Proceedings of the 2001 IEEE
International Conference on Robotics & Automation
Seoul, Korea - May 21-26, 2001

Real-Time Robot Learning

Bir Bhanu, Pat Leang, Chris Cowden, Yingqiang Lin and Mark Patterson

Center for Research in Intelligent Systems
University of California, Riverside, CA 92521
{bhanu, pleang, ccowden, yqlin, mark}@vislab.ucr.edu

Abstract

This paper presents the design, implementation and testing of a real-time system using computer vision and machine learning techniques to demonstrate learning behavior in a miniature mobile robot. The miniature robot, through environmental sensing, learns to navigate a maze choosing the optimum route. Several reinforcement learning based algorithms, such as Q-learning, Q(λ)-learning , fast online Q(λ)-learning and DYNA structure, are considered. Experimental results based on simulation and an integrated real-time system are presented for varying density of obstacles in a 15×15 maze.

1. Introduction

Real-time robot learning has been a very challenging and active research area for several years [3]. Reinforcement learning has been applied to robot learning and machine intelligence tasks, and several different techniques have been developed [8, 9, 11]. In this paper, we consider a systematic comparison of some key reinforcement learning techniques, such as Q-learning, Q(λ)-learning, fast Q(λ)-learning, DYNA-based learning and an algorithm that combines heuristics with Q-learning. First, we compare these algorithms in simulation for navigation in a maze that may contain a number of obstacles. Second, we build a real maze and a miniature robot equipped with basic sensors and a vision camera that monitors the activity of the robot. We integrate a number of reinforcement learning techniques for real-time navigation and control of the miniature robot.

In Section 2, we present the related research using reinforcement learning. This is followed by a discussion of several reinforcement learning algorithms that we compare in Section 3. Here we also present the details of the robot, interface and the integrated learning system. Section 4 presents experimental results, both in simulation and in real time. Finally, Section 5 presents the conclusions of the paper.

2. Related Research

Asada et al. [1] present a method for vision based reinforcement learning where a robot learns to shoot a ball into a goal. The robots use a technique called "learning from easy missions," which reduces the learning time to about the linear order in the size of the state space. Yamagnchi et al. [13, 14] propagate learned behaviors of a virtual agent to a physical robot in order to accelerate learning in a physical environment. Reinforcement learning is used as the basis of the ball-pushing task. Suh et al. [10] extend Q-learning that incorporates a region-based reward to solve a structural credit assignment problem and a triangular type Q-value model. This may enable a robot to move smoothly in a real maze. Hailiu and Sommer [5] embed syntax rules and environment knowledge into the learner to achieve satisfactory performance with a reinforcement-learning algorithm. Huber [6] presents a hybrid architecture that applies reinforcement on top of an automatically derived, abstract, discrete event dynamic system supervisor. This reduces the complexity of the learning task and allows the incorporation of *a priori* knowledge. Bhanu and Peng [2] have also developed techniques to incorporate *a priori* knowledge into a reinforcement-learning framework for integrated image segmentation and object recognition.

As compared to the previous works, we present a systematic comparison of old and some new reinforcement-learning algorithms (for example, fast Q(λ), where complexity is O(| actions |)), and implement and evaluate them on a real miniature robot which has a bi-directional wireless link with the computer.

3. Technical Approach

Figure 1 shows various components of the integrated system for robot learning that we developed. It consists of a maze, an overhead camera, a robot, a vision module, a learning module and an interface module. In the following, first, we describe various reinforcement learning algorithms that we have considered. This is followed by the details of the real-time integrated system that we have developed.

3.1 Reinforcement-Learning Algorithms

The reinforcement algorithms we consider are Q-learning, Q(λ)-learning, fast Q(λ)-learning and some of their variants, such as incorporating DYNA structure and some environmental information into the learning algorithm.

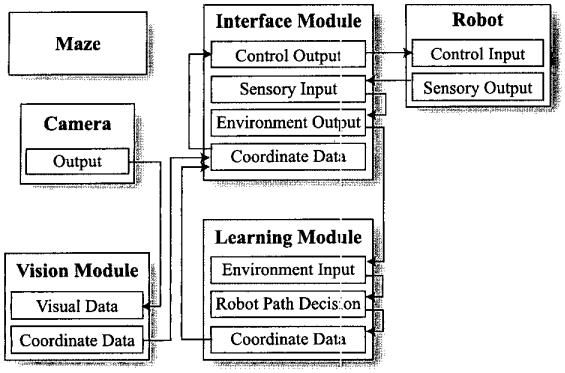


Figure 1. Various modules of the robot learning system.

• Discrete-Time Finite Markov Random Process

The discrete time finite Markov random process is a 4-tuple *(S, A, P, R)*, where:

S is a finite set of states: $S = \{S_1, S_2, \ldots, S_n\}$

A is a finite set of actions taken at each state.

P is a probabilistic state-transition law. It is a function P_{ij}^{a}: $S \times S \times A \rightarrow \Re$, where $\Re$ is the set of real nos.

$P_{ij}^{a} = P\{S_{t+1} = S_j \mid S_t = S_i, a_t = a\}$ *for* $S_i, S_j \in S$ *and* $a \in A$

P_{ij}^{a} is the probability to reach state S_j from state S_i when action a is taken at state i. S_t means at time t, the agent is at state S_t.

R: $S \times A \rightarrow \Re$ is the reward function mapping $(S_i, a) \in S \times A$ to a scalar reinforcement $R(S_i, a)$.

The reward at time t is r_t, and a_t is the action taken at time t. A discount factor $\gamma \in [0, 1]$ discounts later against immediate rewards.

As an example, in the maze problem, the state is the cell in the maze. The actions that can be taken at each state are: traveling north, traveling west, traveling south and traveling east. The state-transition function is very simple and deterministic. For example, if the agent takes the action of going to the north, the next state is the cell above; if the cell is in the top row, or if the cell above is occupied by an obstacle, the agent stays in the original cell. If the agent achieves the goal, the reward is 100. Otherwise, the reward is 0.

• ε-Greedy Policy

There is a trade-off between exploration and focusing on the path that has already been found. In order to find a short path, the agent must explore the search space (e.g., maze) actively at the beginning. But as the exploration proceeds, we assume the agent can find a good (short) path, so the agent should gradually focus on the path that has been found.

We use ε-greedy policy to guide the exploration. With probability ε, the agent explores the search space, and with probability 1 - ε, it goes along the best path already found. So the larger the value of ε, the more actively the agent explores the search space.

Initially, the value of ε is 0.9. Each time the agent finds a path, ε is decreased by 0.05. The minimum value of ε is greater than zero or a small positive number.

• Q-learning

Given (S_t, a_t, r_t, S_{t+1}), standard one-step Q-learning updates just a single Q-value $Q(S_t, a_t)$ as follows:

$$Q(S_t, a_t) \leftarrow Q(S_t, a_t) + \alpha\, e_t' \qquad e_t' \text{ is given by:}$$

$$e_t' = (r + \gamma V(S_{t+1}) - Q(S_t, a_t)), \quad \gamma \in [0, 1]$$

where $V(S) = \max_a Q(S, a)$ and α is the learning rate.

The agent can take one of the actions at each cell. We associate each action, which can be taken at each cell, with a value. This associated value represents the "goodness" of the corresponding action. We refer this value as the Q-value.

Algorithm 1 (Q-learning):

1. *Put the agent at the starting point.*
2. *While (not end) do*
3. *Use ε-greedy policy to select an action a_t.*
4. *Take the action to get (S_t, a_t, r_t, S_{t+1}).*
5. *Apply Q-learning to update Q-value.*
6. *if goal is reached then*
7. *change the value of ε and put the agent back at the starting point.*

end

end

• Q(λ)-learning

Q-learning only considers the immediate reward. It propagates the reward backward only one step. Unlike Q-learning, Q(λ)-learning not only considers the immediate reward, it also takes the discounted future rewards into consideration. Q(λ)-learning updates the Q-values in the following way:

$$Q(S_t, a_t) \leftarrow Q(S_t, a_t) + \alpha\, e_t^{\lambda}$$

$$e_t^{\lambda} = e_t' + \sum_{i=1}^{\infty} (\gamma\lambda)^i e_{t+i} \qquad \text{where} \quad \lambda \in [0, 1]$$

$$e_{t+i} = (r_{t+i} + \gamma V(S_{t+i+1}) - V(S_{t+i}))$$

• Online Q(λ)

The above updates in Q(λ)-learning cannot be made as long as future rewards are not known. However, we can compute them incrementally [7], by using eligibility traces. We define $\eta^t(S,a)$ as 1 if *(S,a)* occurred at time t, and 0 otherwise. Also,

$$\text{define } l_t(S,a) = \sum_{i=1}^{t-1} (\gamma\lambda)^{t-i} \eta^i(S,a), \text{ we have}$$

$$Q(S,a) \leftarrow Q(S,a) + \alpha\, [e_t' \eta^t(S,a) + e_t l_t(S,a)]$$

Based on the last expression, an online Q(λ)-learning algorithm is proposed in [9]. We can use Q(λ)-learning to replace the Q-learning in *Algorithm 1*.

• **Fast Online Q(λ)-learning**

Fast online Q(λ)-learning [12] is an improvement of Q(λ)-learning. At each step, it only updates a state-action pair *(S,a)* when it is needed, thus increasing the speed of learning. In Q(λ)-learning, all the *(S,a)* pairs in history list are updated at each step.

Note that the complexity of Q(λ) is O(|S||A|), but the complexity of fast Q(λ) is only O(|A|).

The algorithm is based on the observation that only Q-values needed at any given time are those for the possible actions given the current state. Hence, using "lazy learning", we can postpone updating Q-values until they are needed. Suppose some state-action pairs (SAP) *(S, a)* occur at steps $t_1, t_2, t_3, \ldots$. Let us abbreviate $\eta^t = \eta^t(S,a)$, where η^t is 1 for $t = t_1, t_2, t_3, \ldots$ and 0 otherwise, $\phi = \gamma\lambda$ and define:

$$\Delta_t = \sum_{i=1}^{t} e_i \varphi^i, \quad l'_t(S,a) = \sum_{i=1}^{t} \frac{\eta^t(S,a)}{\varphi^t}, \text{ we can have:}$$

$$\Delta Q(S,a) = \lim_{k \to \infty} \sum_{t=1}^{k} [e'_t \eta^t(S,a) + l'_t(S,a)(\Delta_{t+1} - \Delta_t)]$$

Based on the above expressions, we can build a fast online Q(λ)-learning algorithm. This algorithm relies on two procedures: the Local update procedure calculates exact Q-values once they are required; the Global update procedure updates the global variables and the current Q-value.

• **DYNA Structure**

With DYNA, we incorporate planning into learning process. It is proposed as a simple but principled way to achieve more efficient reinforcement learning in an autonomous agent.

DYNA algorithm:

1. *Initialize All Q(S,a) to 0 and the priority queue PQueue to empty.*
2. *While (not end)*
   - *(a) x ← the current state.*
   - *(b) Use ε-greedy to select an action a.*
   - *(c) Carry out action a to get the experience (x, a, y, r), where x is the current state, a is the action taken by the agent, and y is the next state and r is the immediate reward.*
   - *(d) Apply Q-learning to update Q-value.*
   - *(e) Compute e = | r + γV(y) – Q(x, a) |. If e ≥δ, insert (x, a, y, r) into PQueue with key e.*
   - *(f) If PQueue is not empty, do planning:*
     - *i. (x', a', y', r') ← **first_experience**(PQueue).*
     - *ii. Update:*
       $Q(x',a') = Q(x',a') + \alpha(r' + \gamma V(y') - Q(x',a'))$
     - *iii. For each predecessor x'' of x' do:*
       Compute $e = | r_{x''x'} + \gamma V(x') - Q(x'',a_{x''x'}) |$.
       If e ≥δ, insert $(x'', a_{x''x'}, x', r_{x''x'})$ *into PQueue with key e.*

δ is a parameter of the algorithm. It is a threshold value.

After we insert a *(x, a, y, r)* into PQueue, we can extract a fixed number of *(x, a, y, r)* from PQueue (if it has) and use them to do planning.

• **Incorporating Environmental Information**

We try to allow the agent to remember and use some environmental information. In particular, during exploration in the maze, we make the following assumptions:

1) Whenever the agent runs into an obstacle, it receives a negative reward (-100) and uses it to update the Q-value associated with the action taken at the cell. So, the next time the agent is at this cell, it won't take this action, avoiding the obstacle.
2) After the agent goes from cell S_1 to S_2, it won't go back from S_2 to S_1 immediately.
3) The agent remembers the cells it has visited in a trial. It won't visit a cell twice in a particular trial. Here, a trial is for the agent to go from the starting point to the goal. During exploration, after the agent goes from S_1 to S_2 and finds itself stuck, i.e., the other 3 neighbors of S_2 are either boundaries or obstacles, it has two choices: to either give up this trial and return to the starting point or to return to S_1 and continue the current trial.

We can also incorporate the environmental information and DYNA structure together into the learning process.

3.2 Integrated Learning System

• **Maze**

The maze used in real experiments is an 8-foot square block made from 2 sheets of 4×8 inch square plywood and bordered by 2×4 boards. The surface is entirely black with ¼-inch white lines. The lines are made with white typographer's tape. These lines, placed horizontally and vertically, produce a grid whereby each cell is 6×6 square inches. There are a total of 225 cells. The only two colors that exist on the maze are black and white.

The obstacles are made from 6-inch cubed blocks of wood. The obstacles are all black in color. The sides have been lined with paper to produce a smooth surface. This smooth finish allows for better reflection of IR light. Experiments have shown that the rough texture of the wood creates uncertainties in object detection because of the many different angles by which IR light can be reflected from the obstacle's rough wooden surface. Uncertainties in the amount of reflected IR signal causes errors in its detection, and thus, errors in recognizing the presence of the obstacle.

• **Robot Design**

The robot is a miniature robot that stands 6 inches tall with a diameter of 4.5 inches. The robot is controlled by the Handy Board micro-controller. It is based on the Motorola 68HC11 microprocessor. Features of the

controller include a 32K static RAM, four DC motor output ports, a variety of sensory inputs, and a 16×2 LCD display. The software used is Interactive C.

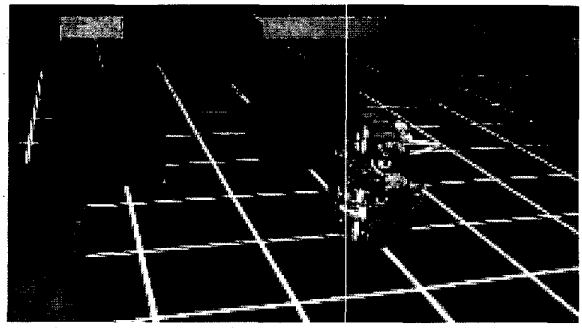


Figure 2. The miniature robot navigating the real maze in the presence of obstacles.

The robot is equipped with various sensors [4] to aid in communication with its environment and obstacle detection. It has four modulated IR proximity sensors mounted on its front, back, and on each of its sides for detecting objects within a 3-inch distance. A line sensor is mounted in the bottom front portion of the robot. It consists of three pairs of IR emitters and collectors. Detection of the ¼-inch white line depends on the amount of IR light that is reflected from the emitters to the collectors.

For added insurance, a touch sensor has been added in the front of the robot. Errors are inevitable, but the detection of these errors is invaluable. The touch sensor acts as a last resort by physically sensing the presence of an obstacle upon contact. In the event that the IR proximity sensor fails to detect the object, the touch sensor will eventually detect it, and thus, prevent the robot from constantly trying to run into the obstacles.

Two Tower Hobbies TS-53 servomotors propel the robot. These motors have been modified for continuous rotation, meaning that the position-sensing device has been removed as well the physical gear stopper, allowing the motor to turn continuously.

The communication between the robot and the host computer is done via an RF module. The RF module is made by Parallax, Inc. They are transmitter and receiver pairs, model TXAM315A and RXAM315A, respectively. Powered by 5V standard TTL output, these modules operate at 315 MHz. Conversion of the TTL signal to RS232 is done for communication with the computer.

The use of battery power to the robot has been completely eliminated due to problems with consistency. The robot was subjected to running for long hours at a fairly predictable level of performance. Batteries could not provide for this prolonged usage or consistency due to the power drainage over time. The solution was to connect it to a power cord so that any standard wall socket can serve as the power source. This would ensure continuous usage and power level consistency.

- **Vision Algorithms**

The vision system was implemented to ensure that overall activity would be error free. It ensures that the learning algorithm has correct information about the position of the robot. Before any command is sent to the robot, the vision system is called upon to determine the current position of the robot. If this position does not match the position that the learning algorithm thinks the robot is in, then a correction algorithm is invoked, in which case, the robot is automatically commanded to the correct position intended by the learning algorithm. Not until both vision and learning algorithm positions match will the program continue.

The vision hardware is the ITEX imaging system. It is a Modular Vision Computer (MVC) 150/40, and a CCD camera combined with a Sun Ultra 1 computer. The camera is mounted 10 feet above the center of the maze, and captures images at a rate of 30 frames per second. The image is captured in 256 gray scales. The image of the maze consists of 512×480 pixels with a resolution of 72 pixels per inch.

Software used to control the captured images was written in the C programming language. To detect the robot, a circular white disc is placed atop the robot. A white surface indicates the presence of a robot and a black surface indicates the maze floor. The software scans the image and looks at every sixth pixel. If the pixel value is within a certain prescribed threshold value for white, indicating the presence of a robot, then 4 surrounding pixels are also sampled. If all these 5 pixels are within the threshold value, then the robot has been located, and its corresponding position is relayed back to the learning algorithm for processing. The average runtime to sample an image is 0.10 seconds.

- **Interface**

The interface program was written in the C programming language. The program is run using the Sun Microsystems computer under a Solaris operating system. This is the main module that controls all other peripherals, which includes the robot, the RF communication between the robot and computer, the vision system, the learning algorithm, and the user interface. All information coming from these peripherals is routed and controlled by the interface program.

The program begins with the initialization of the peripherals. For the robot, it makes sure that the robot is ready to receive commands and that it is in the correct cell position for starting. In the event that the robot is not initialized, the interface scheme will allow for a user interface mode in which the user can command the robot, test its sensors, or test its response before the main learning program should begin. The vision system is also initialized, meaning that it sets up the hardware, initializes the display and calibrates the maze size. The RF communication is also set up to receive signals through its RS232 serial port at a rate of 300 baud.

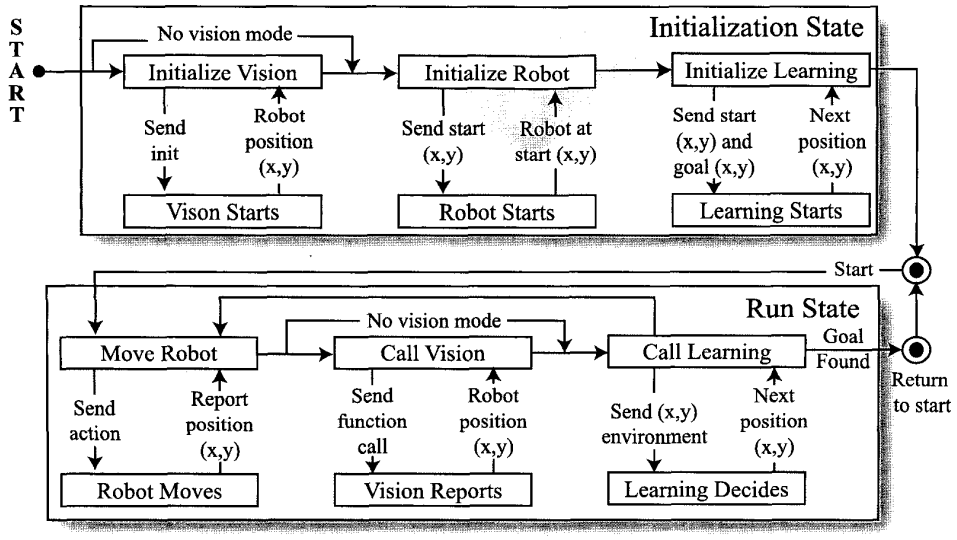


Figure 3. Details of initialization procedure and the run loop.

The sequence of events to control the robot is as follows. First of all, it queries the robot for environmental information. This means that it gathers information from the robot about surrounding obstacles in its 4 cardinal directions relative to the robot. Vision information is next queried for the robot's position. The intended position, the position assumed by the interface program, of the robot is compared against the actual position gathered from the vision. Discrepancies between the intended position and the actual position are handled by a position-correction algorithm, which relocates the robot to the correct position. Once both positions agree, the learning algorithm is called to determine the next position. The interface next commands the robot to this new position and this entire sequence of events is repeated until the goal is found. From goal position, the robot is directed to its starting position using a gradient-descent algorithm. From there, the next trial will begin. When the maximum number of trials has been reached, the shortest path is calculated based upon the type of learning algorithm used in the interface program. The entire event is done in real-time. Figure 3 shows the details of initialization procedure and the run loop.

4. Experimental Results

4.1 Simulation of Reinforcement-Learning Algorithms

A maze is a two-dimensional array with a cell as the starting point and some cells as a goal or goals. There may also be some cells occupied by obstacles. The agent can move from a cell to any one of its 4 direct neighbors with the restriction that the agent cannot move out of the maze, or into a cell occupied by an obstacle. The task of the agent is to find a good path from the starting point to the goals efficiently. A good path should be a short path, if it is not the shortest one.

Figure 4 is an example maze. The size of the maze is 18×12. The light gray cell on the left side is the starting point; the four cells on the upper right side are the goal; the dark gray cells are obstacles and the rest are empty cells. One of the paths from the starting point to the goals is also shown and it consists of a sequence of mid gray cells. The path shown here is one of the shortest paths with length 27 from the starting point to the goals.

The initial value of ε is 0.9. Each time the agent finds a path, ε is decreased by 0.05. The minimum value of ε is 0.1. The number of trials is 1000, i.e., the algorithm stops after the agent goes from the starting point to the goals 1000 times. The values of the parameters are: α=0.5, γ=0.95, λ=0.9. Threshold value for eligibility trace is 0.0001 and δ=10. Figure 5 shows the learning curves. The horizontal axis represents the number of successful trials and the vertical axis represents the number of steps. Here, we only show the first 200 trials.

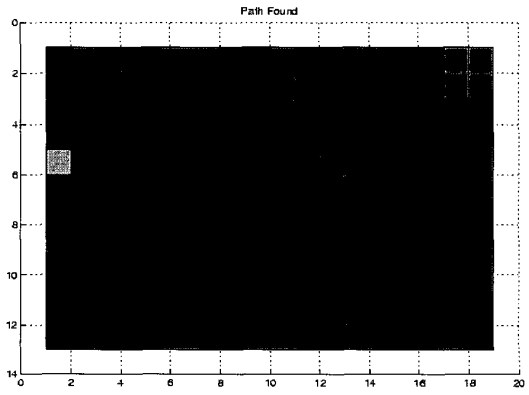


Figure 4. Simulated maze

Figure 5(a) is the learning curve of pure Q-learning. The length of the path found by the agent is 28. The total number of steps in all 1000 trials is 80156. The algorithm runs in less than 1 second.

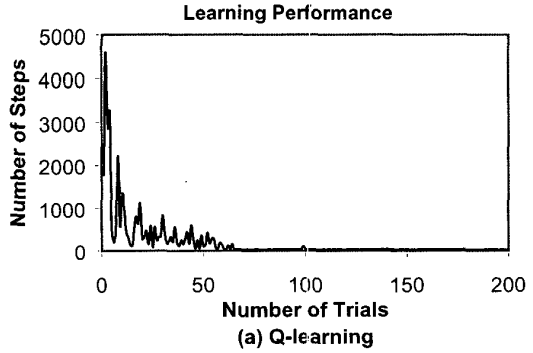
Figure 5(b) and 5(c) show the learning curves of pure Q(λ) and fast Q(λ)-learning. The length of the path found by the agent is 30. Q(λ) and fast Q(λ)-learning take 48571 and 50684 steps, and run in 3 and 1 seconds, respectively. It can be seen that Q(λ)-learning takes much fewer steps than Q-learning and fast Q(λ)-learning is much faster than Q(λ)-learning.

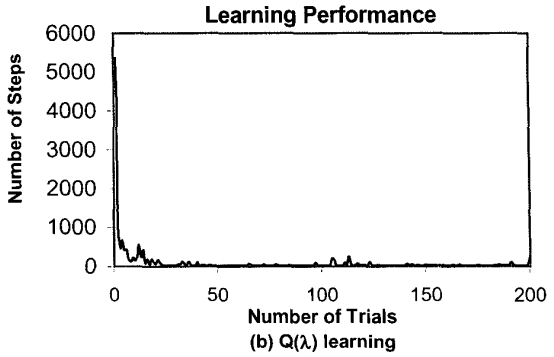
Figure 5(d) shows the learning curve of Q-learning incorporating DYNA planning. The length of the path found by the agent is 27. Each time an experience is inserted into the PQueue, we extract at most 10 experiences from the PQueue and use them to do the planning. The total number of steps in all 1000 trials is 34145. The algorithm runs in 3 seconds. It can be seen after using DYNA planning, the agent takes fewer total number of steps than it takes in Q and Q(λ)-learning

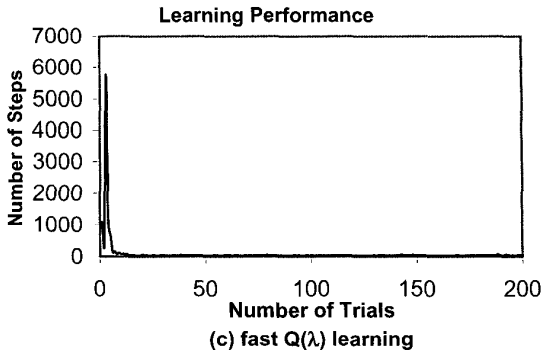
Figure 5(e) shows the learning curves of incorporating some environmental information into Q-learning. In this experiment, the agent doesn't visit a cell twice in a trial, and if the agent gets stuck during exploration, it doesn't give up the current trial. The length of the path found by the agent is 28. The total number of steps is 45368. The algorithm runs in less than 1 second. It can be seen that after using some environmental information, the agent takes fewer total number of steps than it takes in pure Q-learning.

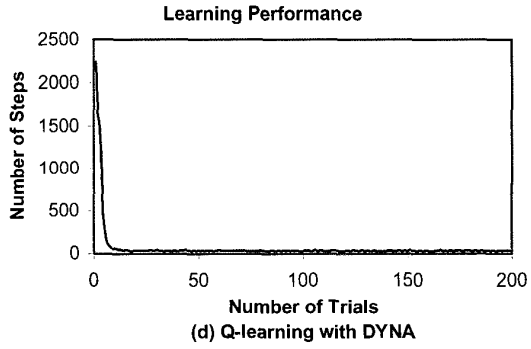
From the above curves, we can see that initially it takes the agent a large number of steps, because at that time, the agent has no knowledge about the maze; it can only explore the maze blindly. As learning proceeds and the agent gains more and more knowledge, the number of steps in a trial drops dramatically.

Although reinforcement learning can solve the search problem, it is expensive. It takes the agent many steps to find a good path, especially in the initial trials. It can be seen that Q-learning incorporated with DYNA or environmental information reduce about 50 percent of the number of steps taken by the agent. The combination of Q-learning and DYNA gave the best results.









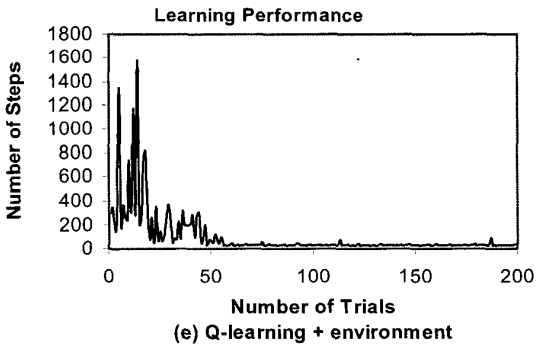


Figure 5. Learning performance for various algorithms.

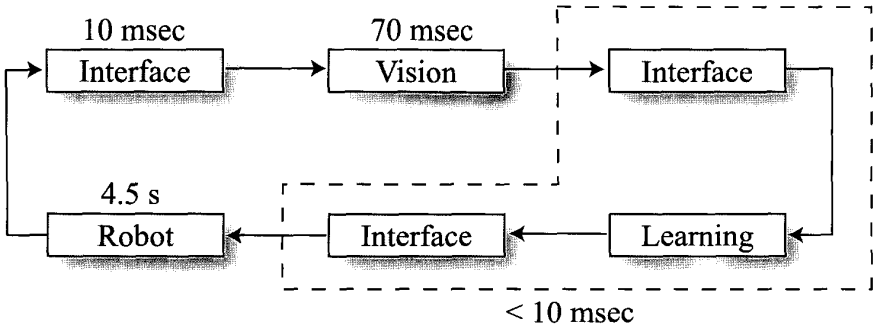


Figure 6. System-timing diagram.

4.2 Integrated Real-Time System

Figure 6 shows the time needed to complete one iteration of the learning process.

The results of the 15×15 maze with 28 obstacles are shown in Figure 7. The figure shows the decrement of actions taken as the number of trials increases. Before the first 8 trials, however, the number of actions taken oscillates tremendously, but as the number of trials increases, the graph begins to converge upon a set value. This value is the number of actions taken to produce the shortest path from start to goal.

In Figure 8, this same 15×15 maze result is graphed along with the results from a 6×6 maze with 12 obstacles. From the 6×6 maze, the results indicate the same type of oscillation found in the beginning trials but converges in later trials. 25 total trials were done for the 6×6 maze. After about 12 trials, the robot shows little oscillation in the number of actions taken and the values are at their minimum, meaning that the robot travels the shortest path nearly every time from start to goal.

In Figure 9, results from two 6×6 maze events were graphed against each other. The first event contained 6 obstacles while the second event contained 12 obstacles. Because the size of the maze was the same, similar results can be seen. In both events, the number of actions taken tends to fluctuate within roughly the same range. However, both experiments show that the 12-obstacle event tends to converge more quickly than the 6-obstacle event. With more obstacles on the maze, there are fewer cells for the robot to travel, and hence, it takes the robot a shorter time to go from the starting point to the goal. With fewer obstacles, there are more cells to explore, and so it requires the robot to take more actions.

Results for the Q and Q(λ) algorithms for a 6×6 maze are plotted against each other in Figure 10. From this graph, there appears to be few significant differences between the two learning algorithms. Both begin to converge upon the shortest path at approximately the same number of trials. Overall, the Q(λ) algorithm takes fewer actions during the entire experiment, which suggests that it is faster in finding the shortest path, but for a better comparison, more trials on a larger size maze should be done.

Q(λ)-learning was tested on a larger size maze of 15x15 with 28 obstacles. This experiment was conducted under the same parameters as the Q-Learning experiment shown in Figure 7, but with the Q(λ) algorithm. The results of this test are shown in Figure 11. The results show that in the Q(λ) algorithm, the number of steps in the later trials is lower than the Q-Learning algorithm. After 4 trials, the number of steps decreases below that of the Q-Learning algorithm. The number of steps shows a significant decrease after 10 trials.

Figure 12 shows two results of the Q(λ) experiments, one with 6 obstacles and the other with 12 obstacles. Similar to the results in Figure 9, these graphs show that the 12-obstacle experiment converges more quickly upon the shortest path than the 6-obstacle experiment, ascertaining the fact that with more obstacles, there are fewer cells to explore and hence, faster convergence upon the shortest path.

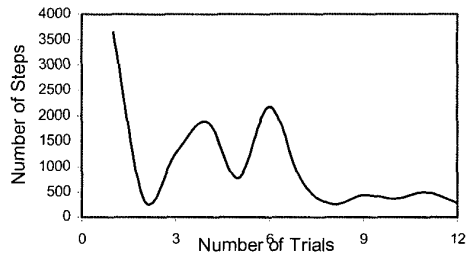


Figure 7. Q-learning results for 15x15 maze, 28 obstacles.

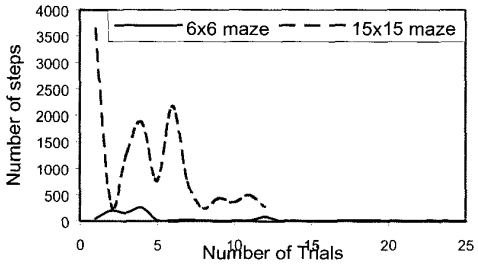


Figure 8. Q-learning results for different maze sizes.

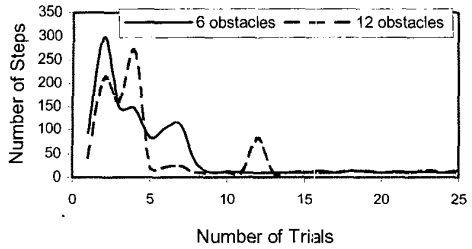


Figure 9. Q-learning results for 6x6 maze.

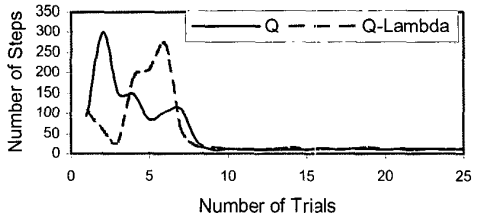


Figure 10. Results for 6x6 maze, 6 obstacles.

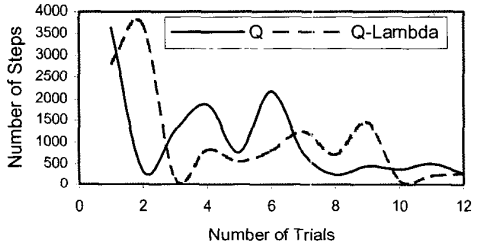


Figure 11. Results for 15x15 maze, 28 obstacles.

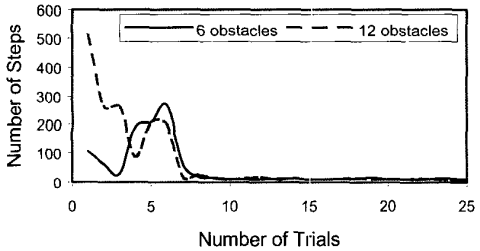


Figure 12. Q-lambda results for 6x6 maze.

5. Conclusions

In this paper, we have presented a systematic comparison of several reinforcement learning algorithms. We find that the fast Q(λ) and Q-learning with DYNA structure are the best. In the future, we plan to incorporate *a priori* task knowledge into these techniques and have hierarchical abstraction of the state space in a multi-agent framework for robotic applications.

• **Acknowledgments:** This work was supported by a NSF grant EIA-9610082-003. The authors would like to acknowledge the support they received from Roberto Carrillo, Vincent Hernandez, Tom Huyuh, Jing Peng, Michael Boshra, Sohail Nadimi, Stephanie Fonder, Grinnell Jones and James Harris.

6. References

[1] M. Asada et al., "Vision-based reinforcement learning for purposive behavior acquisition," *Proc IEEE Int. Conf. Robotics & Automation*, pp. 146-153, May 1995.

[2] B. Bhanu and Jing Peng, "Adaptive integrated image segmentation and object recognition," *IEEE Trans. on Systems, Man and Cybernetics*, Vol. 30, No. 4, pp. 427-441, November 2000.

[3] J. H. Connell and S. Mahadevan, *Robot Learning,* Kluwer Academic Publishers, 1993.

[4] H. R. Everett, *Sensors for Mobile Robots – Theory and Application.* A. K. Peters, Ltd, MA, 1995.

[5] G. Hailu, G. Sommer, "Embedding knowledge in reinforcement learning," *Proc. 8th Int. Conf. On Artificial Neural Networks*, pp. 1133-1138, September 1998.

[6] M. Huber, "A hybrid architecture for hierarchical reinforcement learning," *Proc. IEEE Int. Conf. On Robotics & Automation*, pp. 3290-3295, April 2000.

[7] J. Peng and R. J. Williams, "Incremental multi-step Q-learning," *Machine Learning,* vol. 22, No. 1-3, pp. 283 – 290, Kluwer Academic Publishers, January – March 1996.

[8] J. Peng and B. Bhanu, "Closed loop object recognition using reinforcement learning," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 20, No. 2, pp 139-154, February 1998.

[9] J. Peng and B. Bhanu, "Delayed reinforcement learning for adaptive image segmentation and feature extraction," *IEEE Trans. on System, Man and Cybernetics,* pp 482 – 488, August 1998.

[10] H. Suh, J. H. Kim and S. R. Oh, "Region based Q-learning for intelligent robot systems," *IEEE Int. Symp. on Computational Intelligence in Robotics & Automation*, pp. 172-178, July 1997.

[11] R. S. Sutton and A. G. Barto, *Reinforcement Learning*, MIT Press, 1998.

[12] M. Wiering and J. Schmidhuber, "Fast online Q(λ)," *Machine Learning*, Vol. 33, No. 1, pp. 105-115, October 1998.

[13] T. Yamagnchi et al., "Propagating learned behaviors from a virtual agent to a physical robot in reinforcement learning," *Proc. IEEE Int. Conf. On Evolutionary Computation*, pp. 855-859, May 1996.

[14] T. Yamagnchi et al., "Reinforcement learning for a real robot in a real environment," *European Conf. On Artificial Intelligence*, pp. 694-698, August 1996.