the beginning of the learning process than at the end. These errors must be translated into a tangible cost for the system, such as the probability of hitting obstacles or the root mean square path-following error.

As long as the cost is related to the amount of learning that the robot has experienced, then a path planning search algorithm that chooses the minimum-cost plan will *automatically incorporate learning strategies in the plan* as necessary, even if it means temporarily deviating from the mission.

In this paper, *knowledge* refers to the values of the various system parameters within a known model structure; increasing knowledge in this case refers to decreasing the variance of the parameter values. The effectiveness of learning algorithms depends on the quality of the data provided to them, so the amount of parameter convergence we can expect depends on the input data which is a function, in turn, of the plan that the robot is following. The exact relation depends on the details of the plant and the learning algorithm.

The cost we consider is a combination of the plan duration and the probability of colliding with obstacles while executing the plan; in this way, a plan that minimizes the cost strikes a balance between efficiency (time) and risk. The plan duration is trivial to compute, but the collision risk is a function of the robot's dynamics, external disturbances,

978-1-4244-3608-8/09/$25.00 © 2009 IEEE.

and the quality of the controller used to move the robot [7]. The controller is designed from the estimates of the system parameters; consequently the parameter convergence drives the collision risk. Figure 1 shows a block diagram of the cost function.

Parameter convergence does not appear explicitly in the cost function. The robot is not intrinsically motivated, as in [8] and the references cited above: it does not seek to learn for learning's sake, but rather to learn what is necessary to perform the given task safely and efficiently. The integrated planning and learning strategy can be applied to any system as long as the knowledge evolution and resulting cost can be predicted accurately. We seek analytic solutions to the parameter convergence and collision risk problems so that the cost function can be computed without resorting to costly Monte Carlo simulations. It is possible to analytically compute the collision risk for a holonomic planar vehicle given a known level of parameter convergence [9]; parameter convergence predictions depend on the specific vehicle model and the learning algorithm.

B. Simple Proof of Concept

In this paper we consider a mobile robot moving on a Cartesian grid from a start point to a specified goal point. The robot uses the A* search algorithm [10] to generate plans that move between adjacent grid points (no diagonals) to the goal. The control action is an impulse that pushes the robot from one grid point to the next. The appropriate strength of the impulse must be learned by the robot as it executes the plan; we use least squares regression as the learning algorithm. We assume that the robot has perfect knowledge of its position, it has a perfect map of the environment, and there are no external disturbances. We also assume that x and y motions of the robot are uncoupled. While the vehicle model is simplistic and the assumptions are restrictive, they serve to make the parameter convergence prediction and the error prediction mathematically simple. The plant model and the control law are described in Section II. The Least Squares learning algorithm and its regularization are discussed in Section III. Section IV describes the cost function and the heuristic used by the A* planning algorithm. Simulation results are presented in Section V.

II. PLANT DESCRIPTION AND CONTROL LAW

Consider a mobile robot that moves on a Cartesian grid (Figure 2). The position of the robot is $\mathbf{x} = [x, y]^T$. The robot has a control input vector $\boldsymbol{\tau} = [\tau_x, \tau_y]^T$ that is used to push the robot from one point to another. The magnitude of the resulting position change depends on the scalar gains b_x and b_y which form the diagonal of the 2×2 matrix $\mathbf{B}$. Time is discretized by the index k. The system model is shown below.

$$\begin{aligned} \begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} &= \begin{bmatrix} x_k \\ y_k \end{bmatrix} + \begin{bmatrix} b_x & 0 \\ 0 & b_y \end{bmatrix} \begin{bmatrix} \tau_{xk} \\ \tau_{yk} \end{bmatrix} \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \mathbf{B}\boldsymbol{\tau}_k \end{aligned} \tag{1}$$

The robot is asked to follow a reference $\mathbf{r}_k = [r_{x,k}, r_{y,k}]^T$ where $r_{x,k}$ and $r_{y,k}$ are the reference x and y positions, respectively, at time t_k. The position error is $\mathbf{e}_k = \mathbf{x}_k - \mathbf{r}_k$. The reference path can be expressed using the cardinal directions North (`N`), South (`S`), East (`E`), West (`W`), and no motion (`0`). For example, the plan shown in Figure 2 is `EEENNNWSWN`.

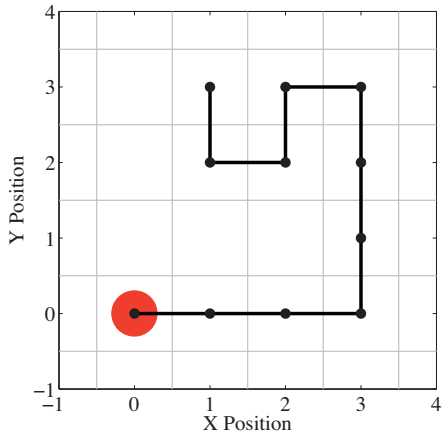

Fig. 2. The grid world in which the robot operates, with an example reference trajectory `EEENNNWSWN`. The robot moves counter-clockwise around the path.

To arrive exactly at $\mathbf{r}_{k+1}$, (1) can be solved to find the appropriate control actions at time k.

$$\boldsymbol{\tau}_k = \mathbf{B}^{-1}(\mathbf{r}_{k+1} - \mathbf{x}_k) \tag{2}$$

However, if the control gains $\mathbf{B}$ are not known exactly, then the best estimates $\hat{\mathbf{B}} = \text{diag}([\hat{b}_x,\ \hat{b}_y])$ must be used instead. The final control law can either be written in terms of the state $\mathbf{x}_k$ or the state error $\mathbf{e}_k$.

$$\begin{aligned} \boldsymbol{\tau}_k &= \hat{\mathbf{B}}^{-1}\left(\mathbf{r}_{k+1} - \mathbf{x}_k\right) \\ &= \hat{\mathbf{B}}^{-1}\left(\mathbf{r}_{k+1} - (\mathbf{r}_k + \mathbf{e}_k)\right) \\ &= \hat{\mathbf{B}}^{-1}(\mathbf{r}_{k+1} - \mathbf{r}_k) - \hat{\mathbf{B}}^{-1}\mathbf{e}_k \end{aligned} \tag{3}$$

Using (3), the error evolves as follows:

$$\begin{aligned} \mathbf{e}_{k+1} &= (\mathbf{I} - \mathbf{B}\hat{\mathbf{B}}^{-1})(\mathbf{x}_k - \mathbf{r}_{k+1}) \\ &= (\mathbf{I} - \mathbf{B}\hat{\mathbf{B}}^{-1})\left(\mathbf{e}_k - (\mathbf{r}_{k+1} - \mathbf{r}_k)\right) \end{aligned} \tag{4}$$

III. LEAST SQUARES LEARNING ALGORITHM

The Least Squares (LS) algorithm is used to learn the parameter values $\hat{b}_x$ and $\hat{b}_y$. The LS algorithm compares a measured output to a predicted output and uses the difference to adjust the parameter values. The resulting parameter convergence rate can be predicted analytically. In practice the Recursive Least Squares (RLS) algorithm could be used with identical results, but to simplify the analysis in this paper we use the LS algorithm.

A. Algorithm Description

The parameters used by the control law (3) at t_k are $\hat{b}_{x,k}$ and $\hat{b}_{y,k}$. The output used to generate the parameter estimates is $\mathbf{z}_k = \mathbf{x}_k - \mathbf{x}_{k-1}$. This value is measured with additive zero-mean Gaussian noise $w_{x,k}$ and $w_{y,k}$. Because the x and y channels are uncoupled in (1), the two channels

can be handled separately. The following analysis applies to both channels, and the subscripts x and y are removed for notational convenience.

$$z_k = b\,\tau_{k-1} + w_k, \quad k > 0 \tag{5}$$

The control value τ_k is the input to the learning algorithm; it is stored in the $1 \times N$ vector $\mathbf{\Phi}$, where N is the length of the learning data. The outputs are stored in the vector $\mathbf{Z}$. The noise values are sampled independently from a normal random process with a noise variance W: $w_k \sim N(0, W)$. The noise values are stored in the vector $\mathbf{w}$.

$$\begin{aligned} \mathbf{\Phi} &= [\tau_0 \ldots \tau_{N-1}] \\ \mathbf{Z} &= [z_1 \ldots z_N] \\ \mathbf{w} &= [w_1 \ldots w_N] \end{aligned} \tag{6}$$

In this notation, (5) can be rewritten as follows for the entire data set:

$$\mathbf{Z} = b\,\mathbf{\Phi} + \mathbf{w} \tag{7}$$

The least squares estimate for the parameter using the entire data set is:

$$\hat{b}_N = \left(\mathbf{\Phi}\mathbf{\Phi}^T\right)^{-1}\mathbf{\Phi}\mathbf{Z}^T \tag{8}$$

and the prediction of the outputs given the parameter estimate $\hat{b}_N$ is $\hat{\mathbf{Z}}$.

$$\hat{\mathbf{Z}} = \hat{b}_N\mathbf{\Phi} \tag{9}$$

For convenience we define $R = \mathbf{\Phi}\mathbf{\Phi}^T$. This value represents the information content of the input data. The expected value of the parameter estimate is equal to the true value if the noise is uncorrelated with the input, which we assume is true.

$$\begin{aligned} E[\hat{b}_N] &= E\left[R^{-1}\mathbf{\Phi}\left(\mathbf{\Phi}^T b + \mathbf{w}^T\right)\right] \\ &= R^{-1}R\,b + R^{-1}E\left[\mathbf{\Phi}\mathbf{w}^T\right] \\ &= b \\ \text{if} \quad E\left[\mathbf{\Phi}\mathbf{w}^T\right] &= 0 \end{aligned} \tag{10}$$

The parameter variance P is derived below.

$$\begin{aligned} P &= E\left[(\hat{b}_N - b)(\hat{b}_N - b)^T\right] \\ &= E\left[R^{-1}\mathbf{\Phi}\mathbf{w}^T\mathbf{w}\mathbf{\Phi}^T R^{-1}\right] \\ &= WR^{-1}RR^{-1} \\ &= WR^{-1} \end{aligned} \tag{11}$$

B. Regularization

If the input channel has not been sufficiently excited (that is, $\forall k,\ \tau_k \approx 0$), then R will be close to zero and the parameter variance P will be large. This means the parameter estimates themselves will be poor. It is often the case that some *a priori* knowledge about the parameter value is available.

Consider the problem in which the learning occurs in two stages. In the first stage the input data is $\mathbf{\Phi}_0$ and the output data is $\mathbf{Z}_0$, and in the second stage the input data is $\mathbf{\Phi}_1$ and the output data is $\mathbf{Z}_1$. The respective lengths of the data sets are N_0 and N_1. While the LS algorithm could be applied to each data set separately, we are interested in the parameter estimate resulting from the entire data set taken together, $\mathbf{\Phi} = [\mathbf{\Phi}_0\ \mathbf{\Phi}_1]$ and $\mathbf{Z} = [\mathbf{Z}_0\ \mathbf{Z}_1]$. It is easily discovered that the information for the total data set is $R = R_0 + R_1$. Through two applications of the Woodbury matrix identity, we arrive at a useful result.

$$\begin{aligned} \hat{b}_N &= R^{-1}\mathbf{\Phi}\mathbf{Z}^T \\ &= (R_0 + R_1)^{-1}\left(\mathbf{\Phi}_0{\mathbf{Z}_0}^T + \mathbf{\Phi}_1{\mathbf{Z}_1}^T\right) \\ &= \hat{b}_0 + {R_0}^{-1}\mathbf{\Phi}_1\Big(\mathbf{I}_{N_1\times N_1} + \\ &\qquad + {\mathbf{\Phi}_1}^T{R_0}^{-1}\mathbf{\Phi}_1\Big)^{-1}\left(\mathbf{Z}_1 - \hat{\mathbf{Z}}_1\right)^T \\ &= \hat{b}_0 + {R_0}^{-1}\left(1 - R_1(R_0 + R_1)^{-1}\right)\mathbf{\Phi}_1\left(\mathbf{Z}_1 - \hat{\mathbf{Z}}_1\right)^T \end{aligned} \tag{12}$$

The intermediate result of (12) is the Recursive Least Squares (RLS) algorithm when $N_1 = 1$. Note that the final result does not include the input/output data from the first stage, only the information R_0 and the parameter estimate $\hat{b}_0$.

Returning to the regularization problem, suppose an initial guess of the parameter value is $\hat{b}_0$ and its variance is approximated as P_0. A large P_0 indicates low confidence in the validity of $\hat{b}_0$, and a small P_0 indicates high confidence. From (11) we can construct an equivalent information value $R_0 = W/P_0$. Next we can use (12) to evaluate the parameter estimate given the *a priori* knowledge that we have. As real data is collected, the effect of the initial estimate $\hat{b}_0$ will disappear. The persistence of the initial estimate is a function of the initial variance P_0.

C. Predicting the Parameter Convergence Rate

For planning purposes it is useful to be able to predict the expected parameter variance at any time during a motion plan. From (11) we see that the expected P is:

$$E[P] = WE[R^{-1}] \tag{13}$$

where the expectation is taken over the sensor noise values that corrupt the parameter estimate throughout the plan. The first two nonzero terms of the Taylor series approximation to (13) are shown below. In the following analysis we use only the first term of the series.

$$\begin{aligned} E[P] &= W\left(E[R]^{-1} + E[R]^{-3}\text{var}(R) + \cdots\right) \\ &\approx WE[R]^{-1} \end{aligned} \tag{14}$$

The expected information $E[R]$ is computed using the linearity of the expectation operator:

$$\begin{aligned} E[R] &= E\left[\sum_{k=0}^{N-1}{\tau_k}^2 + R_0\right] \\ &= \sum_{k=0}^{N-1}E[{\tau_k}^2] + R_0 \\ &= \sum_{k=0}^{N-1}\left(\bar{\tau}_k^2 + \text{var}(\tau_k)\right) + R_0 \end{aligned} \tag{15}$$

where $\bar{\tau}_k = \frac{1}{b}(r_{k+1} - r_k)$ and $\text{var}(\tau_k) = E[(\tau_k - \bar{\tau}_k)^2]$. The variance is computed as follows.

$$\begin{aligned} \tau_k &= \frac{1}{\hat{b}_k}(r_{k+1} - x_k) \\ &\approx \left(\frac{1}{b} - \frac{\hat{b}_k - b}{b^2}\right)(r_{k+1} - r_k) \\ \tau_k - \bar{\tau}_k &= -\frac{\hat{b}_k - b}{b^2}(r_{k+1} - r_k) \\ E[(\tau - \bar{\tau}_k)^2] &= \frac{1}{b^2}(r_{k+1} - r_k)^2 E[P_k] \\ \text{so} \quad E[{\tau_k}^2] &= \bar{\tau}_k^2(1 + E[P_k]) \end{aligned} \tag{16}$$

where $E[P_k]$ is the expected parameter variance after k data points. The expected parameter variance of $\hat{b}_N$ is $E[P_N]$, computed using the following recursive relation with P_0 defined in the previous section.

$$E[P_N] = W\left(\sum_{k=0}^{N-1} \bar{\tau}_k^2(1 + E[P_k]) + \frac{W}{P_0}\right)^{-1} \tag{17}$$

Using the same approximation as in (16), we can express the error evolution as:

$$\begin{aligned} e_k &= \left(1 - \frac{b}{\hat{b}_{k-1}}\right)(e_{k-1} - (r_k - r_{k-1})) \\ &\approx \frac{\hat{b}_{k-1} - b}{b}(e_{k-1} - (r_k - r_{k-1})) \end{aligned} \tag{18}$$

If we define the variance of the error as $V_k = E[{e_k}^2]$, then the variance evolves as follows.

$$\begin{aligned} E[{e_k}^2] &= E\left[\frac{(\hat{b}_{k-1} - b)^2}{b^2}\left({e_{k-1}}^2\right.\right. \\ &\qquad \left.\left. -2e_{k-1}(r_k - r_{k-1}) + (r_k - r_{k-1})^2\right)\right] \\ V_k &= \frac{E[P_{k-1}]}{b^2}V_{k-1} + E[P_{k-1}]\,{\bar{\tau}_{k-1}}^2 \end{aligned} \tag{19}$$

The two parameter estimates $\hat{b}_{x,k}$ and $\hat{b}_{y,k}$ are assembled in the diagonal matrix $\hat{\mathbf{B}}_k$. The corresponding diagonal parameter variance matrix is $\mathbf{P}_k$. For the reference trajectory shown in Figure 2 with $\mathbf{P}_0 = 0.1 \times \mathbf{I}$ and $W = 0.01 \times \mathbf{I}$, the predicted parameter variance is plotted in Figure 3. The predictions are compared with the sample variance from 10^4 Monte Carlo simulations. The true parameter values are $b_x = 1$ and $b_y = -1$, and the *a priori* parameter estimates used in the Monte Carlo simulations are sampled from normal distributions whose variances are $\mathbf{P}_0$, centered around the true parameter values: $\hat{\mathbf{B}}_0 \sim N(\mathbf{B}, \mathbf{P}_0)$.

In Figure 3, note that the y-channel parameter variance P_y remains at its initial value $P_{y,0}$ until the reference path moves in the vertical direction. Until that time there is no information in the y channel, so no learning can take place for $\hat{b}_y$.

The parameter variance prediction depends on the *a priori* parameter estimate $\hat{\mathbf{B}}_0$. If this value is incorrect then it affects the scaling of the predictions. However, the trends through a given plan do not change; that is, the *shape* of each curve in Figure 3 is the same. Incorrect *a priori* data is addressed in the second example in Section V.

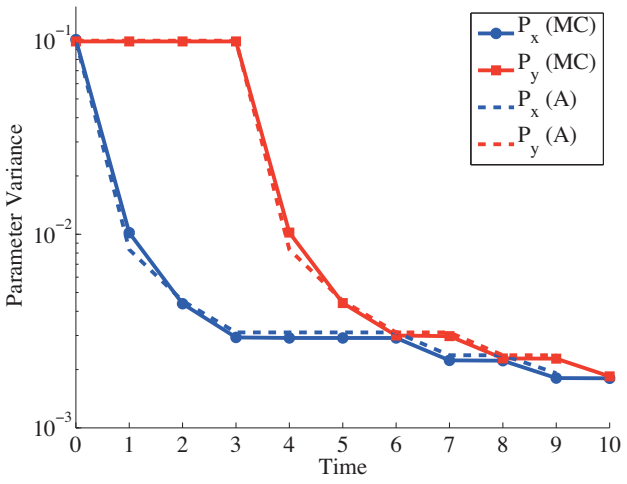

Fig. 3. Analytic parameter variance predictions (dashed) and Monte Carlo results (solid) for the reference trajectory shown in Figure 2.

IV. A* PLANNING ALGORITHM

The A* planning algorithm [10] is used to find collision-free paths through the configuration space to a goal position. A search tree expands out from the robot's initial position; the planner stops when it finds a plan ending at the goal that is guaranteed to be the plan with the lowest cost. The A* algorithm requires a cost function g to compare different plans, and a heuristic function h that is an optimistic prediction of the cost from the end of the plan to the goal. Standard algorithms for finding the shortest paths in grid worlds do not apply in this case, because the plan cost at any location depends on the path used to get there.

A. Cost Function

We are interested in minimizing a combination of the time to the goal and the risk of collisions along the way. The time to the goal T is simply the length of the plan. The collision probability is calculated by combining the effects of all obstacles. At each step of the plan, the robot's position is normally distributed around the reference position. The position variance for each channel, $V_{x,k}$ and $V_{y,k}$, is computed from (19). If the distance between the edge of the robot at t_k and the edge of the j'th obstacle is d_{kj}, and the angle of the shortest connecting line is θ_{kj}, then the variance in the θ_{kj} direction is $V_{kj} = \cos^2(\theta_{kj})V_{x,k} + \sin^2(\theta_{kj})V_{y,k}$. The probability density function for the vehicle's position on the θ_{kj} axis, x_θ, is:

$$p(x_\theta) = \frac{1}{\sqrt{2\pi V_{kj}}} \exp\left(-\frac{{x_\theta}^2}{2V_{kj}}\right) \tag{20}$$

The probability of hitting an obstacle is the integral of (20) from the edge of the obstacle to infinity:

$$P_{hit,kj} = \frac{1}{2} - \frac{1}{2}\,\text{erf}\left(\frac{d_{kj}}{\sqrt{2V_{kj}}}\right) \tag{21}$$

For the vehicle to execute the entire plan without hitting an obstacle, it must pass each obstacle without a collision. The overall collision probability for the plan is:

$$P_{hit} = 1 - \prod_{k=0}^{N} \prod_{j=1}^{\#\,obs} (1 - P_{hit,kj}) \tag{22}$$

If the robot must keep trying to execute the plan until it can do so without hitting any obstacles, then the expected time to complete the plan is:

$$g = \frac{T}{1 - P_{hit}} \tag{23}$$

This cost function (23) is a nonlinear function that strongly discourages plans with a high collision probability: $g \to \infty$ as $P_{hit} \to 1$. An alternative cost function linearly combines T and P_{hit}, but that approach requires tuning a weighting parameter. Note that parameter convergence does not appear explicitly in the cost function, but it implicitly affects the collision probability.

B. Heuristic

The A* algorithm can only return the optimal plan if the heuristic used by the planner, h, never overestimates the cost from the end of the plan to the goal. An appropriate heuristic is the duration of the shortest path to the goal that does not intersect with any obstacles. To create the heuristic, we first form a network graph $\mathcal{G}$ between the corners of the obstacles, which are the nodes of the graph. The edge cost c_{mn} between two nodes m and n is the Manhattan distance because the robot cannot move diagonally. Next we run Dijkstra's algorithm [11] on the graph to compute the minimum Manhattan distance from each node to the goal; this defines the node cost C_n. These steps are performed offline before the A* algorithm executes. Online, to compute h for a plan, the planner finds the Manhattan distance from the end of the plan to each node, c_{kn}. If there is an obstacle between the end of the plan and the node, then $c_{kn} = \infty$. The length of the shortest path from the end of the plan to the goal is found by comparing the routes through all nodes:

$$h = \min_{n \in \mathcal{G}} (C_n + c_{kn}) \tag{24}$$

C. Search Procedure

Starting from the initial position, the planner branches out by concatenating actions a from the set $\{\texttt{N}, \texttt{S}, \texttt{E}, \texttt{W}, \texttt{0}\}$. The total number of plans may be infinite, even if the configuration space is bounded, because different plans can arrive at the same location through different paths. Unlike standard shortest-path problems, plans with the same length may have different costs. The plans are stored in a queue $\mathcal{Q}$ that is sorted by each plan's predicted total cost, $f = g + h$.

The search procedure is listed in Algorithm 1. As long as a free path exists from the start to the goal, which can be verified using the heuristic h evaluated at the start location, the algorithm returns the optimal plan (the plan with the smallest cost g that ends at the goal).

Algorithm 1 Expanding A* Motion Planning Algorithm.

```
Load the obstacle map and the goal position.
Construct the obstacle graph G.
Load the a priori parameter estimates B̂_0 and the initial variance P_0.
Initialize the search queue Q with a plan containing only the start position.
while Q is not empty do
    Remove the first plan from the queue, p ← Q(0).
    if p ends at the goal then
        return p with success.
    end if
    for each action a ∈ {N, S, E, W, 0} do
        Add the action to p: p' ← p + a.
        if p' is collision-free then
            Compute the cost g(p').
            Compute the predicted cost-to-go h(p').
            Insert p' into Q sorted by f(p') = g(p') + h(p').
        end if
    end for
end while
return with failure.
```

V. SIMULATION RESULTS

We present two examples to show how the integrated planner creates plans that automatically incorporate learning strategies. In the first example, the robot is given the task of driving to a point in the hallway shown in Figure 4. The shortest path is to drive straight into the hallway and proceed up to the goal. However, in that case the first vertical movement that the robot executes is in the constrained region of the hallway. With the initial parameter estimate $\hat{\mathbf{B}}_0 = \mathbf{B}$, an initial parameter variance $\mathbf{P}_0 = \mathbf{I}$, and sensor noise variance $\mathbf{W} = 0.1 \times \mathbf{I}$, the planner chooses the plan ENSEEEEENNN shown in the figure. This plan was found in 0.142 seconds after 251 iterations of the A* algorithm on a 2.33 MHz Intel Core 2 Duo processor.

Even though the initial parameter estimates are actually correct in this example, the large variance (low confidence) of those estimates causes the robot to try to learn the parameters by practicing the actions in the unconstrained region of the space. This plan has a predicted collision probability of 6.0%. A set of 10^4 Monte Carlo simulations results in a measured collision probability of 7.6%. The difference can be attributed to the approximations made when predicting the parameter convergence (14,16) and the resulting error variance (18). The Monte Carlo collision probability of the shortest plan EEEEEENNN, which drives straight into the hallway, is 17.6%.

A more complicated example is shown in Figure 5. As in the first example, the robot chooses to practice before entering the hallway; the plan is WWNSEEEENNEESS (found in 0.654 seconds after 1,943 iterations). In this case, the initial parameter estimates are $\hat{b}_{x,0} = 2$ and $\hat{b}_{y,0} = -0.5$, which are 200% and 50% of the true values, respectively. $\mathbf{P}_0$

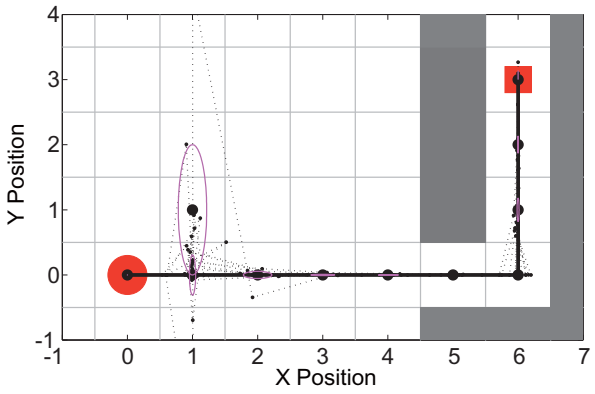

Fig. 4. The robot starts at the circle and drives to the square. It chooses to practice the vertical motion before entering the hallway so that the parameter variance is reduced. The ellipses show one standard deviation of predicted position error. The dotted lines show 10 representative Monte Carlo simulations.

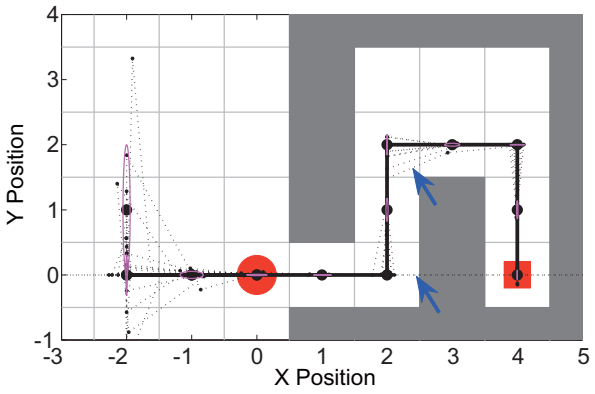

Fig. 5. The robot moves away from the goal to learn a controller before entering the hallway. The ellipses show one standard deviation of predicted position error. The dotted lines show 10 representative Monte Carlo simulations, two of which have collisions (indicated with arrows).

and $\mathbf{W}$ are the same as in the previous example. Even though the *a priori* estimates are wrong, the robot chooses a plan that will correct those estimates before driving close to obstacles. In this example the predicted collision probability is 24.3%, and the collision probability evaluated from 10^4 Monte Carlo simulations is 22.4%. The Monte Carlo collision probability of the shortest plan EENNEESS, which drives straight into the hallway, is 74.0%.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have presented an integrated planning and learning algorithm for a simple mobile robot. Because the relation between model learning and collision risk is known, plans that seek to reduce the collision risk will include actions that contribute to learning the model. By including the learning process implicitly in the cost function, but not explicitly (with a weighting parameter, for example), the robot only tries to learn as much as is necessary to perform the task safely and efficiently (mission-specific learning). The result of this strategy is that the robot practices maneuvers that it knows it will need in the future before using those maneuvers near obstacles.

This integrated strategy has the most impact when the initial parameter variance is large; in that case it takes very few maneuvers to significantly reduce the parameter variance and improve the predicted error variance. As R increases, the effect of each subsequent action is reduced. The algorithm presented in this paper is particularly suited to situations where (a) the robot is starting from a nearly clean slate with very little knowledge of its system parameters, or (b) the robot senses a configuration change and chooses to restart the learning algorithm to learn the new parameters. The latter situation is easily handled by resetting $\mathbf{P}_0$ to an appropriate value and running the planner again.

While the system model and controller used in this paper are very simplistic, they serve as a proof-of-concept for integrated strategies for general mobile robots. As long as the effect of parameter error on the cost function can be predicted, the planner will be able to automatically incorporate learning strategies into the motion plan of any mobile robot. We are currently extending these results to a 3 DOF holonomic robot that is subjected to external disturbances.

VII. ACKNOWLEDGMENTS

This work was supported by the Office of Naval Research, Grant N00014-02-1-0623, and by a Science, Mathematics and Research for Transformation (SMART) scholarship (M. Greytak).

REFERENCES

[1] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[2] T. Kollar and N. Roy. Trajectory optimization using reinforcement learning for map exploration. *International Journal of Robotics Research*, 27(2):175, 2008.

[3] M. Rickert, O. Brock, and A. Knoll. Balancing exploration and exploitation in motion planning. In *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, pages 2812–2817, 2008.

[4] S. Huang, N. M. Kwok, G. Dissanayake, Q. P. Ha, and G. Fang. Multi-step look-ahead trajectory planning in SLAM: Possibility and necessity. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1091–1096, 2005.

[5] T. L. Lai. Adaptive treatment allocation and the multi-armed bandit problem. *Annals of Statistics*, 15(3):1091–1114, 1987.

[6] J. C. Bongard, V. Zykov, and H. Lipson. Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121, 2006.

[7] M. Greytak and F. Hover. Robust motion planning for marine vehicle navigation. In *Proceedings of the 18th International Offshore and Polar Engineering Conference*, volume 2, pages 399–406, Vancouver, Canada, 2008.

[8] F. Kaplan and P-Y. Oudeyer. Intrinsically motivated machines. In *50 Years of Artificial Intelligence*, pages 303–314. Springer, Heidelberg, 2007.

[9] M. Greytak and F. Hover. Analytic error variance predictions for planar vehicles. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, Kobe, Japan, 2009. to appear.

[10] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[11] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.