The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

A Learning Approach to Integration of Layers of a Hybrid Control Architecture

Matthew Powers and Tucker Balch

Abstract— **Hybrid deliberative-reactive control architectures are a popular and effective approach to the control of robotic navigation applications. However, the design of said architectures is difficult, due to the fundamental differences in the design of the reactive and deliberative layers of the architecture. We propose a novel approach to improving system-level performance of said architectures, by improving the deliberative layer's model of the reactive layer's execution of its plans through the use of machine learning techniques. Quantitative and qualitative results from a physics-based simulator are presented.**

I. INTRODUCTION

Hybrid deliberative-reactive control architectures for robotic navigation have long been an active area of research. Despite their success, open questions remain how to best integrate the layers to maximize overall system performance. In this work, we propose a novel method to improve the integration of deliberative planning and reactive control in a robotic navigation system. In particular, we will use machine learning techniques to improve the deliberative layer's model of the reactive layer's interpretation of its plans.

Today, many modern systems use an implementation of a hybrid layered approach to robot control architecture. Hybrid and hierarchical layered approaches make use of decoupled layers of functionality to satisfy both the robot's immediate constraints and its longer-term objectives. In the case of robot navigation (especially in the area of field robotics), many modern architectures make use of a lower-fidelity global deliberative planner and a higher-fidelity local reactive controller [1], [2], [3].

Finding a compromise between global objectives and local constraints is not always easy, and often the tradeoffs have to be empirically "fine-tuned" by the robot software designer. Either the deliberative layer's model of the world and the robot, or the reactive layer's interpretation of the deliberative layer's input must be adjusted. This process can be time-consuming and is subject to human interpretation of the robot's performance. It can also simply be difficult for a human to make sense of how all the degrees of freedom that a complex software system may contain might affect the robot's system-level performance.

We propose an approach to improving system-level performance of hybrid control architectures by learning models of the reactive layer's execution of the deliberative layer's plans, based on measurements of actual executions. Our approach begins by measuring the performance of the execution of the plans in the context of a training environment. Machine learning techniques, particularly supervised learning, are used to abstract that performance to predict performance in other environments. Then, this learned model is fed back to the planner for use in creating plans with better overall performance.

Matthew Powers and Tucker Balch are with the College of Computing, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30332, USA `mpowers@cc.gatech.edu`, `tucker@cc.gatech.edu`

II. RELATED WORK

A. Hierarchical and Hybrid Control Architectures

Arkin's AuRA architecture [4] and Gat's Atlantis architecture [5] are early examples of hybrid deliberative-reactive architectures. In both, the reasoning done by the deliberative layer is fundamentally different from that done by the reactive layer. The deliberative layer works to achieve global goals based on world models. The reactive layer works to achieve local constraints based on current sensor input. Each architecture suggests methods for combining the globally-based deliberative input with the locally-based reactive reasoning.

Layered architectures have been especially successful in the area of field robotics. Albus' 4D-RCS architecture [1] builds upon the concept of layered architectures, implementing an extended hierarchy of layers. Stanley, the robot winner of the DARPA Grand Challenge [2] made use of a global trajectory planner advising a lower-level real-time controller to navigate 120 miles of desert. The winner of the DARPA Urban Challenge, Boss [3], made use of an architecture that included a global task planner, a local path planner and a local behavior-based design to navigate over 60 miles of urban terrain.

B. Learning within Hierarchical Control Architectures

Because of the difficulties associated with assuring system-level performance within hierarchical control systems, a body of work has evolved promoting learning across layer boundaries or across task decompositions. Stone [8] implemented an approach to task decomposition and learning within the context of robot soccer. Rather than using reinforcement learning at all layers, Stone relied on human insight to choose appropriate learning techniques at each layer. Higher-level layers were learned making explicit use of learned low-level layers. Stone extended this hierarchy up to the team level and demonstrated its effectiveness in several international competitions.

In [9] Balch demonstrated the use of reinforcement learning for robots to learn a sequential layer strategy in the form of a finite state automata (FSA), based on a designer-implemented reactive layer. Balch implemented a set of behavioral assemblages, and defined states in an FSA mapping to each behavioral assemblage. He then used Q-learning to learn transitions between the states in the FSA.

III. REPRESENTATION

Following the pattern of other hybrid control architectures, we divide the architecture into two distinct components, the reactive layer and the deliberative layer. The reactive layer is responsible for monitoring the robot's sensors, performing low-level navigation and decision making, and actuating the robot's motors. We model the robot's reactive layer as a continuous controlled dynamical system. The deliberative layer is responsible for integrating sensor input into maps, and planning routes and actions toward a given goal. We model the deliberative layer as a discrete process providing regular input to the reactive layer.

A. Sensing and Reactive Control

We begin by modeling the robot as a controlled dynamical system,

$$\dot{x} = f(x,u),\ x \in \mathbb{R}^a,\ u \in \mathbb{R}^b \tag{1}$$

which exists in a world $W \subset \mathbb{R}^d$.

We are given a measurement equation,

$$y = h_y(x) \tag{2}$$

that provides access to the state of the system. Additionally, we are given another measurement equation that gives sensory access to the state of the world, $h_s(x,w)$. We then define s as the collection of all observable sensory input:

$$s = \{h_s(x,w)\}_{w \in W} \tag{3}$$

We can then close the loop, defining the control input u as a function of the measurements of both the system state and the environment state. Thus,

$$u = g(y,s) \tag{4}$$

B. Mapping and Deliberative Control

We model the robot's deliberative layer as a regularly updating discrete-timed event system which updates at times $t_0, t_1, \ldots, t_{final}$, where $t_i - t_{i-1} = \Delta t$, $\Delta t > 0$. These intervals account for the practical requirements of the execution of complex algorithms and management of large data sets.

Given that the robot is using a map to guide its path planning algorithms, we define the map M as an integrated set of sensory input. In each update cycle, the most recent set of sensory input, S is integrated into the map by the integration function m,

$$M_t = m(s_t, y_t, M_{t-1}) \tag{5}$$

We assume m is a non-invertable function. That is, given M_t we cannot directly recover $\{(s_0,y_0),\ (s_1,y_1)\ \ldots,\ (s_t,y_t)\}$. This is an important point, as given M_t, we cannot directly recover the reactive layer's control output, $u(y_t,s_t)$

Given that the world W is compact and connected, assume W is partitioned into a set of n regions,

$$\boldsymbol{R} = \{r_i\}_{i=1}^{n} \tag{6}$$

such that

$$\bigcup_{i=0}^{n} r_i = W \tag{7}$$

and

$$r_i^o \bigcap r_j^o = \phi,\ \forall i,j\ i \neq j \tag{8}$$

where r_i^o denotes an interior region.

For each region, we are given a collection of m control laws,

$$G_{r_i^o} = \{g_j(y,s)\}_{j=0}^{m},\ \forall r^o \in R^o \tag{9}$$

and a transition function $d(r^o, M, g_{r^o})$ which provides a mapping from an interior region and a control law to the next region the control law will drive the robot toward. Intuitively, we can think of this is as the expected outcome of the control law. This mapping is important to the planning process as it provides a model of the outcome of the action of employing a particular control law. We are also given a cost model, $c(r^o, M, g_{r^o})$ that provides an expected cost of traversing region r^o, using the control law g, given the map M.

Given a goal region, r_{goal}, and an starting region, r_{start}, this representation is easily mapped into a graph-based model (compatible with many planning algorithms), $\Gamma = (V, E, l, v_{start}, v_{goal})$, where:

- V is a set of vertices, directly corresponding to the set of interior regions, $\{r^o\}_{r^o \in R^o}$
- E is a set of directed edges, $E \subset V \times V$. This set of edges corresponds to the connectivity described by the transition model, $E = \{r^o \times d(r^o, M, g_{r^o})\}_{r^o \in R^o,\ g_{r^o} \in G_{r^o}}$
- l is a cost function $l : E \rightarrow \mathbb{R}^+$ directly corresponding to the cost model $c(r^o, M, g_{r^o})$, where the edge corresponding to the weight is given by the transition model, $E = (r^o \times d(r^o, M, g_{r^o}))$.
- v_{start} and v_{goal} are the starting and goal vertices, respectively. These vertices correspond directly to the regions r_{start} and r_{goal}.

Within this graph-based representation, the path planning problem can be defined as selection a sequence of edges

$$Plan = \{e_0 = (V_{start}, V_1), \ldots, e_N = (V_{goal-1}, V_{goal})\} \tag{10}$$

to minimize the total cost

$$Cost = \sum_{e \in Plan} l(e) \tag{11}$$

In this case, the set of edges is provided by the transition model, $d(r^o, M, g_{r^o})$, which is a function of the selection of of the control law, g_{r^o}. We can then more precisely define the planning problem as choosing a mapping $b \in B$ (where B is the set of all possible mappings), from each r_i^o to a $g_{r^o} \in G_{r_i^o}$,

$$\dot{x} = f(x, g_{r^o}(y,s)),\ \forall x \mid p(x) \in r_i^o,\ g_{r^o} = b(r_i^o) \tag{12}$$

(where $p(x) \in W$ is the position of the system) , such that

$$b = \arg\min_{b \in B} \sum_{i=0}^{goal} c(r_i^o, M, b(r^o)) \tag{13}$$

where

$$r_{i+1}^o = d(r_i^o, M, b(r_i^o)) \tag{14}$$

C. Learning

Two components of the deliberative layer's planning process rely primarily on a priori models of the reactive layer's execution of the provided plans. The transition model, $d(r^o, M, g_{r^o})$ predicts the the next region the system will enter, given the region the robot is currently in and the control law the robot is currently executing. The cost model $c(r^o, M, g_{r^o})$ predicts the cost incurred by the system until the next region is reached. It is the goal of this work to improve the integration of the deliberative and reactive layers by learning a cost model that better represents the cost actually incurred by the reactive execution of the plan. Improving performance by learning the transition model as well will be discussed further in the Future Work section.

We begin by defining a measurement function, $m_c(x, r^o)$ which measures the cost incurred by the execution of control law g in region r^o given map M. We define the learning problem as choosing a cost model that best predicts the measured cost,

$$c = \arg\min_{c \in C} E[\ |m_c(x, r^o) - c(r^o, M, g_{r^o})|\ ] \tag{15}$$

where C is the set of all possible cost functions. This optimization is over a expectation not only because of possible noise in the sensory information, but because, as noted earlier, the mapping function is non-invertable. Therefore, the cost model, which is a function of M, cannot directly access the reactive output measured by the measurement function. The best the cost model can do is a prediction of the reactive output. Our goal is to minimize the error in this prediction.

IV. IMPLEMENTATION

To demonstrate the capabilities and performance of the proposed system, a simulated robot and hybrid control architecture was implemented. A car-like robot was implemented in the Gazebo simulation environment [10]. The robot's physical state is represented as simply its 2-dimensional position and heading,

$$x = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{16}$$

The robot has control over its translational velocity, v and its steering angle, which is proportional to the curvature of its path, κ,

$$u = \begin{bmatrix} \kappa \\ v \end{bmatrix} \tag{17}$$

Thus, the dynamics of the robot are defined,

$$\dot{x} = \begin{bmatrix} v \cdot \sin\theta \\ v \cdot \cos\theta \\ v \cdot \kappa \end{bmatrix} \tag{18}$$

The simulated robot is equipped with sensors to measure its own state, and the state of the world. A simulated GPS module provides the robot with a measurement of its own state,

$$y = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{19}$$

Simulated laser range-finders provide measurements of the state of the world,

$$s = \begin{cases} o(w) & \text{if } \|p(x) - w\| \leq \Delta \\ \phi & \text{otherwise} \end{cases} \tag{20}$$

where Δ is the range of the measurement system, and $o(w)$ is the *occupancy* of the point $w \in W$. The occupancy of a point w is defined as:

$$o(w) = \begin{cases} 1 & \text{if the point } w \text{ is occupied} \\ 0 & \text{else} \end{cases} \tag{21}$$

A. Reactive Layer

The reactive layer is implemented in a behavior-based voting design, explained in detail in [11]. In this design a number of behaviors evaluate candidate actions over a short temporal scale, each behavior representing a specific interest pertaining to the robot's objective.

In this implementation the behaviors reason over constant curvature arcs. Each behavior distributes an allocation of votes over an array of potential arcs for the robot to navigate along. The behaviors can allocate votes for arcs that work to achieve its interests, or against arcs that are detrimental to its interests. In addition to distributing votes for or against arcs, behaviors assign a maximum allowable velocity, associated with each arc.

To choose a curvature and velocity for the robot to execute, an arbiter sums the votes cast by each behavior for each curvature arc, weighting the votes for each behavior according to a predetermined weighting scheme. It selects for execution the curvature arc with the highest total of votes. It then selects for execution the minimum of the maximum allowable velocities assigned by the respective behaviors to the selected curvature arc. The selected curvature and velocity are sent on to low-level controllers for execution.

Five behaviors were used in this implementation:

- *Move to Waypoint* - allocates positive votes to arcs according to a linear control law relating the local heading to the waypoint to a commanded curvature. The votes are allocated according to a Gaussian distribution around the output of the linear control law.
- *Avoid Obstacles* - allocates negative votes to arcs according to the distance along the arc that the arc crosses into the configuration space around a detected obstacle. Arcs that do not cross into the configuration space of the obstacle are not voted against.
- *Maintain Headway* - sets maximum allowable velocities for each arc according to the distance along the arc that the arc crosses into the configuration space around a detected obstacle. If the arc does not cross into

the configuration space of the obstacle, the robot's maximum speed is assigned. If the arc crosses into the configuration space of the obstacle within a parameterized safety distance, the maximum allowable velocity is zero.

- *Slow for Congested Areas* - sets maximum allowable velocities for each arc according to the distance along the arc that the arc crosses into an intentionally large configuration space around a detected obstacle. If the arc does not cross into the the configuration space of the obstacle, the robot's maximum speed is assigned. A velocity of zero is never assigned. That responsibility is left to the maintain headway behavior.
- *Slow for Turns* - sets a maximum allowable velocity for each arc according to a parameterized maximum allowable rotational velocity. if the calculated maximum allowable velocity is larger than the robot's top speed, the robot's top speed is assigned.

In this implementation, the set of control laws G_{r^o} is provided by parameterizing the given set of behaviors with a waypoint from each adjacent region. (i.e., each member of the set of control laws drive the robot toward one of the adjacent regions, using the full compliment of behaviors.) The transition model $d(R^o, M, g_{r^o})$ is simply defined as mapping to the region associated with the waypoint parameterizing the control law g_{r^o}, regardless of the map.

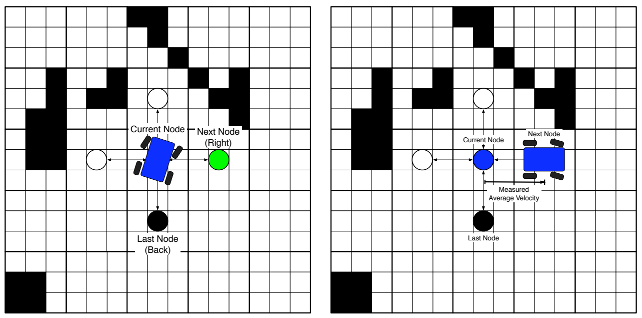
B. Deliberative Layer

The deliberative layer is implemented as a global path planner over a relatively high-resolution occupancy grid. As sensory information is accumulated in the local frame it is integrated into the global map based on the robot's current global state measurement. Detected obstacles are placed into grid cells based on their discretized global position. Each grid cell can be marked as either occupied or unoccupied. Obstacles associated with unoccupied cells cause the cell to be marked as occupied. Obstacles associated with occupied cells have no effect on the cell; the cell remains marked occupied.

The grid cells are then grouped into regions, as depicted in Figure 1. A count of occupied grid cells is kept within each region. This count is used by the planning algorithm in evaluating the cost of traversing each region.

To represent the connectivity between the regions, a graph is overlaid on the map, as shown in Figure 1. One graph vertex is placed at the center of each region. Edges are added between contiguous nodes. Figure1 depicts a four-connected graph based on the structure of the occupancy grid. The cost of traversing each edge is proportional to the expected time to move between its source node and destination node. The time to move between nodes is modeled as the distance between the nodes divided by the expected average velocity of the robot over that distance. The baseline planner uses a binary model of the robot's velocity. If the count of occupied grid cells within the region associated with either node is larger than a parameterizable count, the expected velocity is zero (i.e. the edge is not traversable and is assigned an infinite cost). Otherwise, the expected velocity is the robot's top speed. This graph structure is a suitable data structure for many planning algorithms. In this implementation, an instance of the D*-Lite [12] [13] algorithm is employed.

C. Learning



(a) Learning experience setup (b) Learning experience measurement

Fig. 1. A depiction of the encoding of the learning experience. In (a), the learning experience is setup. The robot is shown in the center of the diagram. The next planned waypoint is represented by the graph vertex to the right of the robot. The local map, out to the range of the robot's sensors is included under the graph. In (b), the robot has completed the experience. The supervisory signal (in this case, average velocity) is measured and provided to the learning algorithm, along with the experience representation.

The learning component of this approach is implemented within a supervised learning paradigm. To keep the learning problem tractable, it is important to create a compact, yet meaningful representation of the robot's experiences executing proposed planning segments. We define the length of a learning experience to be time to move from one region, r_i^o to the next region in the plan, $d(r_i^o, M, g_{r^o})$, given the control law g_{r^o} provided by the planning algorithm. We use the following representation of a learning experience:

$$Exp = (g_{r^o}, M_{local}) \tag{22}$$

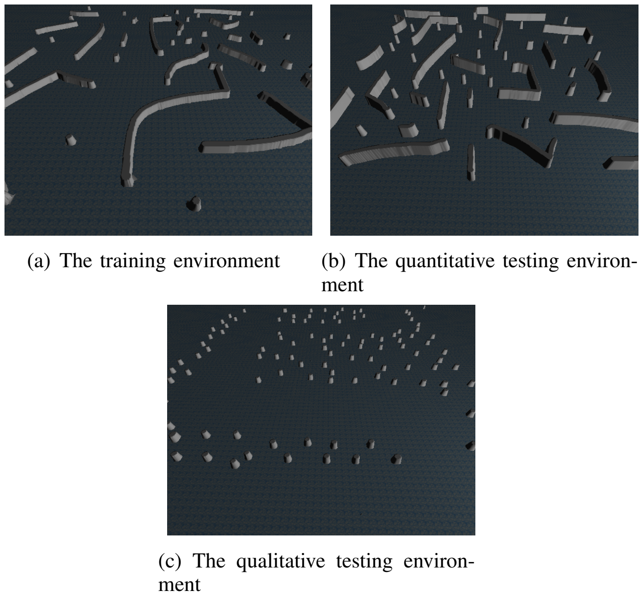
where M_{local} is a local representation of the map, M. To take advantage of symmetry in the problem, we orient the experience into the robot's local frame. That is, rather than encoding the segments of the plan as "move north" or "move east", it is more general to encode the robot's experiences as "move forward" or "move right". A more general encoding of experiences makes learning over these experiences more tractable, as it reduces the dimensionality of the problem. Figure 1 is a graphical representation of the robot's planning experience.

A supervisory signal is provided by the measurement function $m_c(x, r^o)$, which measures the reactive layer's interpretation of the commanded plan. As shown in Figure 1(b), the measurement function measures the average speed of the robot during the experience.

Once a sufficient number of planning experiences have been recorded, the experiences are used as data for a supervised learning algorithm. The supervised learner uses the experiences to extrapolate expected results from new proposed experiences. In this implementation, the learned model of expected velocity is used by the global planner to plan subsequent navigation paths. The planner uses the model to evaluate the expected cost of traversing an edge, in terms of expected time to traverse the edge. To evaluate the cost of an edge, the edge is encoded in terms of a planning experience. The learned velocity model returns an expected velocity over the edge. The time-based cost model is obtained by dividing the distance between the source node and the destination node by the expected velocity. The planning algorithm plans over these costs to find the fastest route.

V. EXPERIMENTAL SETUP AND RESULTS

A. *Experimental Setup*


(a) The training environment (b) The quantitative testing environment

(c) The qualitative testing environment

Fig. 2. The environments used in training and testing. The environments were built in the Gazebo simulation environment. Environment (a) was used in gathering training data for the learning process. Environment (b) represents a slightly more complex environment than (a), and was used for quantitative testing. (c) represents a plausible environment, consisting of a path through a dense forest, and was used for qualitative testing.

Two complex environments were designed within the Gazebo simulation environment. The environments used are shown in Figure 2. Environment 2(a) was used for gathering training data. Environment 2(b) was used for running tests comparing different systems. Environment 2(c) was used to demonstrate the qualitative behavior of the system.

Data was gathered in the training environment by tasking the robot to achieve a series of randomly generated goals around the environment, using the baseline planner and the above described reactive layer. Every time the robot achieved a waypoint the robot's experience was recorded, including the local map, the robot's average velocity, the commanded waypoint, and the waypoint actually achieved. Approximately 5000 learning examples were collected.

Several different supervised learning algorithms were evaluated for use. The Weka machine learning environment [14] provided a library of community-supported implementations of well known algorithms. For learning prediction of the robot's velocity (a real-valued signal), we evaluated the k-nearest neighbor algorithm for several values of k, a multi-layer perceptron network, linear regression, and the baseline strategy of always assuming the robot travels at its maximum speed.

	Simulated Time to Complete Test	Simulated Distance Travelled	Relative Improvement over Baseline System	Paried T-Test p-Value
Baseline System	7745 sec	5124 m	N/A	N/A
Learned System (KNN $k=5$)	6434 sec	5577 m	17%	.02

TABLE I
RESULTS OF TIMED TEST OF BASELINE AND LEARNED SYSTEMS. THE LEARNED SYSTEM IMPROVED PERFORMANCE OVER THE BASELINE BY 17%, WITH A STATISTICALLY SIGNIFICANT p-VALUE OF .02.

Models were built from the training data, using each algorithm. Cross-validation tests on the training data were performed to evaluate the effectiveness of each algorithm. The k-nearest neighbor algorithm with $k = 5$ produced the lowest average relative error and highest correlation of the algorithms tested, and was chosen for use in testing.

The learned models were then incorporated into the cost function of the global planner. The baseline system was compared to the system using the learned model. Each system was tasked with achieving a sequence of goals criss-crossing the test environment. This sequence of goals totaled a piecewise straight-line distance of over 1500 simulated meters. Results were compiled comparing the average time to complete each goal between different systems.

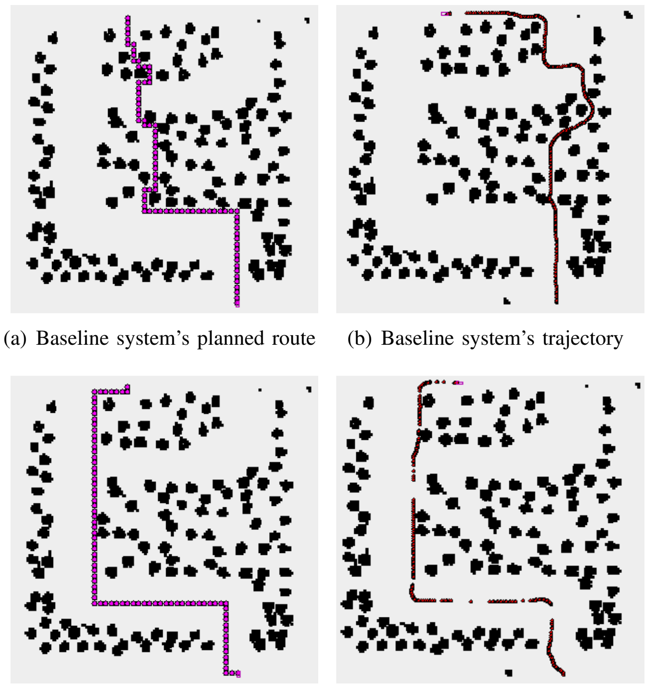
In addition to quantitative experiments, a qualitative experiment was performed to demonstrate, in an intuitive way, the effect learning had on the overall system performance. An environment was constructed to resemble an open path through a wooded area, shown in Figure 2(c). The wooded area is sparse enough that the robot is capable of finding a path between the trees, but would travel that path slowly due to its tendency to drive slowly in tight spaces and slow down for the frequent required turns. The robot was tasked with navigating to a goal whose straight-line path would take the robot through the woods. The plans and resulting paths created by the baseline system and the system that had learned a cost model were compared qualitatively and quantitatively.

B. *Quantitative Results*

Table I displays the results of using the learned models in the global planner on the time to complete the sequence of goals criss-crossing the environment shown in 2(b). Using the learned velocity model reduced the time to complete the overall mission by 17% over the baseline system. The calculated p-value indicates that the result is statistically significant.

The fact that the learned model does show improvement indicates that using supervised learning to improve the deliberative layer's model of how its plans are interpreted by the reactive layer can improve the robot's overall system performance.

C. Qualitative Results


(a) Baseline system's planned route (b) Baseline system's trajectory

(c) Learned system's planned route (d) Learned system's trajectory

Fig. 3. A comparison of the plans and trajectories produced by the baseline and learned systems in the "path in the woods" environment.

Figure 3 shows the results of the qualitative tests in the path through the woods environment. Figure 3(a) shows the baseline planner's planned route through the environment. Note how the planned route snakes through the dense obstacle field on its way to the goal. Figure 3(b) shows the trajectory actually taken by the robot following the planner's output. Note that it departs from the planned route early in the mission. The planner continues to suggest updated routes based on the robot's position, and the robot eventually achieves the goal.

Figure 3(c) shows the route provided by the planner using both the velocity and transition models. Note that it prefers a slightly longer (by distance) route that follows the wide path. Figure 3(d) shows the trajectory actually taken by the robot following this plan. In this trial, the robot completes the mission in 25% less time than the baseline planner. This demonstrates a clear qualitative and quantitative improvement in system-level performance in a plausible environment.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a novel approach to the problem of improving system-level performance of a hybrid deliberative-reactive control architecture for robotic navigation. In particular, we propose using machine learning techniques to improve the deliberative layer's cost model, based on measured performance of the reactive layer's execution of plans. The system was implemented in physics-based simulation environment. Quantitative and qualitative experimental results were compiled and presented.

While, this work concentrated on batch-learning of the cost model, certainly more work in the area can be done. For example:

- It is not yet clear how map representation effects the performance of the learning component of the approach.
- Incremental learning may be desirable in many applications of robotic navigation. How does this approach handle incremental improvements "on the fly"?
- While improving the planner's cost model certainly has the potential to improve overall system performance, judging from the trajectory in Figure 3(d), it is apparent that improving the transition model may also have a significant effect on system performance.

We look forward to exploring these questions in future work.

REFERENCES

[1] J. S. Albus, "4d/rcs: a reference model architecture for intelligent unmanned ground vehicles," in *In Proceedings of SPIE Aerosense Conference*, pp. 1–5, 2002.

[2] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, *et al.*, "The robot that won the darpa grand challenge," *Journal of Field Robotics*, vol. 23, pp. 661–692, 2006.

[3] C. Urmson, J. Anhalt, H. Bae, J. A. Bagnell, *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I*, vol. 25, pp. 425–466, June 2008.

[4] R. C. Arkin and T. Balch, "Aura: Principles and practice in review," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, pp. 175–189, 1997.

[5] E. Gat, "Integrating planning and reacting in a heterogeneous asynchronous architecture for mobile robots," *SIGART Bulletin*, vol. 2, no. 4, pp. 70–74, 1991.

[6] L.-J. Lin, "Hierarchical learning of robot skills by reinforcement," in *International Conference on Neural Networks*, 1993.

[7] R. Sutton and A. Barto, *Reinforcement Learning, an Introduction*. The MIT Press, 1998.

[8] P. Stone, *Layered Learning in Multi-Agent Systems*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1998.

[9] T. Balch, *Behavioral Diversity in Learning Robot Teams*. PhD thesis, Georgia Institute of Technology, December 1998.

[10] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *In Proceedings of the 11th International Conference on Advanced Robotics*, pp. 317–323, 2003.

[11] D. Wooden, M. Powers, M. Egerstedt, H. Christensen, and T. Balch, "A modular, hybrid system architecture for autonomous, urban driving," *Journal of Aerospace Computing, Information, and Communication*, vol. 4, pp. 1047–1058, December 2007.

[12] S. Koenig and M. Likhachev, "D*-lite," in *National Conference on Artificial Intelligence*, pp. 476–483, 2002.

[13] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, pp. 354–363, June 2005.

[14] E. Frank, M. A. Hall, G. Holmes, R. Kirkby, B. Pfahringer, I. H. Witten, and L. Trigg, "Weka - a machine learning workbench for data mining," in *The Data Mining and Knowledge Discovery Handbook* (O. Maimon and L. Rokach, eds.), pp. 1305–1314, Springer, 2005.