Developing Task Specific Sensing Strategies Using Reinforcement Learning

Srividhya Rajendran and Manfred Huber

Department of Computer Science Engineering
University of Texas at Arlington
Arlington, TX 76019-0015
{rajendra, huber} @ cse.uta.edu

Abstract

Robots that can adapt and perform multiple tasks promise to be a powerful tool with many applications. In order to achieve such robots, control systems have to be constructed that have the capability to handle real world situations. Robots use sensors to interact with the world. Processing the raw data from these sensors becomes computationally intractable in real time. This problem can be tackled by learning mechanisms for focus of attention. This paper presents an approach that considers focus of attention as a problem of selecting controller and feature pairs to be processed at any given point of time in order to optimize system performance. The result is a control and sensing policy that is task-specific and can adapt to real world situations using feedback from the world. The approach is illustrated using a number of different tasks in a blocks world domain.

Introduction

AI and robotics technologies have made significant strides but robots that are present today are still very task specific. To have robots that are more useful, it is necessary that they perform a range of tasks. Examples of such systems would be robots assisting in dangerous and/or repetitive tasks or that can assist elderly and handicapped people by monitoring their surroundings. These tasks require the robots to deal with real world situations. In order to interact with humans, to interpret the state of the world, and to represent continuous time, robots have to process the huge amount of data generated by their sensor modalities. Representing data of this magnitude (most of which is irrelevant) increases the complexity of the system and processing huge amounts of data in real time is computationally intractable. This requires that robots have effective mechanisms to extract the relevant data from the raw data pool.

Similar mechanisms are observed in biological systems which can not consciously process the huge amount of data they get from their sensor modalities. As a result they pay attention only to a small subset of their perceptual data while ignoring the rest. This mechanism is known as "Focus of Attention". Over time, while the system learns complex tasks, it develops successful task-specific sensing strategies depending on the resources available and the task at hand.

Robot systems require the ability to learn similar mechanisms of task-specific focus of attention to deal with the vast amount of potentially irrelevant data, and to perform complex tasks based on the available resources. The focus of attention problem mainly consists of two parts:

a. Knowing what things in the world could be important.
b. Knowing what things we need to pay attention to at a particular point in time.

In recent years, this school of thought has attracted many researchers to learn task-specific attention strategies. (McCallum 1996) developed mechanisms for learning selective attention for sequential tasks. Using the U-Tree algorithm this work dynamically builds a state space representation by pruning the large perceptual state space and augmenting the state space with short term memory containing the crucial features that were missing because of hidden state. This mechanism has a disadvantage that it does not perform well in continuous state spaces. (Laar, Heskes and Gielen 1997) learned task dependent focus of attention using neural networks. This idea used a limited sensor modality, constructed a special computational mechanism and did not have any real time feedback, thus limiting its use in real world robotic tasks. (Goncalves et. al 1999) presented an approach that uses similar learning mechanisms as presented here to identify objects' identities in an active stereo vision system. This approach introduced special purpose visual maps to accomplish the required task. (Piater 2001) presented an approach that learns a small number of highly useful task-specific features by "general-to-specific" random sampling. The system incrementally improves by dynamically updating the Bayesian network classifiers with information of highly distinctive features.

The approach to task-specific focus of attention presented here is aimed at finding out what we need to pay attention to given that we already know what things could be important. It considers focus of attention as a problem of selecting a set of features to be processed at any given point in time during task execution. It considers more than

one sensor modality for identifying the various object identities in the world. The set of features to be processed at any give time is determined on-line using reinforcement learning. This is done by tight integration of the sensing policy and the control policy of the robot. The learning algorithm here acquires a sensing and control policy simultaneously by selecting the set of features and associated control actions.

Technical Background

The approach presented here uses a control architecture based on hybrid discrete event dynamic system (Ramadge and Wonham 1989, Huber and Grupen 1997). The framework acts as an interface between the learning agent and the physical world. Figure 1 shows the basic organization of this framework.

The physical sensors and actuators are handled by the controller / feature pairs, the bottommost layer of this architecture. Here, each action that the robot system executes in the real world is associated with a set of features that form the control objective for that action. For example, action "*Reach*" "*Blue*" results in the robot arm reaching for a blue object. At each point in time the robot has to decide the relevant features to process in the context of the chosen action. This reduces the amount of raw data

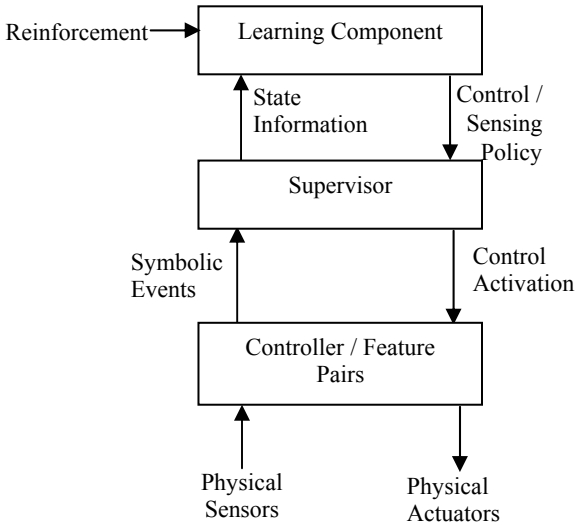

Figure 1. Organization of Control Architecture

that has to be analyzed to the one required for the related features. The convergence of controllers represents the completion of a control objective and results in the generation of a discrete symbolic event. This symbolic event is used by the supervisor to generate an abstract state.

State Space Representation

A robotic system with closed loop controllers handling a wide range of tasks has very complex state and action spaces. The exploration based technique of on-line learning of continuous control strategies becomes impractical on such systems.

This control architecture makes it possible to learn policies for such robots by using an abstract state space representation for the learning of control and sensing policies.

Given the set of control actions and the set of resources available in this system, only a finite set of closed loop responses are possible in the system. Each action here is associated with a set of features which dynamically define the objective of the action. The objective, in turn, defines the discrete states of the system. For example, "Reach" "Blue", represents the action "Reach" and the feature "Blue" defines a location in the world where the robot arm has to reach. As a result of this action the robot arm enters a state where it is at a blue object in the world. Each of the discrete states is defined by a vector of predicates. These predicates indicate the effects of action and feature pairs in the world.

The aim of the controller is to reach an equilibrium state where the control objective is satisfied, thus asserting the related predicates of the abstract state. However the outcomes of these control actions at the supervisor level are nondeterministic due to kinematic limitations, controller interactions and other nonlinearities in the system. Therefore it may happen that the same action with the same set of features to be monitored from state s may lead to a set of different states {s1,s2,s3} at different points in time. Hence the supervisor forms a nondeterministic finite automaton that triggers transitions with the convergence of controllers.

Reinforcement Learning

Q-learning is an effective reinforcement learning mechanism to learn control policies for agents that have no prior knowledge of the world and receive feedback about their actions only in the form of delayed reward. However, reinforcement learning does not perform well for highly complex tasks and in continuous state spaces because of the size of the search space. In the approach presented here this problem is addressed at the supervisor level by using abstract states and closed loop action, resulting in a reduced search space.

In Q-learning the agent interacts with the world over a period of time. At each time step, t, the agent observes a state s_t, chooses an action a_t, and executes it. As a result the world reaches a new state s_{t+1} and gives a reward r_t to the agent. This information is used by the agent to learn a policy that maximizes the expected reward over time.

The Q-function (Watkins 1989) $Q : S \times A \rightarrow \Re$ represents the expected discounted sum of future rewards if action *a* is taken from state *s* and the optimal policy is followed thereafter. The Q value for the state-action pairs is learned iteratively through on-line exploration. Each time a state-action pair is visited its value is updated using:

$$Q(s,a)' = Q(s,a) + \alpha(r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a))$$

Q(s, a)' is the updated value of Q(s, a), r is the reward for reaching the new state s' by selecting action a in state s, α ($0 \leq \alpha < 1$) is the learning rate, and γ ($0 \leq \gamma < 1$) is a constant that represents the relative value of delayed versus immediate rewards. Upon convergence of the algorithm the optimal action in each state s can be extracted from the value function as:

$$a_{max} = \arg\max_{b \in A} Q(s,b)$$

Convergence of this algorithm is based on a few basic assumptions:

1. The problem is Markovian.
2. Each state/action pair is updated infinitely often.
3. There exists some positive constant *c* such that for all states *s* and actions *a* $|r(s,a)| < c$.

During learning it also requires exploration versus exploitation strategies to allow the agent to explore when it has no knowledge of the world and exploit more when it does have knowledge of the world.

In the current architecture, Q-learning is used. Each time a new state is reached, the world gives feedback in the form of reinforcement to the learning component. This, in turn, is used to update the state-action pair in the abstract state space (a vector of predicates derived from symbolic events that represent whether the control objective defined by controller /features pairs is achieved or not).

Experiments

To illustrate the proposed approach, let us consider examples in the blocks world domain where the robot has to interact with objects on the table top. Our robot configuration consists of a robot arm, a stereo vision system, and feature extraction algorithms to identify visual features of the objects in the world, such as color, shape, size and texture. Through interaction with the blocks world the robot learns a task-specific control and sensing policy in order to optimize the system performance for the given task.

The robot can perform the following actions:

1. "*Reach*": This action is used by the robot arm to reach for an object at any given location within the boundaries of the blocks world.
2. "*Pick*": This action is used to pick or drop an object at the current location.
3. "*Stop*": This action allows the robot to stop its interaction with the blocks world. This action is mainly used since

    a. The robot does not know what the task is and learns it from the reinforcements provided by the world.

    b. There are no absorbing states in the real world. So, the robot has to learn when to stop performing a task in order to maximize expected reward.

Each "*Reach*" or "*Pick*" action is associated with a set of features that the robot has to process in order to derive the goal of the action and then to complete the given task. For example, "*Reach*" "*Blue*" will cause the robot arm to reach for a blue object in the blocks world if such an object exists.

As the number of features present in the world increases, the complexity of the system also increases. In order to restrict the system complexity, the number of features that can be processed with each action at a given point in time is here limited to two.

The Q-learning algorithm uses an abstract state space to learn the sensing and control policy. Each abstract state can be represented as a vector of predicates. In the blocks world domain, the vector of predicates constituting an abstract state consists of:

- Action Taken: This indicates what action was taken
- Action Outcome: This indicates if the action was successful or unsuccessful.
- Feature 1 and / or Feature 2: These are the features that were chosen by the robot to determine the control objective of the last action.
- Feature Successful / Unsuccessful: This represents whether the feature combination used by the last action was found.
- Arm Holding: This indicates what the arm is holding.

For example: if the robot is in the start state s_0 { "*Null*", "*Null*", "*Null*", "*Null*", "*Null*", "*Holding Nothing*"} and takes action "*Reach*" and the features that formed the control objectives were color "*Blue*" and shape "*Square*" , then the robot tries to reach for an object that has color "*Blue*" and has a "*Square*" shape. If the world contains an object with this feature combination then this action will lead to a new state and the vector of predicates for this state will have the value {"*Reach*", "*Successful*", "*Blue*", "*Square*", "Successful", "*Holding Nothing*"} meaning that the action "*Reach*" for an object with features "*Blue*" and "*Square*" was *successful* and the feature combination "*Blue*" and "*Square*" was found. By the end of this action the arm is "*Holding Nothing*".

Following are example tasks that were learned in the blocks world domain.

Table Cleaning Task

In this experiment there are a number of objects on the table top and the robot can move or pick up only one object at a time. The task is to learn a sensing and control policy that will allow the robot to reach, and pick up objects (that are movable) from the table, and then reach and drop these objects in a box. While learning a policy for the task, the robot also has to learn when to stop since

there is no explicit information or feedback as to when the task is completed.

The robot has a cost associated with each action it takes and receives a small positive reward each time it picks up and drops an object from the table into the box.

The blocks world has the following objects and features present on the table (as shown in Figure 2)

1. Object 1: Texture 1, Square, Small.

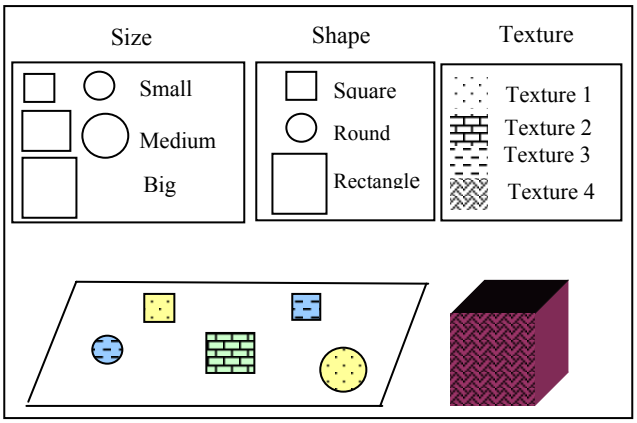

Figure 2. Table Cleaning Task Setup

2. Object 2: Texture 1, Round, Medium.
3. Object 3: Texture 2, Square, Medium.
4. Object 4: Texture 3, Round, Small.
5. Object 5: Texture 3, Square, Small.

and a box (Texture 4, Rectangle, Big) in which all the objects are to be dropped. All the objects on table are movable or unmovable. Whether an object is movable or unmovable can only be determined by trying to pick it up at least once. The objects with size "*Big*" in world can not be picked up by the robot arm.

Once a particular object (suppose with feature "Texture 1") is dropped in the box it is no longer visible. If the robot in state s_x {"*Pick*", "*Unsuccessful*", "*Texture 4*", "*Null*", "*Successful*", "*Holding Nothing*"} again tries to reach for an object with feature "Texture 1". Then the "*Reach*" action will only be successful if there is another object with "Texture 1" on the table. Otherwise it will be unsuccessful since the features of the dropped object are no longer accessible to the feature extraction algorithm.

Starting from the derived abstract state representation, the system uses the learning component to learn the value function using the reward it receives each time a transition from state s_t to s_{t+1} takes place. The robot starts out exploring the world by taking random actions (100 % exploration) and incrementally decreases its exploration using the Boltzmann "soft-max" distribution until it reaches a stage where the exploration is about 10 %. This amount of exploration is maintained to allow the system to visit different potentially new parts of the abstract state space even after it achieves a locally optimal policy. This is done in order to improve its performance by enabling it to learn a more global strategy.

For the table cleaning task, 3 possible cases were considered:

1. *All objects on the table are movable*
2. *Not All objects on table are movable*: A few objects on the table are movable and a few are unmovable.
3. *None of the objects on the table are movable*.

Case 1. Figure 3 shows the Learning curve for the table cleaning task when all objects on the table are movable.

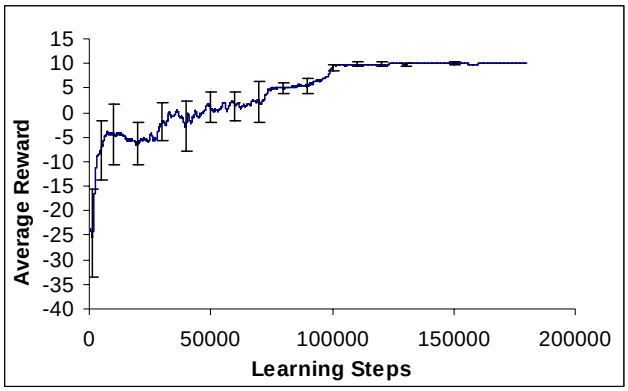

Figure 3. Learning Curve Case 1

The learning curve is plotted as a running average over 30 steps and depicts the average of 10 learning trials. The intervals indicate one standard deviation.

State	Predicate Values
S_0	{"Null", "Null", "Null", "Null", "Null", "Holding Nothing"}
S_1	{"Reach", "Success", "Small", "Null", "Success", "Holding Nothing" }
S_2	{"Pick", "Success", "Small", "Null" "Success", "Holding Small"}
S_3	{"Reach", "Success", "Texture 4", "Null", "Success", "Holding Small" }
S_4	{"Pick", "Unsuccessful", "Texture 4", "Null", "Success", "Holding Nothing"}
S_5	{"Reach", "Unsuccessful" "Small", "Null", "Unsuccessful", "Holding Nothing"}
S_{11}	{"Stop, "Successful" "Null", "Null", "Null", "Holding Nothing"}

Table 1. States and Predicate Values

Figure 4 shows a part of the control policy learned by the robot for case 1 of the table cleaning task. Table 1 shows the states and the predicate values for some of the states shown in Figure 4. Each arrow in the figure represents a possible result of the related action in terms of a transition from the old state to the new state of the world. The robot starts in state S_0 and takes actions "Reach" "Small" to reach the object with size "Small" on the table, leading to a transition from state S_0 to S_1. If there is more than one object with size "Small", then it randomly reaches for any

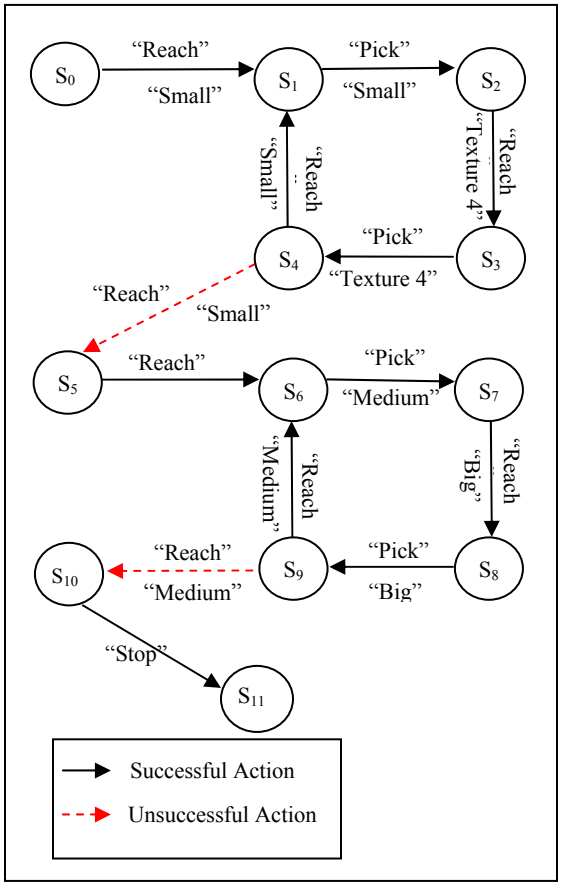

Figure 4. Partially Learned Policy for Case 1

one of those objects since the objects with the same size look similar. Then, from state S_1 it takes action "*Pick*" "*Small*", resulting in successfully picking up the object with size "*Small*" , leading to state S_2.The arm then reaches for the box where it needs to drop the object using action "*Reach*" "*Texture 4*", leading to state S_3. Finally, taking the action "*Pick*" "*Texture 4*" it drops the held object into the box, thus reaching state S_4. From state S_4 it again tries action "*Reach*" "*Small*" and if there are more objects with size "*Small*" then the states S_1, S_2, S_3, S_4 are repeated until all objects of size "*Small*" are dropped into the box. Once all the objects of size "*Small*" are dropped, taking action "*Reach*" "*Small*" results in a transition to a state S_5. This transition tells the robot that there are no more objects with size "*Small*" on the table and thus helps the robot to shift its attention from the current feature "S*mall*" to some other feature that can be used as a control objective for the actions to further clean the table.

Once the table is clean all the features except those in the box are unsuccessful. The robot learns that taking any more actions in the blocks world domain once the table is clean results in no reward.

This causes the robot to learn that "Stop" is the best action once the table is clean, thus maximizing expected reward for this task.

Case 2. In this case the blocks world contains the same number of objects as in case 1, however objects 2 and 3 on the table are not movable. Figure 5 shows the learning curve for this task.

In this case the robot learns a policy that results in the robot picking and dropping all the movable objects into the box. It learns to stop once all movable objects are dropped in the box, since it receives no reward for moving and trying to pick up the unmovable objects. The result is a partial cleaning of the table.

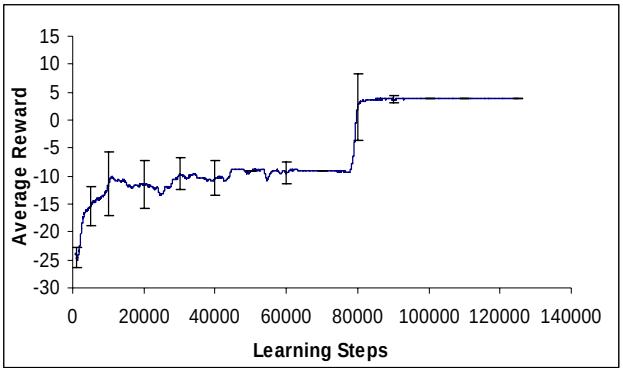

Figure 5. Learning Curve Case 2

Case 3. Figure 6 shows the learning curve for this task when none of the objects on the table are movable.

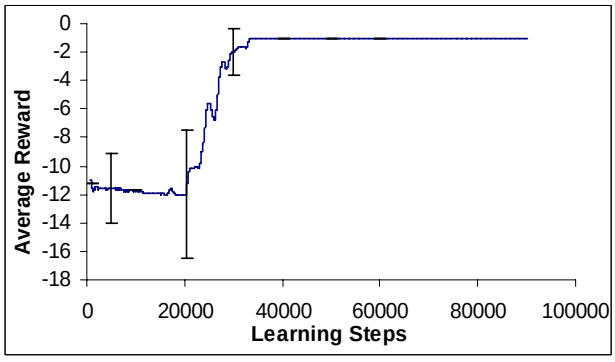

Figure 6. Learning Curve Case 3

In this case the robot starts out by randomly exploring the world and soon learns that none of the objects on the table are movable. Thus, it learns to stop immediately instead of taking any other actions, since in this case the table can never be cleaned.

Sorting Task

In this task the robot has to learn a policy for sorting objects on the table into different boxes.

The blocks world has the following objects in the sorting task (shown in Figure 7):

1. Object 1: Texture 1, Square, Small.
2. Object 2: Texture 1, Round, Medium.

and a box 1 (Texture 3, Rectangle, Big), box 2 (Texture 2, Rectangle, Big) in which all the objects are to be dropped.

The robot gets a reward when it drops

a. an object having features "Texture 1" and "Small" into box 1.
b. an object having features "Texture 1" and "Medium" size into box 2.

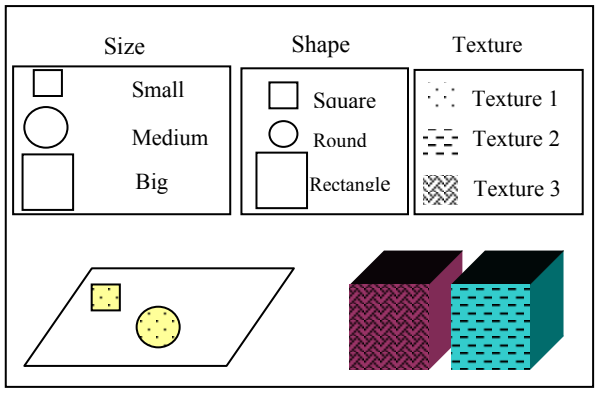

Figure 7. Sorting Task Setup

Figure 8 shows the learning curve for this task.

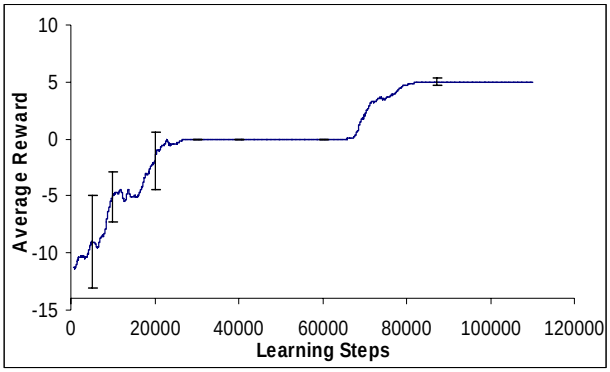

Figure 8. Learning Curve for Sorting Task

The robot successfully learns a policy that sorts the objects with features "Texture 1" and "Small" into box 1, and other objects into box 2, and stops once all the objects on the table are sorted.

In the above experiments the learning process requires between 50,000 and 200,000 steps to converge to an optimal policy. While this number might make large tasks intractable if all learning steps have to be executed by the robot (as was done in the experiments), the number of physical actions required could be substantially reduced by using real world experiences to build a model which, in turn, can be used for off-line learning steps to adjust the policy (Sutton 1990, 1991, Moore 1993).

Conclusions and Future Work

The results presented here illustrate that robots can learn task-specific control and sensing policies. This is achieved by using a mechanism for focus of attention that ties actions to perceptual features that form their control objective. This reduces the search space required by an online reinforcement learning algorithm to learn a policy. Here it was assumed that the next state only depends on the current percepts and actions, and not on any previous percepts. But in the real world there are many tasks which cannot be completed unless knowledge about past events is available. To address this we are currently extending the approach by incorporating short term memory to remember past events. In order for the short term memory to work effectively the robot not only has to learn control and sensing policies, but also a memory policy. The memory policy tells the robot what to remember and when to remember in order to successfully complete a task.

Acknowledgements

This work was supported in part by UTA REP-14748719

References

Andrew McCallum R. (1996). Hidden state and reinforcement learning with instance-based state identification. *IEEE Transactions on Systems, Man and Cybernetics*, 26B (3):464 -473.

Goncalves L.M.; Giraldi G.N.; Oliveira A.A. and Grupen R.A. (1999). Learning Policies for Attentional Control. *IEEE International Symposium on Computational Intelligence on Robotics and Automation (CIRA* 1999).

Huber M. and Grupen R.A. (1997). "A Feedback Control Structure for On-line Learning Tasks", *Robotics and Autonomous Systems*, Volume 22, Issues 3-4.

Andrew W. Moore and Christopher G. Atkeson (1993). Prioritized sweeping: Reinforcement Learning with less data and less real time. *Machine Learning* (13).

Justus H. Piater (2001). Visual Feature Learning. Ph.D. dissertation, Dept. of Computer Science, Univ. of Massachusetts, Amherst.

Peter J.G. Ramadge and W. Murray Wonham (1989). The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81-97.

Richard S. Sutton (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. *In Proceedings of the Seventh International Conference on Machine Learning*, Austin, TX.

Richard S. Sutton (1991). Planning by incremental dynamic programming. *In Proceedings of the Eighth International Workshop on Machine Learning*, 353-357.

Van de Laar P.; Heskes T. and Gielen S. (1997). Task-Dependent Learning of Attention. *Neural Networks*, 10(6):981-992.

Watkins, C. J. C. H. (1989). Learning from delayed rewards. Ph.D. thesis, Psychology Department, Univ. of Cambridge.