Evaluating Robustness in a Two Layer Simulated Robot Architecture

Larry D. Pyeatt*
Computer Science Department
Texas Tech University
Lubbock, TX 79409
email: larry.pyeatt@ttu.edu

Adele E. Howe
Computer Science Department
Colorado State University
Fort Collins, CO 80523
email: howe@cs.colostate.edu

Keywords: Robot Navigation, Reinforcement Learning, Partially Observable Markov Decision Process

Abstract

Many two layer robot architectures have been proposed and implemented. While justification for the design can be well argued, how do we know it is really a good idea? In this paper, we describe a two layer architecture (reinforcement learning in the bottom layer and POMDP planning at the top) for a simulated robot and summarize a set of three experiments in which we evaluated the design. To address the many difficulties of evaluating robot architectures, we advocate an experimental approach in which design criteria are elucidated and then form the basis for the evaluation experiments. In our case, we test the implementation for its reliability and generalization (our design criteria) by comparing our architecture to one in which a key component is substituted; in these experiments, we demonstrate significant performance gains on the design criteria for our architecture.

1 Evaluation of Robot Architectures

Two layer architectures are common for robot control. The lower layer provides fast, short horizon decisions about quickly executed actions based on sensory data, while the upper layer adopts a more goal oriented view and plans over a longer scope using information abstracted from the sensory data. Components of the layers usually differ between architectures as do the mechanisms for communicating state information and coordinating activity. Although many architectures have been proposed and implemented, few have been compared to alternatives.

Barriers to empirical comparisons of architectures are myriad and daunting, especially for physical robots. Architectures are implemented on different platforms that are often incompatible. Replicability is difficult to achieve, and the multiple trials required for statistical significance are

*The work was done while Larry Pyeatt was at Colorado State University.

simply impractical or infeasible. The design criteria may be quite different as well, leading to potentially unfair conclusions that "my architecture is better than yours because it was designed to do something yours wasn't."

Design criteria may hold the key, however, to understanding the contribution of new architectures. Even if one's architecture (or to be precise, instantiation of an architecture) cannot easily be compared to architectures developed in other laboratories, ablation, substitution, or environment manipulation experiments can be performed on a particular architecture to determine whether it lives up to its designer's expectations or even promises. Given information on how well a particular architecture performs on specific criteria, higher level comparisons across architectures may be possible on the same criteria.

We have been developing an architecture for autonomous control of a simulated robot (the Khepera robot) in a relatively benign environment (a maze meant to represent an office navigation environment). To build our architecture, we started with a set of three design criteria, and we have motivated our design/implementation and tested it along the way for how well it achieves those criteria. In this paper, we will state those criteria, describe the architecture that resulted, and show how we demonstrated our architecture's performance on the first two design criteria.

2 Design Criteria for the Architecture

Our goal was to develop a robust control system for an autonomous robot that can operate in an office-like environment. The environment is expected to be well understood and not overly hostile. The primary task of the robot is navigation between two instructed points through an environment that is mostly predictable (e.g., we don't expect walls to move often, but other obstacles might). The task requires some planning, but does not demand a large knowledge base and deliberative planning (i.e., it does not have a large suite of available parameterizable actions, and the cost of taking the wrong action is relatively low). However, we do expect that the robot will not always function perfectly and that it should be able to cope with significant degradations in performance without requiring operator intervention.

We focused on three design criteria: reliability, generalization, and adaptability. Reliability means that the architecture can still achieve its designated tasks (albeit possibly with some performance reduction in time, cost, or accuracy) even when core functionality is degraded; reliability can be evaluated by manipulating some function (e.g., sensors, effectors, communication, etc.) and measuring the effect on task achievement and performance metrics. Generalization is the ability of the agent to act appropriately (although not necessarily optimally) in situations that it has not encountered before; generalization is assessed, as in machine learning, by separating training and testing and demonstrating performance in new situations. Adaptability goes a step further than generalization by requiring that the agent modify its behavior to perform better in new situations; this criterion can be demonstrated by starting the agent with minimal capabilities and assessing changes to internal structure and external performance over both time and changing conditions.

Our design criteria led us to a two layer architecture in which learning or tuning was integral. The bottom layer was responsible for sensing and effector control; the top level was responsible for deciding on the sequence of actions needed to achieve the goals.

In previous papers, we described and evaluated the bottom layer (Pyeatt and Howe 1998b; Pyeatt and Howe 1998c). We had settled on using reinforcement learning for the bottom layer, but had some choice in how to implement it. In particular, we tested generalization of the coarsest aspect of the low level design: whether to use a monolithic or a distributed representation of separate behaviors (Pyeatt and Howe 1998c). We tested in a simplified environment (Robot Auto Race Simulator (Timin 1995)) where performance is measured in terms of lap-time and number of crashes. We found that the distributed representation provided the best generalization and reliability by assessing overall performance after learning and the time required for performance to level out after being presented with new situations.

To test adaptability of the bottom layer, we required a new behavior, passing other cars, and assessed how the overall performance changed to accommodate the new requirements (Pyeatt and Howe 1998d). We found that the system quickly learned to accommodate the new behavior (within 20 laps) and achieved better overall performance than the previous version had.

The top layer sequences the behaviors of the bottom layer. Consequently, it was difficult to test the design criteria of the top layer independent of the bottom layer. We expected reliability and adaptability to be an outcome of these properties in the lower layer and the POMDP policy computation, but really could not determine it until we tested the architecture as a whole. As it turned out, the primary design consideration for the POMDP layer itself was the computation required to calculate the policy. Thus, our evaluation of this layer focused on demonstrating computational gains through a distributed implementation of a well known POMDP algorithm (Pyeatt and Howe 1999).

3 Architecture

In our two layer, simulated robot architecture, the lower layer (the *execution* layer) controls the actuators that move the robot around and provides a set of actions that can be used by the higher level planning system. The higher layer (the *planning* layer) plans by finding a policy that specifies what action to take in every possible situation in order to move the robot from its current location to the goal. The bottom layer is based on reinforcement learning (RL), while the top layer is based on a partially observable Markov decision process (POMDP).

Figure 1 provides a detailed look at the architecture. The planning layer sends action commands and reward signals to the action execution layer. The execution layer sends sensor information to the planning layer. When following a policy, the planning layer maintains a belief state over the possible states. The planning layer uses the current belief state to select a low level action from the policy. The planning layer also tracks the state with the highest probability: the most likely current state. From the state transition probabilities, it is easy to find the expected next state when performing the current action in the most likely current state. When the most likely current state changes, the planning layer generates a non-zero reward for the action that is currently active.

3.1 Control Loops

The robot control architecture is based on three hierarchical control loops, each running at different time scales and different levels of abstraction.

- The short term control loop consists of the action execution module, effectors, environment, and sensors. Control decisions are made in the action execution module by the action (RL module) that is currently being executed. It produces control signals that are sent directly to the effectors and makes decisions based on raw data from the sensors. This control loop provides fast response and tight coupling of the robot and its environment.

- The medium term control loop includes policy execution, action execution, effectors, environment, sensors, pre-processing, and belief state maintenance. Control decisions are made in the policy execution module according to the POMDP policy that is currently being executed.

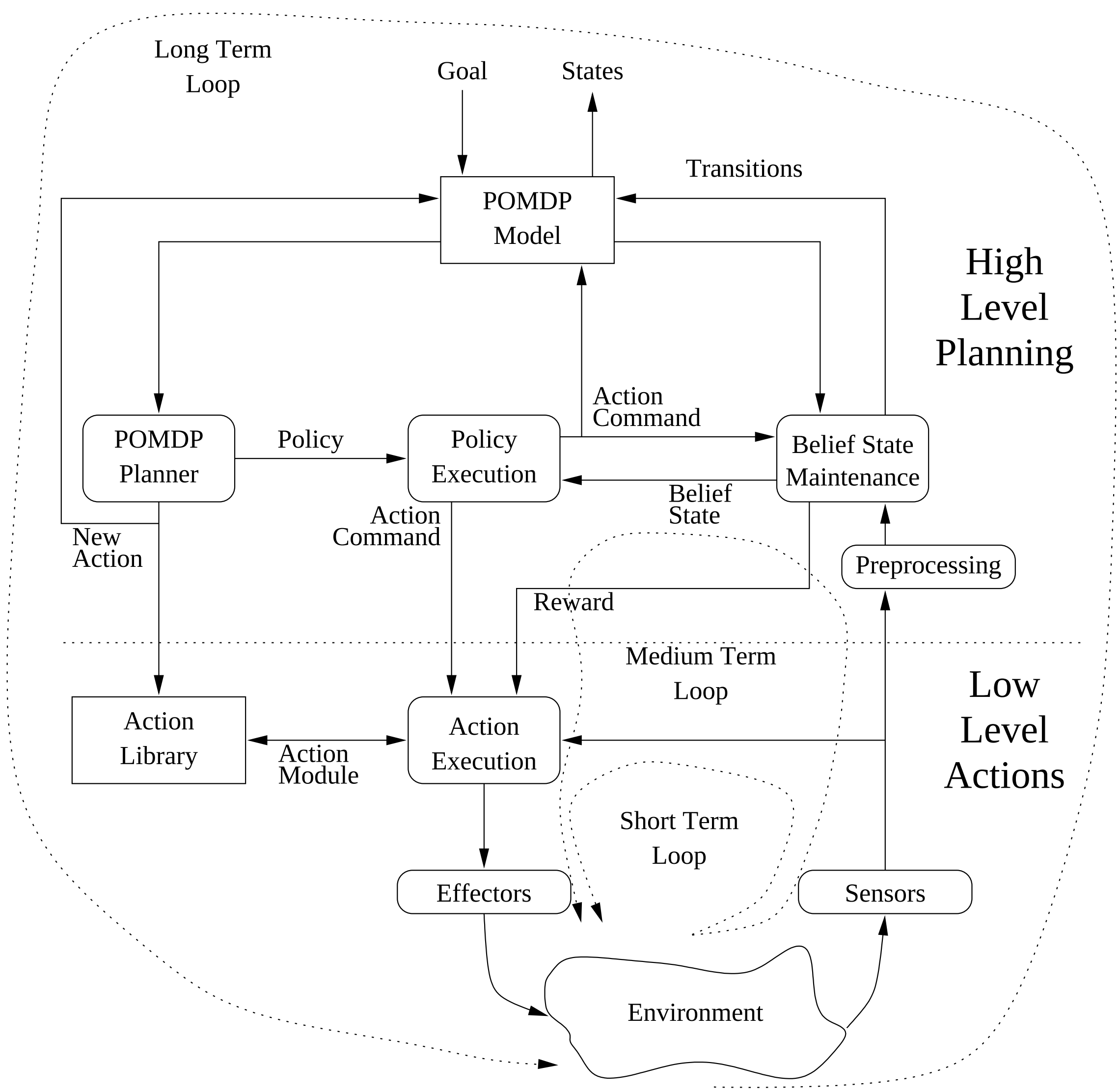


Figure 1: System overview.

It produces control signals in the form of action commands that are executed by the action executor. The decisions made in the medium term control loop are based on the belief state, which is derived from the pre-processed sensor information combined with knowledge about the world in the form of transition and observation probabilities.

- The long term control loop includes the human investigator, POMDP model, POMDP planner, policy executor, effectors, environment, sensors, and belief state maintenance. Control decisions at this level are made by the human investigator, who may specify goals as immediate rewards in the POMDP model. For instance, the investigator may specify that the reward for state x is one and the reward for all other states is zero. The POMDP planner generates a policy from the POMDP model and sends the resulting control policy to the policy executor.

3.2 Low Level Actions

The lower layer implements actions that are executed over a short time period of a few seconds. In our previous experiments, we found that subdividing activity into separate actions expedited initial learning, long term performance, and extending capabilities. Thus, the actions take the form of a set of RL modules that have been or are being trained to achieve particular state transitions. The action modules change the world state by sending control signals to the effectors. Each action module has no notion of what it is supposed to accomplish; its only goal is to maximize the reward received from the POMDP planner, which determines the necessary state transition.

Each of the learned actions has its own RL module based on Q-learning with a decision tree function approximation method (Pyeatt and Howe 1998a). Only one of the actions can be executing at any given time. When it is executing, an action module gains complete control of the actuators and has full access to sensor data. All of the RL modules are implemented identically, but each learns to perform different actions due to the rewards it receives.

After an action has been executed, the action executor must store the updated action back in the library for future use. The action executor can be viewed as a shell that passes information and rewards into the action module that is currently loaded and then passes control signals out to the effectors.

The learned low level actions are not trained to avoid obstacles but learn to do so because it is necessary for maximizing the reward given by the POMDP planner. Suppose the policy for belief

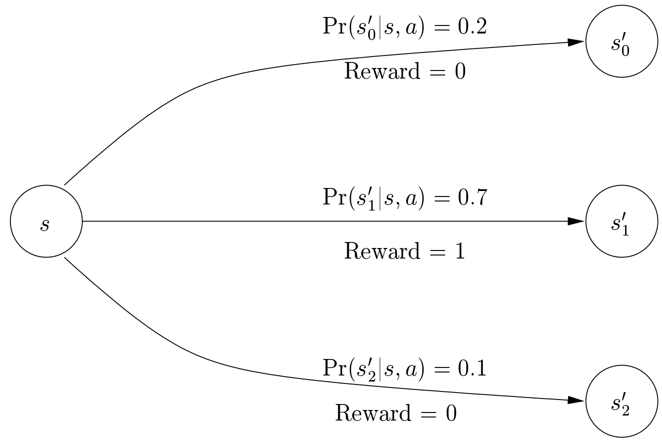


Figure 2: Rewards are assigned according to the most likely next state. In this case, the most likely state to follow s given action a is s'_1. When the most likely state changes, a reward of one is produced only if the new most likely state is s'_1.

state b with most likely state s is to perform action a and the expected next state is s'_1, as shown in Figure 2. (Equation 1 in the next subsection explains how the most likely state is calculated.) Upon entering belief state b, the POMDP planner activates action a and waits for a change in the robot's most likely state. If a drives the robot into a wall, then it will not receive a positive reward. The only way it can receive a positive reward is to cause the robot's most likely state to change to s'_1. Action a will receive zero reward until the robot's most likely state changes or until it runs out of time. The time limit forces the action to avoid a periodic negative reward. The time limit is somewhat domain dependent and should be longer than the expected time for any action to perform a state transition. When the time limit expires or a state transition occurs, the action will receive a positive reward only if the most likely state is s'_1.

Action modules may be learned through RL or hand-coded actions written by a human programmer. Hand-coded actions may be included for purposes of experimentation or to help boot-strap learning of other actions. In the case of hand-coded actions, the reward signals are ignored.

The POMDP model describes what each action does in terms of the action-dependent state transition probabilities $\Pr(s'|s,a)$ and observation probabilities $\Pr(z|s',a)$. These probabilities effectively represent the state transition(s) that can be expected when executing a given action a in a given state s, and therefore describe the effects of that action. The POMDP planner uses these probabilities to generate a policy. This results in a natural division between the description of an action, used by the top layer, and the implementation of that action, used by the bottom layer.

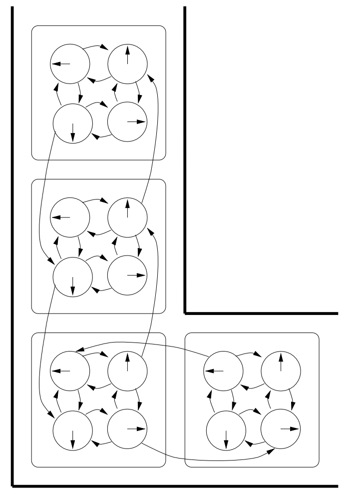


Figure 3: State map for a small section of corridor. All orientations within each position are shown.

3.3 Planning

The planning level is responsible for long term decisions and goal attainment. The human investigator can specify a goal state from the set in the POMDP model, leaving the attainment to the robot. The states in the POMDP model correspond to nodes in a topological map of the environment. The map represents each possible position and orientation for the robot, along with the state transitions that are possible from that state based on environmental constraints. An example state diagram for a corridor section with a turn in it is shown in Figure 3. In this figure, the boxes represent a single position with four possible orientations. The POMDP state space in this example has 16 states.

The POMDP planner constructs a policy that, when followed, will cause the robot to visit a chain of states from the current state (position and orientation) to the goal state. As long as the POMDP map accurately reflects the world, the actions themselves will never be required to perform in an inappropriate situation. For instance, the value of the "move forward" action in states where the robot is facing a wall is very low.

The planning level must determine when a state transition has occurred and generate rewards for the low level actions. The POMDP world model consists of a COMDP model (Completely Observable MDP) plus a set of observation probabilities that depend on the state and action. Low level actions are meant to achieve state transitions in the underlying COMDP. However, the planning level maintains only a belief state; so some mapping from belief states to the underlying COMDP states is needed. For a given belief state b, the *most likely state* is given by

$$s^b = \underset{s \in \mathcal{S}}{\text{argmax}}(b_s), \tag{1}$$

which simply selects the state with the highest probability in b. The belief state maintenance module detects state transitions by comparing the most likely state s_t^b from Equation 1 at each time step with the most likely state s_{t-1}^b on the previous time step. If $s_t^b \neq s_{t-1}^b$, then a state transition has occurred.

With this method of detecting state transitions, it is possible to generate rewards for the low level actions. The *expected state* when a is executed in b is given by

$$s^e = \underset{s' \in \mathcal{S}, s' \neq s^b}{\text{argmax}} \left(\Pr(s'|s^b, a) \right). \tag{2}$$

Whenever $s_t^b = s_{t-1}^b$, a reward of 0 is sent to the action execution module; otherwise, the reward is calculated by comparing the current most likely state s_t^b and the expected state at the previous time step s_{t-1}^e. If $s_t^b = s_{t-1}^e$, a reward of 1 is sent to the action execution module. Otherwise, a reward of -1 is sent.

The belief state maintenance module also maintains a counter that is reset to zero every time a state transition is detected. Whenever the counter passes some threshold τ, a reward of -1 is sent to the behavior execution module and the timer is reset to zero. τ is chosen so that the action is given ample time to achieve the desired state transition. In our experiments, τ was set at 100; a properly functioning action typically required only 10 steps to achieve a transition. A minimum value for τ can be found for most domains from the speed of effectors, length of the time step, and distances between states in the POMDP model.

To simplify the POMDP and make it solvable, the sensory input is pre-processed to provide a small number of observations to the belief state maintenance module. This may result in some occasional discrepancies between the most likely state derived from the belief state and the actual state, but this is generally not a problem, since discrepancies rarely occur when the sensor pre-processing is sufficient. Empirical tests on the architecture showed that the most likely state matches the current state more than 90% of the time. Additionally, reinforcement learning is robust enough to converge even with occasional erroneous rewards (Sutton 1988).

Whenever the human investigator changes the goal state in the POMDP model to one that hasn't been tried before, the POMDP planner must be run to find a new policy. Finding an optimal policy is time consuming, but once the policy is found for a goal state, that policy is saved for future use. The policy consists of a set of labeled $|\mathcal{S}|$ vectors. The label on a vector tells what action to perform if that vector has the largest dot product with the current belief state. Policies were generated using a parallel version of the restricted region algorithm (Pyeatt and Howe 1999).

The architecture was also designed to decide when to learn new actions. As that capability is not central to the robot control function, we do not describe it in this document, but it is described in (Pyeatt 1999).

4 Related Work

Because the architecture coordinates reinforcement learning and POMDP planning (one for each layer), our work resembles other projects that either use only one of the technologies or use two layers. Thus, the first three subsections of this section summarize the most similar approaches to reinforcement learning, POMDP planning, and two layer approaches for robotics. The final subsection briefly discusses successful evaluation strategies for robot architectures.

4.1 Reinforcement Learning for Robotics

Reinforcement learning (RL) requires little intervention or knowledge to learn. It excels at control tasks in which both the value of action and the appropriateness in particular situations needs to be determined. Thus, RL has been applied in several robot projects to provide low level control of sensors and effectors.

Mahadevan and Connell (1992) used RL for automatic programming of behavior based robots. They identified three basic behaviors and trained their system to perform them. They found that the RL techniques are capable of learning individual behaviors that sometimes outperform hand-coded solutions. They concluded that using a behavior based architecture speeds up reinforcement learning by converting the problem of learning a complex task into that of learning a simpler set of special purpose reactive subtasks.

Dorigo and Colombetti (1994) trained a robot to perform a predefined target behavior using RL. Their goal was to create a robot capable of surviving in a real environment. They trained the robot to display five different behaviors and then defined four ways to combine these behaviors. Their approach differed slightly from most RL strategies. Instead of forcing the robot to learn with no initial knowledge, they gave the robot some basic responses and then used reinforcement learning to improve the robot's knowledge. Much of the initial training was done on a simulator; the results were successfully transferred to a physical robot.

There are several other examples of reinforcement learning used for robot control. Wyatt (1995) discusses some of the issues involved in using RL for robot control. Bagnell, Doty, and Arroyo (1998) performs a comparison of various RL techniques for robot behaviors. Kontoravdis, Likas, and Stafylopatis (1992) presents an RL based controller for autonomous vehicles. Most approaches to date have relied on the investigator to provide the reinforcement signal. In our approach, the reinforcement signal is generated by the higher level control system.

4.2 POMDP Planning for Robotics

POMDP planning can be viewed as the task of finding a *policy* for a sequential decision task (Zhang and Liu 1996; Cassandra 1998b). The policy is a mapping from states to actions that maximizes some reward function. For plan execution in this framework, the agent determines its "current state" and chooses an action from its policy at each time step. The effects of actions need not be deterministic. The current state is estimated from a vector of sensor observations that substitute for knowing the true state. Thus, a POMDP formulation of the planning problem is particularly well suited to mobile robotics, where the environment can be modeled as a state space, but the sensors may be inadequate to determine the current state.

Cassandra et al. (1996) used a variant on POMDP planning in a robot navigation task. In their approach, the policy was based on a completely observable MDP model of the environment. However, to execute the policy, the robot maintains its POMDP belief state b, which can be related to the most likely state and then used to select an action from the COMDP policy. This approach allowed their robot to navigate in an environment that was too large for exact POMDP methods while still providing some of the benefits of the POMDP model.

One highly visible success in POMDP planning for robotics is the Xavier robot at Carnegie Mellon University (Simmons and Koenig 1995; Koenig, Goodwin, and Simmons 1996; Simmons, Goodwin, Haigh, Koenig, and O'Sullivan 1997). Xavier uses POMDP planning to navigate in an office environment. Tasks are specified by telling the robot where to go using verbal commands or a World Wide Web interface. Like Cassandra et al.'s approach, Xavier does not try to create a POMDP policy directly. Instead, it finds a policy for a COMDP but maintains a POMDP belief state to determine its current location.

Although we believe that the work listed above is closest in spirit to our POMDP approach, there are many other possible variations on the theme, and this is far from a complete listing. Additional work has been performed by Roy (1998), Mahadevan (1997), and others.

4.3 Two Layer Architectures

The two architectures that ours most closely resemble are CQ-L and CHILD. In CQ-L, the low level or *elemental* behaviors use reinforcement learning, and a task-sensitive gating module learns to compose the appropriate elemental modules over time in order to achieve long term goal (Singh 1991; Singh 1992). In this approach, all elemental behaviors are Markov Decision Tasks (MDTs) that share the same state set $\mathcal{S}$, action set $\mathcal{A}$, and the same environment dynamics. Each elemental task has its own reward function based on the desired final state. A *composite* task is rewarded when one of the subtasks reaches its terminal state. This architecture was tested in a simulated robot navigation environment and was able to solve some composite tasks, such as moving from one point to another and then proceeding to a third point. This approach shows that reinforcement learning of low level behaviors can be made to work when the exact behaviors are specified, but it does not address the issue of higher level planning and is not taskable.

Continual, Hierarchical, Incremental Learning and Development (CHILD) (Ring 1994) combines a neural network architecture (Temporal Transition Hierarchies) with Q-learning. At each step, the robot's sensory input from the current state is given to the transition hierarchy network which produces a value for each possible action. The network's response at the previous time step is modified according to the maximum action value from the current time step. This system is designed to learn continually and adapt to new situations and environments. New hierarchies are added for each skill or behavior, and new behaviors can be constructed from old ones by building onto extant hierarchies. This allows the robot to encapsulate skills from early learning into a foundation for later learning.

CHILD has been shown to work well for navigating a maze-like environment where the inputs at each time step may not uniquely determine the current state (Ring 1993; Ring 1994). It provides low level learning and adaptation but does not perform long term planning and cannot pursue dynamically assigned goals.

4.4 Evaluation Strategies

The two most prominent types of evaluation for robots that include planning are comparative evaluation and hypothesis testing. Comparative evaluation addresses the question: "whose approach is best?" The benefit is that the differences between approaches are clarified; however, the cost of a side by side comparison under controlled conditions can be high. Differences in hardware and software need to be sorted out. The evaluation criteria must be measurable and fair to the systems being compared.

While robot competitions are popular comparisons and showcase different approaches to the same problems (Arkin, Noda, Suzuki, Matsubara, Asada, and Kitano 1998), they tend not to support controlled study and understanding of the benefits of differing approaches. Benchmark comparisons examine performance of a few systems on many well known problems from the literature. However, benchmark comparisons are rare in robotics because benchmark problems must be easily represented/communicated and well constrained, and few problems have been generalized or popularized. A notable exception is in testing specific components. For example, the POMDP planners for navigation (e.g., (Koenig and Simmons 1998)) tend to be evaluated on benchmark problems before being incorporated in full systems.

Hypothesis testing is usually performed by an individual researcher to determine the influence of controlled features of the environment or system on performance. For example, Cassandra et al. (1996) tested the influence of starting position and noise on performance of four heuristic control strategies for POMDP planning. To make the study tractable, they restricted the characteristics they examined. From the resulting analysis, they were able to make recommendations about which strategies to use when.

Statistical hypothesis testing requires multiple instances over which to generalize results. Performing repeated trials on physical robots is time and resource consuming, and may require considerable direct control of the environment. However, repeated trials become easier after robustness or reliability is addressed. For example, the reliability of the Xavier robot was evaluated by allowing

WWW visitors to issue commands over an extended period; at the termination of the reported experiment, Xavier had a 95% success rate on its tasks. Once the reliability was established, other experiments have been run to test aspects of its design (e.g., (Koenig and Simmons 1998; Haigh and Veloso 1997)).

One methodological approach that shows promise for overcoming some of the difficulties of robot evaluation is exploiting simulation to test specific capabilities. Several researchers have tested robot planners in simulated environments (e.g., (Cassandra, Kaelbling, and Kurien 1996; Feiten, Wienkop, Huster, and Lawitsky 1994; Mahadevan and Connell 1992; Dorigo and Colombetti 1994)) and then demonstrated that the results transferred to the physical platforms (with some caveats). The simulators free the researchers to try alternatives, perform many trials, and to test specific characteristics of the designs. Moreover, as computational power increases, the simulators are becoming increasingly accurate models of the robots and their environments.

5 Experiments

The purposes of our experiments were to show basic capabilities and to demonstrate performance on our three design criteria: reliability, generalization, and adaptability. In this section, we describe three experiments. Experiment 1 shows that basic capability can be achieved: the simulated robot can use a POMDP model constructed from hand-tuned actions and learn to traverse the maze. The underlying hypothesis of this experiment was that learned actions would perform at least as well as, or better than, the hand-coded actions under normal operating conditions. Experiment 2 tests the hypothesis that learned actions will be more reliable than hand-coded actions in the presence of simulated sensor and effector failure. Experiment 3 tests the generalization of learned actions. The hypothesis was that learned actions will initially perform poorly in novel situations but will eventually perform better than hand-coded actions.

5.1 Simulation Environment

Khepera is a small (55mm diameter), modular robot. The basic system has two drive wheels, one on each side, that can be controlled independently, providing the ability to control speed and direction of the robot. Khepera has a small processor and eight infrared transmitter/receiver pairs. It also has pulse encoders on the wheels, but they are not used in this work. The low number of sensors allows fast simulation and reduces the amount of information which must be processed by the controller. However, it also limits the ability of the robot to make subtle distinctions in its environment. This could lead to difficulty in the robot knowing where it is.

The Khepera robot is simple enough to be adequately modeled in a simulator (Michel 1995). The discrete time simulator can run in real-time or much faster than real time. The user-supplied robot controller is called by the simulator at each time tick. The sensor model includes stochastic modeling of noise and responds similarly to the real sensors. The simulation environment includes some stochastic modeling of wheel slippage and acceleration.

The user writes a set of subroutines that conform to a simple programming interface. The subroutines are used by the simulator to initialize, run, and shutdown the user control code. The simulator provides routines that allow the controller to query sensors and control effectors.

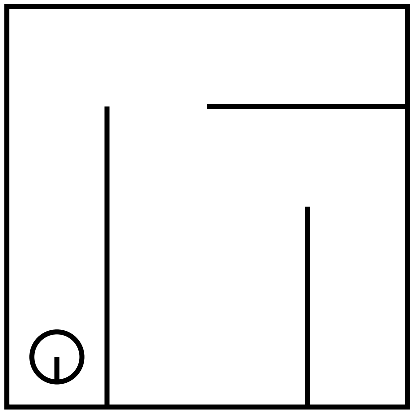
The user can create, save, and restore a simulated environment with obstacles. However, the simulator does not include any settings to simulate failures. We added hooks into the simulator to allow simulated sensor and effector failures.

5.2 Creating the POMDP Model

For these tests, a small navigation area was created and divided into 16 positions. Then the robot's orientation was discretized into the four compass directions, resulting in 64 possible states for the robot. The resulting environment and state space is shown in Figure 4. For belief state maintenance, sensor information was pre-processed and reduced to four bits by combining the sensors in pairs and setting a threshold on the combined sensor values. This resulted in 16 observations for the POMDP model. These measures kept the state space size small so that exact solutions could be found for the POMDP policy. However, even with this simple state space, generating a policy required over two weeks of computation.

The POMDP model was initialized using three hand-coded actions. The actions were: go forward, turn left, and turn right. The hand-coded actions try to avoid obstacles while performing their required function. With the hand-coded actions installed in the library, all state transition and observation probabilities in the POMDP were initialized to zero.

Next, the robot was placed at a random position and orientation within each of the 64 possible states for 30 trials each. On ten of the trials, the robot executed the forward action. On ten of the



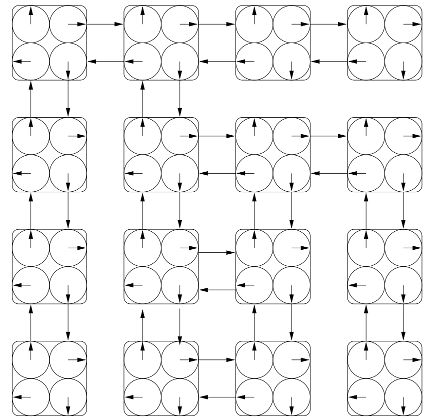


Figure 4: Environment and state space for the experiments.

trials, it executed the turn left action, and on the remaining ten of the trials, it executed the turn right action. At each time step during a trial, the actual state of the robot and the observation from sensors were recorded. For each set of ten trials, the observed transitions and observations were used to calculate a rough estimate of the transition and observation probabilities for that state in the POMDP model. This "seeded" the POMDP transition and observation probabilities.

A simple control policy was hand-coded to cause the robot to wander about in the environment. The control policy simply chose an action at random and executed it until either a state transition was observed or 500 time steps transpired. Then another action was chosen and executed. As the robot wandered around in its environment, state transitions were noted, and the POMDP model was further refined. Figure 5 shows the improvement in belief state estimation as the robot wandered in its environment. After 300,000 time steps, the most likely state matched the actual state more than 90% of the time. Most of the discrepancies were caused by the most likely state changing one or two time steps before or after the actual state change. Initialization would have taken approximately 300 minutes on a real robot but only required about 5 minutes on the simulator.

For simplicity and due to computational limitations, state 13 (the robot's location in Figure 4) was chosen to be the goal for all of the experiments. The parallel restricted region algorithm was used to find a policy that would direct the robot to state 13 from any initial state. This is a very large POMDP problem, with 64 states, 3 actions, and 16 observations. To reduce the time required to generate the policy, a high threshold was set on the value returned by the linear program solver. Cassandra (1998a) proposed this method for speeding up POMDP solution, which

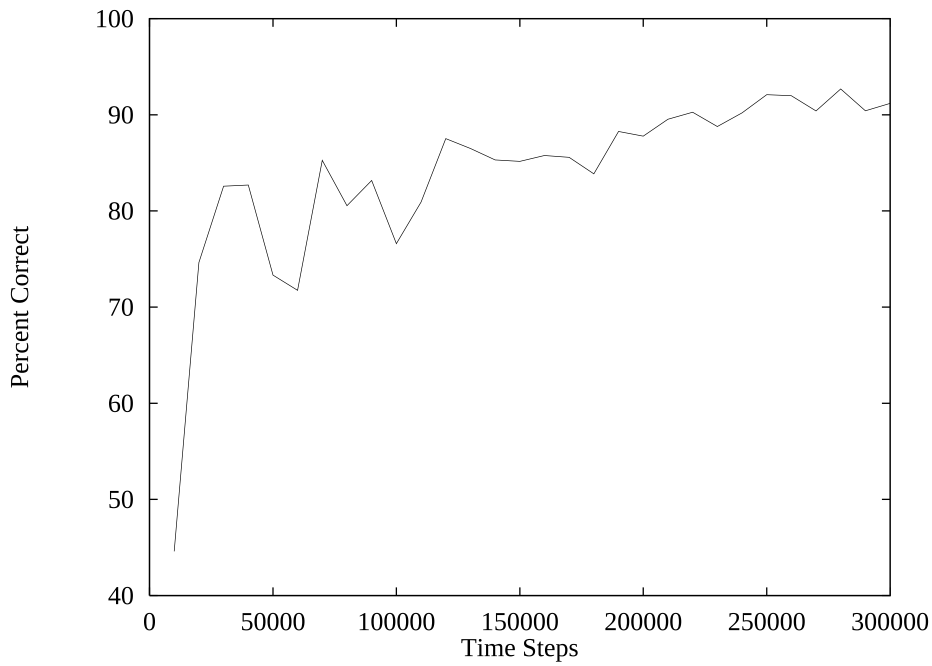


Figure 5: Accuracy of the most likely state improves as the robot wanders around in its environment.

has become common practice. Using a high threshold probably resulted in a policy that was less than optimal. However, the policy was adequate to provide reliable performance. Even so, finding a policy required approximately 16 days of computation; thus we used only one policy for the three experiments.

6 Experiment 1: Basic Competence

The purpose of the first experiment was to show that the components of the architecture would work together to achieve basic competence in the environment. From an experimental viewpoint, the hypothesis was that the hand-coded actions could be replaced with learned actions and that the learned actions would perform equally or better than the hand-coded actions.

After solving the POMDP to find the policy, the hand-coded actions were replaced with decision tree based reinforcement learning components. For training the RL actions, the robot was always started in the southeast corner of the environment and had to reach the goal state in the southwest corner of the environment, as shown in Figure 6. This selection of starting state required the robot to use all three actions. Using the same task for all trials simplified analysis of the results. The new

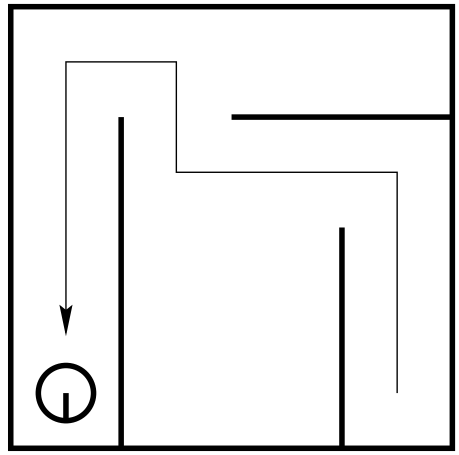


Figure 6: The path used for learning the actions.

actions learned what to do under complete control of the POMDP planner. The robot was run for 500 trials (one trip from start to goal), with the POMDP planner assigning rewards to the actions at each state change. An additional 500 trials were run with the hand-coded actions in order to establish a performance baseline.

Figure 7 shows the performance of the learned actions relative to the hand-coded actions. In less than 100 trials, the learned actions achieved similar performance to the hand-coded actions and continued to improve over the next 100 trials, while occasionally performing worse than the hand-coded actions. After 250 trials, the learned actions were performing equal to or slightly better than the hand-coded actions. The hand-coded actions were programmed using human intuition about what effector settings are appropriate for a given situation. The learned actions, on the other hand, use reinforcement learning to search for an optimal control policy. It is hardly surprising, then, that the learned actions slightly outperform the hand-coded actions.

7 Experiment 2: Reliability

We expected the architecture to be reliable because the POMDP layer does not rely on knowing the exact state, and the RL layer learns to follow the POMDP's expectations. Thus, the system should perform reliably even when the robot's hardware degrades, up to a point. The hypothesis is that the overall performance should degrade gracefully (i.e., roughly linearly with a shallow slope) as sensors and actuators gradually fail until information or control prove inadequate.

To test the hypothesis, the sensor or actuator performance was degraded in a controlled manner

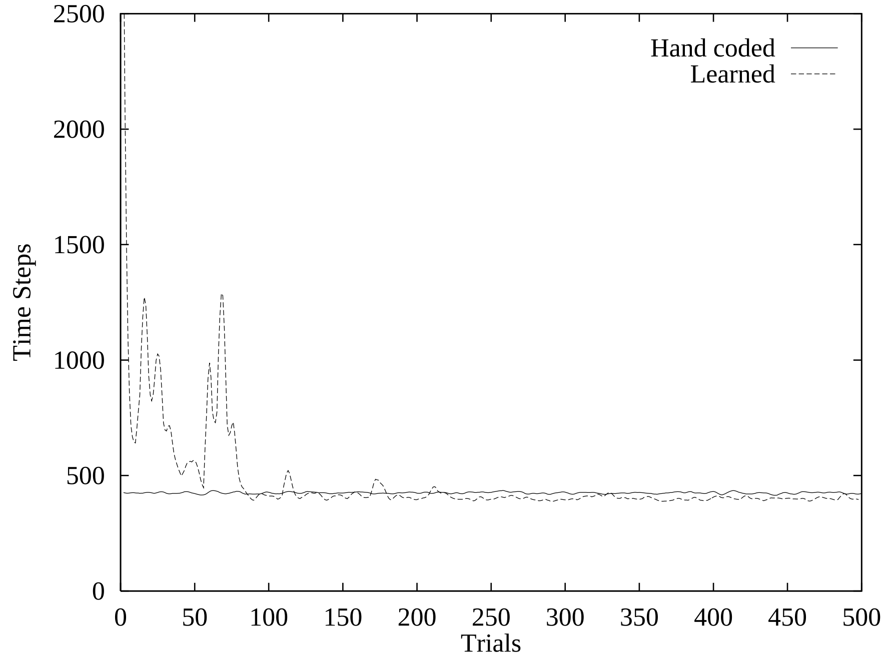


Figure 7: Performance of hand-coded and learned actions while learning the three actions.

and measurements were taken on the performance of two systems: one with the learned actions and the other with hand-coded actions. The hand-coded actions provide a control or baseline of performance to demonstrate task difficulty. The learned actions from the previous experiment were used in this experiment.

For each trial, either the simulated sensor or actuator performance was degraded by a fixed amount. For each system configuration (learned or hand-coded), the simulated robot was started from every position and orientation in the environment and the number of time steps that it took to reach the goal was recorded. On some trials, the robot became wedged and was unable to reach the goal. After a failure, the robot is re-started in the same position and orientation to try again. Thus, for each setting of degradation, data was collected from 64 instances of the robot achieving the goal.

Performance was assessed with respect to two metrics. *Failures during trial* counts the number of times that the robot was unable to reach the goal within 7500 time steps over all 64 configurations in a trial; thus, this is the estimator of reliability. *Average steps to goal* assesses efficiency: how long does it take to achieve the goal. Obviously, repeated failures in a trial causes this number to increase considerably.

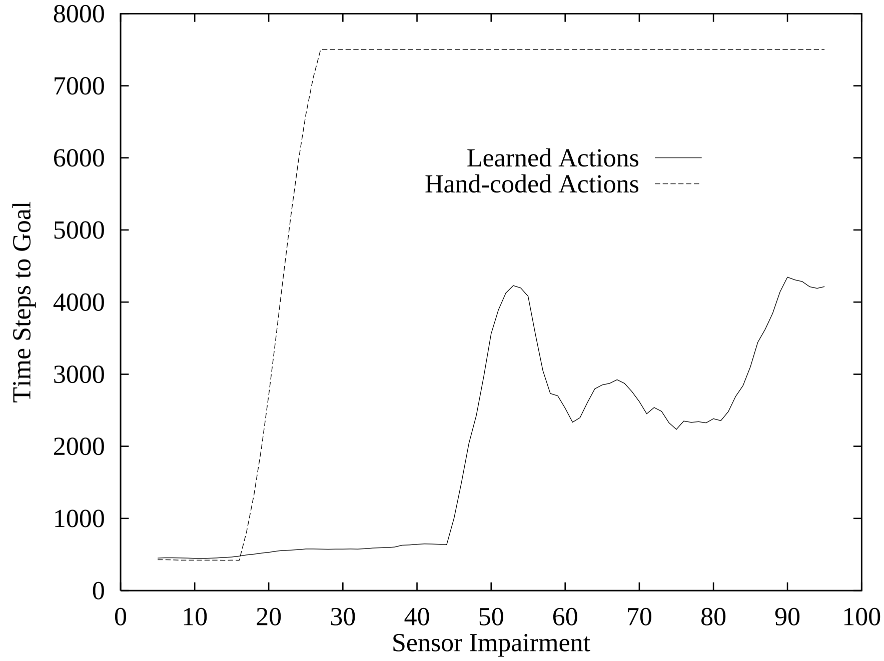


Figure 8: Response to gradual failure of all sensors.

Gradual Sensor Failure Sensors can gradually fail as battery power is used up, when dust accumulates on the sensors, or when calibration is lost. To assess the effect of gradual sensor failure on system performance, a scaling factor was set on all of the sensors to restrict their range. If the range of the sensors was 90% of normal, then the sensors were said to be impaired by 10%. Testing started with all sensors fully operational (0% impairment). For each unit (1%) impairment from 0% to 100%, the robot was directed to go to state 13 from state 64.

Figure 8 shows the results of this experiment. The curves were pre-processed with a Gaussian eleven smooth ($\sigma = 4$) (Cohen 1995). The hand-coded actions failed to reach the goal within 7500 time steps on every trial where the sensor degradation was greater than 21%. The hand-coded actions use thresholds on the IR distance sensors to determine when the robot has collided with a wall and take appropriate action. At sensor impairments above 21%, the collision threshold is never exceeded. The thresholds were chosen after careful trial-and-error to perform well in the best case, when there was no sensor degradation. Changing the thresholds could result in better performance in the presence of sensor failure but at the cost of poorer performance in the best case. For the hand-coded actions, as expected, the time steps to reach the goal is strongly correlated with the number of failures.

The learned actions show better reliability. The robot failed to reach the goal within 7500 time steps on only four of the 100 trials. The time steps to reach the goal during a trial increases gradually, staying under 600, even with sensor impairment as high as 50%. At sensor impairments greater than 50%, the sensor pre-processor for the belief state maintenance module no longer reports all observations. A threshold in the sensor pre-processor is never exceeded, and the only observation that is reported to the belief state maintenance module indicates that nothing is nearby. As a result, the belief state maintenance module only has information from the transition probabilities to determine the current belief state. Even when no observations are reported, the belief state maintenance module continues to operate well enough for the robot to reach its goal state on 92% of the remaining trials, although it takes a great deal more time. This demonstrates an important strength of the POMDP approach to high level control. In the absence of sensor input, it degrades to a form of dead-reckoning based only on the transition probabilities.

Intermittent Effector Failure Effectors may partially or intermittently fail due to loose contacts, shorted motor windings, or other causes. To assess the effect of intermittent partial effector failure, a subroutine that generated the control signal for the right motor was modified to reduce the value of the command signal by a fixed amount with a 50% probability. For example, although the robot control system issued a command for the motor to run at a speed of -5, the failure simulation routine may actually send a -3 command to the right motor. If the command signal was negative, then 2 was added; if the signal was positive, then -2 was added. Mild failures were simulated with ± 2 and severe failures with ± 4 added to the right motor command signal. For each test, one trial was run with all sensors and effectors operational to establish a baseline and then 20 more trials were run while simulating an intermittent failure in the right motor.

Figure 9 shows the results of this experiment on the *average steps to goal* performance metric. Both hand-coded and learned actions performed very well with mild failure; the performance for both types of behaviors was only slightly degraded in the presence of intermittent effector failure. In the presence of severe effector failure, the learned actions quickly adapted to the failing effector and reached the goal on all trials. The hand-coded actions failed to reach the goal on any of the 20 trials with severe failure.

When the robot became wedged against a wall, the hand-coded actions use a “reverse and turn”

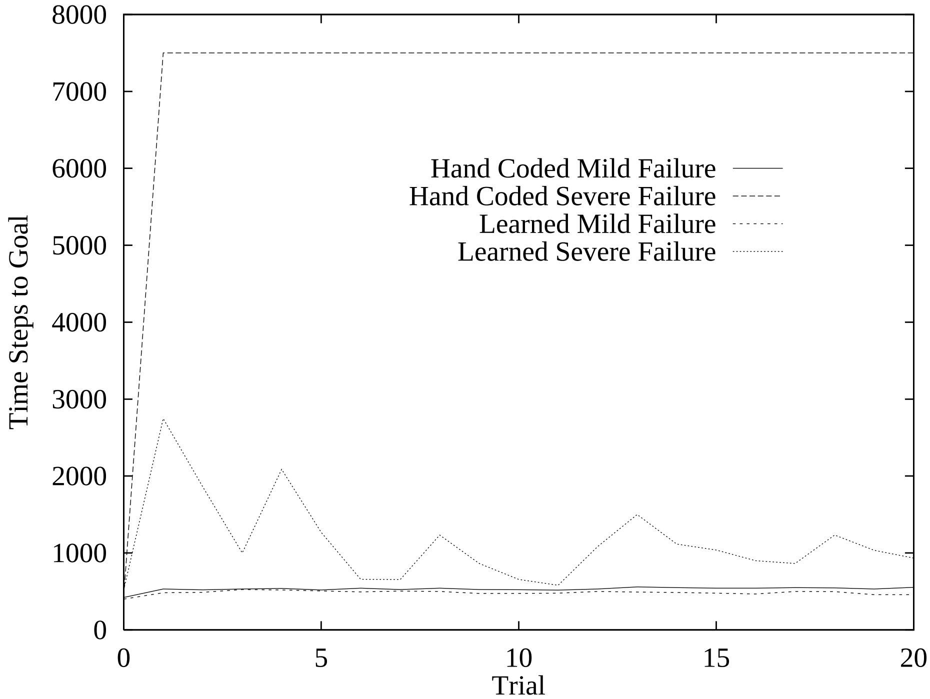


Figure 9: Response to intermittent failure of an effector.

strategy for getting unwedged. When a wedged situation was detected, the action sets a timer and sets the effectors to move away from the wall. The effector settings are not changed for a fixed number of time steps determined by the timer. When the timer runs out, the effectors are set to turn the robot away from the wall for one time step, and then normal operation is resumed. This approach was sufficient to provide reliable operation when effectors were working perfectly or with mild effector failure. However, with severe failure, this strategy no longer worked because the robot was unable to move far enough away from the wall and/or to change its heading sufficiently when attempting to get unwedged. Other strategies might be employed in the hand coded rules, but to do so efficiently would have required recognizing what went wrong. The learned actions did not have a fixed strategy for dealing with this situation and could search for a better strategy when the current one no longer worked. Thus, on the first five trials with severe failure, the learned actions were adapting to the new effector response. After about five trials, the actions had adapted and performed well considering the reduced reliability of the effectors.

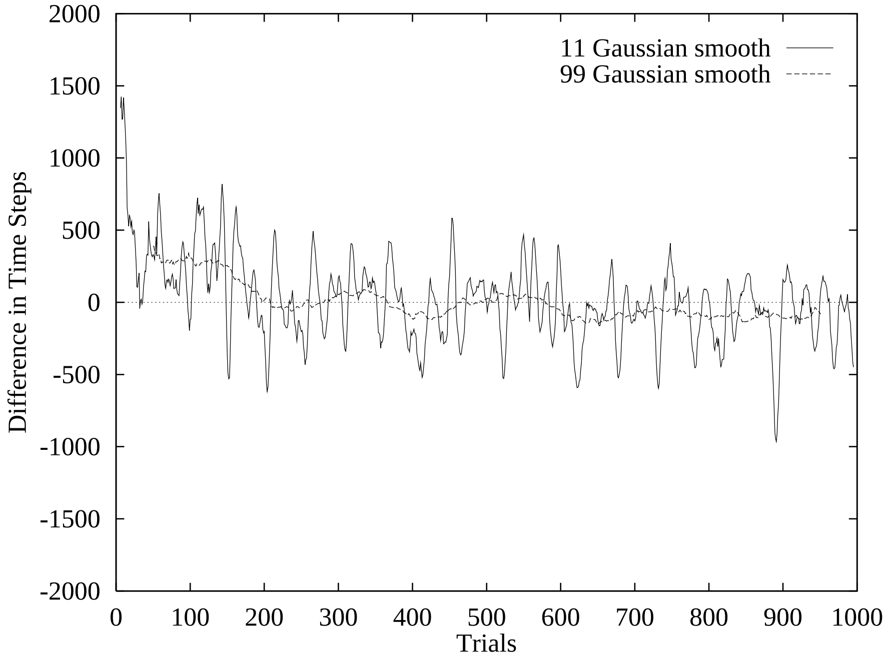


Figure 10: Difference between learned and hand-coded actions in time taken to reach the goal.

8 Experiment 3: Generalization

The robot followed the same path on each trial when learning the actions. The path took the robot to only 16 of the 64 states in the environment, and the path had more left turns than right turns. It would be reasonable to expect that the learned actions will perform well for the states that they have been trained on and that they will not perform as well in novel situations, but to what degree?

A position and orientation was chosen within each of the 64 states, one at a time, and the robot was required to navigate to the goal twice from that initial position. For each new start, the time steps to the goal were recorded for one trial of each version of the robot: learned and hand-coded actions. This was repeated for 1000 trials each.

Figure 10 is a plot of the relative performance of learned vs. hand-coded actions. The plot shows the time taken for the hand-coded actions minus the time taken for learned actions to reach the goal over 1000 trials. The plot is shown with an 11 Gaussian smooth ($\sigma = 7$) and a 99 Gaussian smooth ($\sigma = 65$). Over the first 200 trials, the learned actions did not perform as well as the hand-coded actions, but performance was improving. For trials 200 to 600, performance was roughly equal for both sets of actions. After about 600 trials, the learned actions were taking slightly less time to reach the goal, on average. For trials 500 through 1000, the standard deviation for learned actions

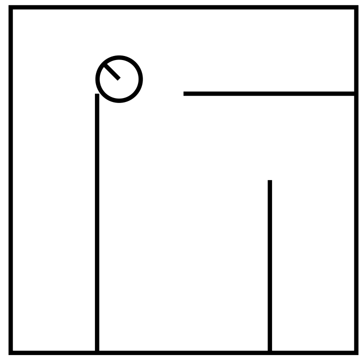


Figure 11: The location in the environment where the robot most often became wedged during the generalization tests.

Table 1: Number of times the robot failed to reach the goal within 7500 time steps.

Action Type	Trials			
	1-250	251-500	501-750	751-1000
Learned	1	0	0	0
Hand-Coded	5	4	3	6

was 357.7, and for the hand-coded actions it was 544.8, indicating that the time required for the hand-coded actions to reach the goal was less consistent than for the learned actions.

However, in some trials, the robot was unable to reach the goal within 7500 time steps. This occurred in a few of the trials where the robot became wedged against a wall, but did not detect a wedged situation due to the coarse thresholding of the sensor information and the sparse sensor coverage of the Khepera robot. The most common location for this failure to occur is shown in Figure 11; at this location in the environment, the sensor coverage of Khepera is inadequate to detect the wall. Reducing the probability of this failure resulted in additional code complexity in the hand-coded actions and required careful setting of the sensor thresholds. As with the effector failure, we could not find an efficient hand codod solution to this problem. When the robot became wedged, the trial was halted and re-started after 7500 time steps. The failures are not reflected in Figure 10. So, Table 1 shows the number of failures for learned and hand-coded actions divided into four intervals. The learned actions failed to carry the robot to the goal only once within the first 250 trials and then did not fail.

Note that the learned actions found a way to deal with the situation where the robot was wedged but the sensory information did not indicate any wall and thus overcame what appears to be a design limitation in the Khepera robot. The timer mechanism in the POMDP belief state maintenance module is responsible for this adaptation. When the robot becomes wedged and the sensors do not indicate any obstacles, the low level action continues to try to move forward. However, the robot cannot move forward, and no state transition is detected by the belief state maintenance module. The timer runs out and the action is given a negative reward. After receiving several negative rewards, the action adjusts its policy and sends a different set of commands to the effectors, freeing the robot from its wedged position, and leaving the action with a policy that is appropriate for unwedging, but not for general use. Thus, the higher level of control continues to command the move forward action, but the lower level adapts and changes what it is doing. Of course, the next time the action is used, it may choose the inappropriate effector settings. If so, then it will receive negative rewards until it re-learns its normal operation. Thus, the system is able to trade off some efficiency in order to improve reliability.

9 Conclusions

We performed a series of experiments in which we compared two versions of our own architecture to see how well our implementation achieved our design criteria. Although such experiments still do not indicate what possible architecture is best, they do exhibit specific advantages of one set of choices over another, which guides future designers in their choices of architectures. In this case, we showed that the two layer POMDP/RL architecture improved reliability and generalization for a simple simulated robot navigation problem, thus, partially validating our design.

However, while useful for informing design, such comparisons still are not enough. Three issues are yet to be resolved: fairness, specificity, and generalization. First, on fairness, the comparison pitted our design against one in which the learning component had been replaced by a hand-coded component, a strawman in effect. The hand-coded actions were programmed to operate from any position of the maze. Because the navigation task did not appear to require adaptability, the comparison seemed fair. Yet, although substantial effort went into the development of the hand-coded actions, they were still somewhat brittle. In the last experiment, the robot, when using hand-coded actions, failed to reach the goal on 1.8% of the trials. The learned actions were much more robust, failing only one time in 1000 trials. This suggests that even a simple simulated controller was more complex than we expected by programming it.

Although we genuinely tried to build a good hand-coded controller, its complexity was significantly lower than the learned actions. The hand-coded actions were implemented in 76 lines of C++ code. We found it difficult to hand-code an action that could work appropriately in all situations. The learned actions were based on the same RL code and required on the order of 4 Kilobytes of storage for the value function.

Second, were we specific enough about the design criteria to support the validation? We focused on three design criteria: reliability, generalization, and adaptability. We found out that they are interrelated, e.g., reliability required some adaptability. In addition, it is obvious that the earlier criteria descriptions are not complete. For each of our criteria, we needed to identify the specific environmental factors to vary, operationalize the measures, and determine the thresholds or relationships that would be said to achieve the criteria. To fulfill the promise of expediting cross-architecture comparison, the field will need to agree on such definitions and the factors that influence them.

Third, although we showed performance differences in each experiment and proposed explanations for those differences, we still cannot be said to understand *why* the architecture works. To better understand its function, we will have to probe more of the design decisions over more environmental conditions. Only then will we be able to confirm under what conditions particular design decisions are appropriate.

9.1 Directions for Future Research

As expected, the RL/POMDP combination exhibits reliable execution in the presence of sensor and actuator degradation and generalization to different initial conditions. The current test application was not inhibited by the known limitations of POMDP and RL methods, specifically, the amount of training required or problem size. However, future work on this project is directed at scaling up the size of the problem and giving the architecture further autonomy. In particular, the architecture must be scaled to work on a robot with more sensors in an environment with hundreds or even thousands of states. Simmons and Koenig (1995) showed that sensor pre-processing can be used to simplify the POMDP model on robots with many sensors. A similar approach in conjunction with the decision tree based reinforcement learning may provide sufficient scaling for RL as well. Thus, a first step is to generalize the architecture further.

Additionally, as suggested earlier in this section, the architecture needs to be tested in a wider range of conditions and compared to alternatives with other design decisions. For example, we proposed that the credit assignment of the POMDP planner may contribute to helping the robot unwedge itself; to confirm this, we should test alternative methods of credit assignment.

The demands of robotics require considerable creativity in experiment design. Replicability and controlled comparisons of multiple systems, the staples of other areas of AI, are often infeasible in physical robotics. However, simulators can expedite replication and control by running faster than real-time and by providing parameters that can be tuned. Although a simulator will never match the fidelity and complexity of the real world, it can often come close enough to draw conclusions and can sometimes surpass what we can test physically by simulating other worlds (literally, as in the case of NASA's domains and figuratively for hostile environments on our own planet). Additionally, careful comparisons of versions of single systems in a wide variety of conditions can serve a similar purpose in furthering our understanding of what works best when.

10 Acknowledgments

This research was supported in part by National Science Foundation Career Award IRI-9624058. The United States Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation herein. We also wish to thank the anonymous reviewers and the issue editor for their well considered comments and suggestions. We regret that time constraints precluded adopting all of them in this article.

References

Arkin, R. C., I. Noda, S. Suzuki, H. Matsubara, M. Asada, and H. Kitano (Eds.) (1998, Fall). *AI Magazine*, Volume 19.

Bagnell, J. A., K. L. Doty, and A. A. Arroyo (1998). Comparison of reinforcement learning techniques for automatic behavior programming. In *Carnegie Mellon Conference on Automated Learning and Discovery*.

Cassandra, A. R. (1998a, May). *Exact and Approximate Solutions for Partially Observable Markov Decision Processes*. Ph. D. thesis, Brown University, Providence, RI.

Cassandra, A. R. (1998b, October). A survey of POMDP applications. In M. Littmann (Ed.), *Working Notes: AAAI Fall Symposium on Planning with Partially Observable Markov Decision Processes*, pp. 17–24. AAAI.

Cassandra, A. R., L. P. Kaelbling, and J. A. Kurien (1996). Acting under uncertainty: Discrete bayesian models for mobile robot navigation. In *Proceedings of the IEEE/Robotics Society of Japan Conference on Intelligent Robots and Systems*. IEEE.

Cohen, P. R. (1995). *Empirical Methods for Artificial Intelligence*. MIT Press.

Dorigo, M. and M. Colombetti (1994, December). Robot shaping: developing autonomous agents through learning. *Artificial Intelligence 71*(2), 321–370.

Feiten, W., U. Wienkop, A. Huster, and G. Lawitsky (1994). Simulation in the design of an autonomous mobile robot. In R. Trappe (Ed.), *Cybernetics and Systems '94*, pp. 1499–1506. Singapore: World Scientific Publishing.

Haigh, K. Z. and M. M. Veloso (1997, February). High-level planning and low-level execution: Towards a complete robotic agent. In W. L. Johnson (Ed.), *Proceedings of First International Conference on Autonomous Agents*, Marina del Rey, CA, pp. 363–370. ACM Press, New York, NY.

Koenig, S., R. Goodwin, and R. G. Simmons (1996). Robot navigation with Markov models: A framework for path planning and learning with limited computational resources. *Lecture Notes in Computer Science 1093*, 322.

Koenig, S. and R. G. Simmons (1998). Xavier: A robot navigation architecture based on partially observale markov decision process models. In D. Kortenkamp, R. P. Bonasso, and R. Murphy (Eds.), *Artificial Intelligence and Mobile Robots*. AAAI Press.

Kontoravdis, D., A. Likas, and A. Stafylopatis (1992). Collision-free movement of an autonomous vehicle using reinforcement learning. In B. Neumann (Ed.), *Proceedings of the Tenth European Conference on Artificial Intelligence*, Chichester, UK, pp. 666–670. Wiley.

Mahadevan, S. and J. Connell (1992). Automatic programming of behaviour-based robots using reinforcement learning. *Artificial Intelligence 55*(2/3), 311–365.

Mahadevan, S., N. Khaleeli, and N. Marchalleck (1997, November). Designing agent controllers using discrete-event markov models. In *AAAI Fall Symposium on Model-Directed Autonomous Systems*, MIT, Cambridge.

Michel, O. (1995). Khepera simulator. Available for download from `http://diwww.epfl.ch/-lami/team/michel/khep-sim/`.

Pyeatt, L. D. (1999, July). *Integration of Partially Observable Markov Decision Processes and Reinforcement Learning for Simulated Robot Navigation*. PhD dissertation, Colorado State University, Computer Science Department.

Pyeatt, L. D. and A. E. Howe (1998a, October). Decision tree function approximation in reinforcement learning. Tech Report TR CS-98-112, Colorado State University, Fort Collins, Colorado.

Pyeatt, L. D. and A. E. Howe (1998b, October). Integrating POMDP and reinforcement learning for a two layer simulated robot architecture. In M. Littman and T. Cassandra (Eds.), *AAAI 1998 Fall Symposium Series – Planning with Partially Observable Markov Decision Processes: Working Notes*, Orlando, Florida, pp. 371–388. AAAI. Revised version appeared in *Third International Conference on Autonomous Agents*.

Pyeatt, L. D. and A. E. Howe (1998c, May). Learning to race: Experiments with a simulated race car. In D. J. Cook (Ed.), *Proceedings of the Eleventh International Florida Artificial Intelligence Research Symposium Conference*, Sanibel Island, FL, pp. 357–361. Florida Artificial Intelligence Research Symposium: AAAI Press.

Pyeatt, L. D. and A. E. Howe (1998d, March). Reinforcement learning for coordinated reactive control. In *Fourth World Congress on Expert Systems*.

Pyeatt, L. D. and A. E. Howe (1999, September). A parallel algorithm for POMDP solution. In *Proceedings of the Fifth European Conference on Planning (ECP-99)*, Durham, United Kingdom.

Ring, M. B. (1993). Two methods for hierarchy learning in reinforcement environments. In J. A. Meyer, H. Roitblat, and S. Wilson (Eds.), *From Animals to Animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior*, pp. 148–155. MIT Press.

Ring, M. B. (1994, August). *Continual Learning in Reinforcement Environments*. Ph. D. thesis, University of Texas at Austin, Austin, Texas.

Roy, N., W. Burgard, D. Fox, and S. Thrun (1998, October). Coastal navigation - robot motion with uncertainty. In M. Littman and T. Cassandra (Eds.), *AAAI 1998 Fall Symposium Series – Planning with Partially Observable Markov Decision Processes: Working Notes*, Orlando, Florida, pp. 135–140. AAAI.

Simmons, R., R. Goodwin, K. Z. Haigh, S. Koenig, and J. O'Sullivan (1997, February). A layered architecture for office delivery robots. In W. L. Johnson (Ed.), *Proceedings of First International Conference on Autonomous Agents*, Marina del Rey, CA, pp. 245–252. ACM Press, New York, NY.

Simmons, R. G. and S. Koenig (1995). Probabilistic navigation in partially observable environments. In *Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada, pp. 1080–1087. IJCAI: Morgan Kaufmann.

Singh, S. P. (1991). The efficient learning of multiple task sequences. In *Advances in Neural Information Processing Systems 3*, San Mateo, CA. Morgan Kaufman.

Singh, S. P. (1992, May). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning 8*, 323–339.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning 3*, 9–44.

Timin, M. E. (1995). Robot automobile racing simulator (RARS). Anonymous ftp `ftp.ijs.com:/rars`.

Wyatt, J. (1995, April). Issues in putting reinforcement learning onto robots. In *Mobile Robotics Workshop, 10th Biennial Conference of the AISB*, Sheffield, UK.

Zhang, N. L. and W. Liu (1996). Planning in stochastic domains: Problem characteristics and approximation. Technical Report HKUST-CS96-31, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong.