LEARNING TASK-SPECIFIC MEMORY POLICIES

Srividhya Rajendran and Manfred Huber
Department of Computer Science and Engineering
University of Texas at Arlington
Arlington, TX-76019
vidhya@ailab.uta.edu, huber@cse.uta.edu

ABSTRACT

Effective AI agents and robots require the ability to adapt to real world situations and perform multiple tasks. This requires them to take into account the important sensory information. Extraction of this information can be made tractable using mechanisms of focus of attention that select perceptual features that have to be processed. This mechanism alone however is inadequate for tasks in real world situations since it still requires the robot to maintain all past information, rendering decision making computationally intractable. This requires the robots and AI agents to have the capability to remember only the past events that are required for successful completion of a task. Here we present an approach (illustrated using block stacking and block copying tasks) that extends a previous focus of attention mechanism by incorporating short term memory to remember past events. The result is a task-specific control, sensing, and memory policy.

KEY WORDS

Focus of Attention, Memory Policies, Robots, AI agents

1. Introduction

Science and technology advances have led to many new pieces of equipment that are used in day to day life. Robots, however, while having a great potential are yet to become popular. One of the reasons is that robots today tend be very task specific and use very simplistic strategies. In order for them to be more useful, they require capabilities to deal with real world situations and should also have the ability to handle multiple tasks. Robots process the data generated by sensor modalities to deal with real world situations, to interact with humans, and to interpret the state of the world. Representing and processing the huge amount of raw data generated by the sensor modalities in real time increases the complexity of the system. Biological systems receive a similar magnitude of data from their sensor modalities. Instead of processing all the data, however, they pay attention to and process only the small subset of their perceptual data that is significant while ignoring the rest. For example, the task of ″Navigation″ requires us to look out for charging animals if we are navigating through a jungle but requires looking out for vehicles if we are navigating through a city. As a result, even though task is the same (″Navigation″), the sensing strategies depend heavily on the situation and on the resources available (e.g. sensing strategies for this task will be different for a blind person and a person with eyesight). To address this biological systems develop task and resource specific strategies for the efficient perception of significant properties. Robots and AI agents can develop similar mechanisms of task-specific focus of attention to deal with the large amount of irrelevant data by focusing on the relevant set of features at any given point in time during task execution [1] [2].

Humans further enhance the mechanism of task-specific focus of attention with short term memory. Short term memory holds information for short amounts of time after which it decays leading to complete loss of the information unless it is further reinforced by rehearsals or repetitions. This reflects the cognitive and perceptual operation the subject is currently involved in [3].The memory allows humans to retain information about past events [4] that are no longer available in the physical world. Remembering significant, task-specific information allows them to complete complex tasks successfully. Experiments have shown that humans have difficulty remembering more than 5 ± 2 novel items. Even this limited amount of short term memory, however, allows them to perform highly complex tasks. For example, while doing a block copying task [5] where a person has to build a copy of the model in the workspace area using blocks in the source area, a person can completely memorize and replicate a stack of blocks only for very small tasks (2-3 blocks) and even in these situations this does not represent a commonly used strategy due to limitations on remembering the block colors and their relations at once. Instead, it was observed [5] that the subjects constantly refer back to the model throughout the execution of the task. It was further observed that subjects had the tendency to break up the tasks into subtasks of single block moves. This shows that humans, instead of memorizing the configuration to be copied in its entirety before moving the blocks (which becomes computationally difficult), divided the task into subtasks and used strategies involving the use of minimal memory. Robots and AI agents, apart from having mechanisms of focus of attention, also require the ability to remember past events. But processing all past events to complete a task is still intractable. Consequently robots and AI agents also require a memory management mechanism that determines what events are significant for task completion and when to remember them.

In recent years some research has been performed to develop algorithms to learn tasks in partially observable environments. Some of them simply ignore the memory issues by focusing on suboptimal, memory-free solutions to partially observable problems [6]. Others use fixed size window approaches [7] to try and resolve the hidden state problem by choosing actions that depend not only on the current observations but also on a fixed number of the most recent observations and actions. These approaches fail to learn a policy when the relevant information falls outside the history window. Other approaches find optimal solutions in partially observable environments [8] [9] but are limited to very small problems and do not perform well in continuous state spaces [10] [11].

This paper presents an approach that is aimed at enhancing a task-specific focus of attention mechanism [1] by adding limited short term memory and learning a memory policy. The learned memory policy tells the robot what past events are significant for task completion and when to remember them in order to successfully complete a task.

2. Control Architecture

The control architecture used here is based on hybrid discrete event dynamic systems [12] [13] [14]. Figure 1 shows a diagram of the control architecture used.

The controller/feature pair component of this architecture directly deals with the environment. This component uses physical sensors to interpret the state of the world and physical actuators to modify it. The activation signal given by the supervisor to the physical sensors and actuators is determined by the learning component. Each action executed in the real world is associated with a set of features that define the functional objective of the action. For example, action "*Reach*" "*Blue*", uses the feature "*Blue*" to form the functional objective of the action reach. This allows the robot to try and reach for a blue coloured object. The convergence of controllers results in the completion of a control objective. This completion, in turn, generates a discrete symbolic event. The supervisor uses this discrete event along with the information in the event memory to generate an abstract state.

2.1 Action Representations

The robot interacts with the world through its control actions. Each control action is associated with a set of features that the robot has to process in order to derive the functional objective of the action. At each point in time the robot has to decide the relevant features to process in the context of the chosen action. This limits the amount of raw data to be analyzed to the data required for the selected features. The convergence of controllers represents the completion of a control objective and results in the generation of a discrete symbolic event.

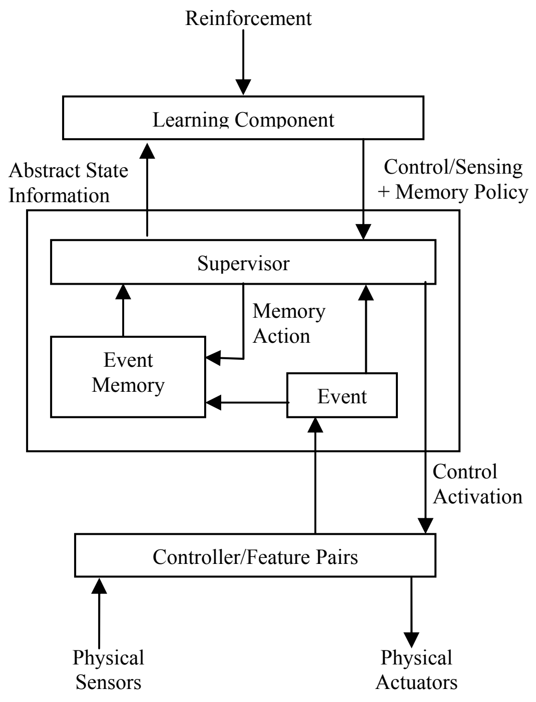

Figure 1: Control Architecture

In the blocks world domain examples used throughout this paper the robot configuration consists of a stereo vision system, an arm, short term memory cells (to remember past events), and a feature extraction algorithm to identify the visual features of the blocks in the world. The robot arm can execute 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. "*Top*": This action is used to find out if the block with feature "y" is on top of the block with feature "x" when the robot is focusing on block with feature "x". For example, if the robot takes action "*Top*" "*Blue*" "*Yellow*", this action is successful if the robot is focusing on a blue block and there is a yellow block on top of the blue block.
4. "*Bottom*": This action is used to find out if the block with feature "y" is at the bottom of the block with feature "x" when the robot is focusing on block with feature "x".
5. "*Stop*": This action permits the robot to stop its interaction with the blocks world. This action is necessary because :
   1. The robot does not know what task it is learning and uses the reinforcements given by the world to learn a policy.
   2. 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 of the actions "*Reach*", "*Pick*", "*Top*", and “*Bottom*" can be associated with multiple features that form its target objective. In the experiments presented here we limit the number features associated with an action to two in order to restrict the complexity of the system. Apart from these actions the robot can also execute memory actions. A memory action cannot be executed by itself. It always has to be associated with a "*Reach*", "*Pick*", "*Top*", or "*Bottom*" action. Associating a memory action with a control action allows the robot to remember the effect of the action on the world in short term memory.

2.2 State Space Representations

In order for the robot to be able to handle multiple complex tasks in real time, and at the sensory-motor level, it has to handle continuous space and time. Representation of such a complex state and action space however poses a serious issue. Further, learning a policy using an exploration based technique becomes easily impractical. To address this, the state space has to be modeled at a different level of abstraction, thus reducing the size of the search space used by the learning component of the control architecture to acquire a policy.

The state space, S, in this control architecture is modeled at an abstract level. Where each abstract state, S_i, consists of two parts:

1. Current Symbolic Event, E_i
2. Event Memory $M_{i_1},\ldots\ldots,M_{i_n}$

Formally, the state space, S, is defined as $S = E \times M^n$, where E is the set of symbolic events, M is the set of memory events, and n is the number of memory cells.

Current Symbolic Event

The symbolic event component of the abstract state is represented as a vector of predicates. Each of these predicates indicates whether the equilibrium state of the controller meets the control objective. For example, action "*Reach*" "*Blue*" defines a location in the world that the robot arm has to reach. Upon convergence of controllers the predicates of the event indicate whether the robot arm has successfully reached for the blue object in the world or has failed. At this level of abstraction the overall behaviour of the system can be characterised by the effects of each controller on the discrete predicate space. However, outcomes of these control actions are non-deterministic due to kinematic limitations, controller interactions and other non-linearities in the system. The symbolic events consist of the following predicates:

1. Action Taken: This indicates what action was taken.
2. Action Outcome: This indicates if the action was successful or unsuccessful.
   Feature 1 and/or Feature 2: This indicates the features that define the target location in blocks world domain. For example, action *"Reach" "Blue" "White"* will result in the arm reaching for a blue block with white background. The action is successful if the blocks world domain contains a blue block in front of a white background.
3. Feature Combination Outcome (successful/ unsuccessful): This indicates whether the color combination used by the last action was found or not.
4. Arm Holding: This indicates if the arm is holding an object.

Event Memory

The event memory component of the abstract state represents the past events that are considered significant for task completion by the robot. In order for a robot to actually remember an event that occurred as a result of executing a control action, the robot has to associate a memory action with the control action. The memory action allows the robot to remember the current event in the event memory. Once an event is stored in the event memory, it remains in the memory until it is explicitly replaced by some other event. The predicates that constitute the memory part of the abstract state consist of:

1. Action Taken.
2. Action Outcome.
3. Feature 1 and/or Feature 2.

These predicates indicate what action was taken, the features that formed the target objective of the action, and the outcome of the action. For example, if a robot in state S_t takes the action *"Reach" "Blue" "White"* + *"Memory"*, it reaches a new state with the predicates {*"Reach"*, *"Successful"*, *"Blue"*, *"White"*, *"Successful"*, *"Holding Nothing"*}, meaning that the robot has successfully reached for the blue block with white background, that the feature combination *"Blue"*, *"White"* was successfully found and that the robot arm is currently holding nothing. Further, the outcome of this action is remembered in short term memory. The predicate values of short term memory after the execution of this action are {*"Reach"*, *"Successful"*, *"Blue"*, *"White"*}.

2.3 Q-Learning

Q-learning [15] [16] is a reinforcement learning algorithm that learns a model-free optimal policy for an agent by estimating the values of state-action pairs using feedback from the environment in the form of delayed reward.

Let the agent be in state S_t at time t. The agent chooses an action A_t from state S_t. As a result the agent receives a reward R_t and reaches a new state $S_{t'}$. This information is used by the agent to learn to choose actions maximizing discounted cumulative rewards over time. The value $Q(S_t, A_t)$ is defined as the expected discounted sum of future payoffs obtained by taking action A_t from state S_t and following an optimal policy thereafter. The Q value of the state-action pairs is learned iteratively through on-line exploration. Each time a state-action pair (S_t, A_t) is selected its value is updated according to:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_t + \gamma[\max_{A_{t'} \in A} Q(S_{t'}, A_{t'}) - Q(S_t, A_t)])$$

where $\gamma = 0.9\ (0 \le \gamma \le 1)$ is the discount factor, and $\alpha = 0.2\ (0 \le \alpha \le 1)$ is the learning rate parameter. Upon convergence of the algorithm the optimal action from any state is the one with the highest Q value. In order for the agent to learn an optimal policy it should follow an exploration-exploitation strategy. This strategy allows the agent to explore when it has no knowledge of the environment and exploit when it gained knowledge of the correct actions. In this paper Q-learning uses the Boltzmann "softmax" [17] strategy since it ensures sufficient exploration while still favouring actions with higher value estimates. Though Q-learning allows learning accurate value functions, it does generally not perform well on complex tasks and continuous state spaces because the size of the state space grows exponentially with the number of state variables making learning of a complete value function unwieldy. This problem is addressed in this paper at the supervisor level of the control architecture. The supervisor uses an abstract state space and closed loop actions, resulting in a reduced search space. Each time a new state is reached, the feedback in the form of reinforcements is used to update the state-action pair in the abstract state space.

3. Experiments and Results

To illustrate the approach proposed in this paper, two tasks in the blocks world domain are considered:

1. Block Copying Task
2. Block Stacking Task

The main objective of the robot in these experiments is to learn the task-specific control, sensing, and memory policies in order to optimize the system performance for the given task.

In all experiments the robot has no prior knowledge of the environment or the task it has to learn. Further it is assumed that all blocks are uniquely identifiable by their colour in this blocks world domain.

3.1 Block Copying Task

In this experiment, two tables with similar set of blocks are present. Table1 has an already built stack (using the blocks on table1). The task of the robot is to learn to copy any stack presented on table1 onto table2. Two scenarios of the block copying task are considered:

1. Block Copying Task with Two Blocks
2. Block Copying Task with Three Blocks

In both tasks, the robot has a cost -1 associated with each action and receives a reward of +10 each time two blocks are stacked correctly.

Block Copying Task with Two Blocks

Figure 2 shows the initial and final situations of a successful strategy for an instance of the block copying task with two blocks and Table 1 shows the two different configurations possible with the two blocks used. In this task, the robot has to learn a policy that will allow it to copy any of the two possible stacks using the two blocks present on the table top. While learning a policy for the task, the robot also has to learn when to stop since there is no explicit information as to when the task is completed.

The robot starts out by exploring the world and uses the feedback to learn a policy for the copying task.

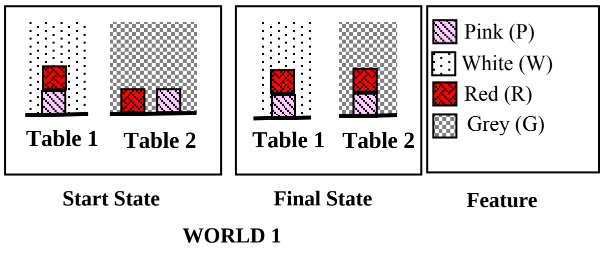

Figure 2: Block Copying Task with Two Blocks

World No.	Top Block	Bottom Block
1	"*Red*"	"*Pink*"
2	"*Pink*"	"*Red*"

Table 1: Different world configurations using "*Red*" and "*Pink*" blocks

Figure 3 shows the learning curve for the block copying task with two blocks. The graph shows that the system learns a policy that is general enough to copy each of stacks when it is presented.

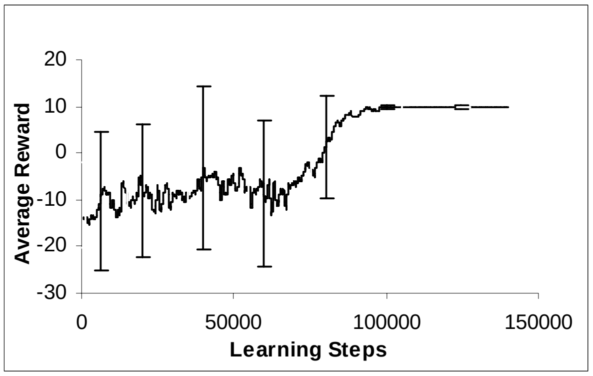

Figure 3: Learning Curve for Block Copying Task with Two Blocks

Figure 4 shows a policy learned by the robot for the block copying task with two blocks. Table 2 shows the different symbolic events and memory events in this learned policy. Each arrow in Figure 4 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 $\{E_0,M_0,M_0\}$ and learns to copy any stack presented on table1 by reaching for the "*Pink*" block with "*White*" background, finding out if it is the bottommost block on table1 and remembering the outcome of this action in memory. The memory event enables the robot to identify which stack it has to build on table2. For example, if the memory event is M_1 then it knows that pink is the bottommost block in the stack on table1 and thus the robot reaches for the red block on table2, picks it up and drops it on top of the pink block. Once the blocks are stacked, any further exploration of the environment by the robot does not earn it any rewards, enabling it to *"Stop"* for maximizing the reward.

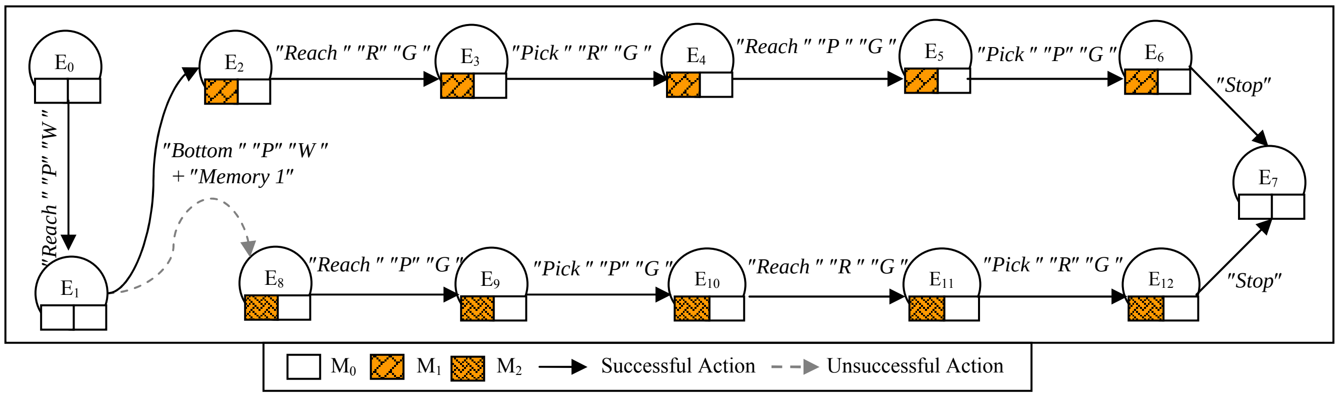

Figure 4: One of the Learned Policies for Block Copying Task with Two Blocks (memory

Event	
E_0	{*"Null", "Null", "Null", "Null", "Null", "HN"*}
E_1	{*"Reach", "S", "Pink", "White", "S", "HN"*}
E_2	{*"Bottom", "S", "Pink", "White", "S", "HN"*}
E_3	{*"Reach", "S", "Red", "Grey", "S", "HN"*}
E_4	{*"Pick", "S", "Red", "Grey", "S", "Holding Red Grey"*}
E_5	{*"Reach", "S", "Pink", "Grey", "S", "Holding Red Grey"*}
E_6	{*"Pick", "S", "Pink", "Grey", "S", "HN"*}
E_7	{*"Stop", "S", "Null", "Null", "Null", "HN"*}
E_8	{*"Bottom", " Unsuc ", "Pink", "White", "S", "HN"*}
E_9	{*"Reach", "S", "Pink", "Grey", "S", "HN"*}
E_{10}	{*"Pick", "S", "Pink", "Grey", "S", "Holding Pink Grey"*}
E_{11}	{*"Reach", "S", "Red", "Grey", "S"," Holding Pink Grey"*}
E_{12}	{*"Pick", "S", "Red", "Grey", "S", "HN"*}

Memory	
M_0	{*"Null", "Null", "Null" , "Null"*}
M_1	{*"Bottom", "S", "Pink", "White"*}
M_2	{*"Bottom", "Unsuc", "Pink", "White"*}

"HN" – *Holding Nothing*, *"S"*– *"Successful"*, *"Unsuc"*– *"Unsuccessful"*

Table 2: Event and Memory Values for a Learned Policy (Block Copying Task with Two Blocks)

Block Copying Task with Three Blocks

To illustrate the power of selective memory policies, a second experiment with stacks of three blocks was performed. Table 3 shows the different configurations possible using "*Red*", "*Blue*" and "*Pink*" blocks.

World No.	Top Block	Middle Block	Bottom Block
1	*"Blue"*	*"Red"*	*"Pink"*
2	*"Blue"*	*"Pink"*	*"Red"*
3	*"Red"*	*"Blue"*	*"Pink"*
4	*"Red"*	*"Pink"*	*"Blue"*
5	*"Pink"*	*"Red"*	*"Blue"*
6	*"Pink"*	*"Blue"*	*"Red"*

Table 3: Different world configurations using "*Red*", "*Blue*" and "*Pink*" blocks

Figure 5 shows the block copying task's start and final states for world1. Here the robot tries to learn with and without a memory policy using the following setups:

1. Setup with memory policy: the robot explicitly learns a memory policy to remember past events required for task completion.
2. Setup without memory policy: the robot does not learn a memory policy but maintains the information of the last two events.

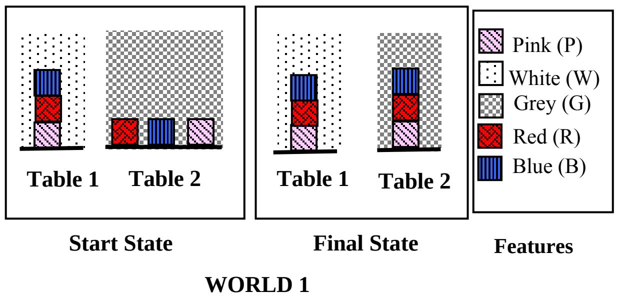

Figure 5: Block Copying Task with Three Blocks

Figure 6 shows the learning curve for the copying task with and without a memory policy. The learning curves here show that as the task becomes more complex (e.g. copying task), the robot with a memory policy performs better than the one with a fixed size history window. The robot that tries to learn the copying task without a memory policy learns to "Stop" because remembering only the past two events does not allow it to complete the task. On the other hand, the robot with a memory policy is able to complete the task successfully.

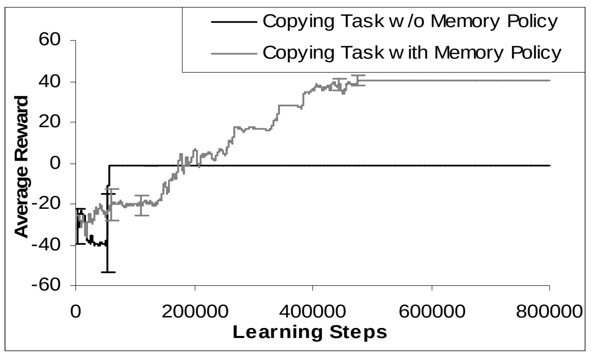

Figure 6: Learning Curve for Block Copying Task

3.2 Block Stacking Task

An additional experiment was performed to investigate the cost of learning memory policies. In this experiment, the task of the robot is to learn to build a fixed stack of blocks. Figure 7 shows the start and final configuration for this block stacking task.

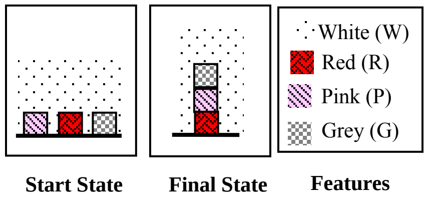

Figure 7: Block Stacking Task

Figure 8 shows the learning curves for the stacking task with and without a memory policy. The learning curves show that the robot learns to complete the task successfully in both cases. This is because it has to remember only the last block stacked before reaching for

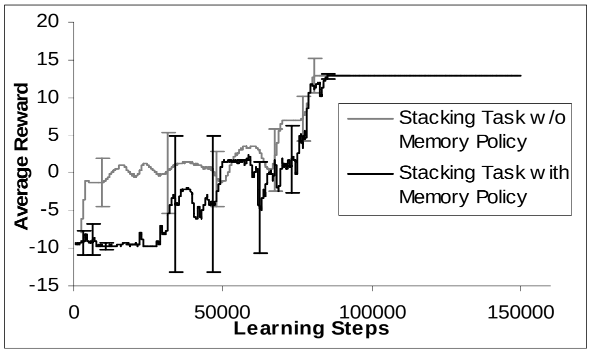

Figure 8: Learning Curve for Block Stacking Task

and picking up the next object to stack. This, however, can be achieved by remembering only the past two events. The graphs show that the robot with a fixed size history window learns the task slightly faster than the robot with a memory policy because the former has a smaller action space compared to the latter one. However, the main advantage occurs only in the early stages and learning of the final policy occurs at almost the same time.

4. Conclusion and Future Work

The results presented in this paper illustrate that enhancing a perceptual focus of attention mechanism with short term memory and by learning a memory policy enables, a robot to perform complex tasks (e.g. block copying) that require information about past events. The memory policy permits the robot to learn task-specific significant events that it has to remember and when to remember them in order to successfully complete the task.

Though this approach permits the robot to successfully learn a policy for the tasks in the blocks world domain but it still has a number of limitations to be addressed. In the experiments presented here, it is assumed that all blocks are uniquely identifiable by their feature combinations. Blocks that are not uniquely distinguishable are treated identically. This approach fails if the stack to be copied has two or more identical blocks. Another limitation with this approach is its small number of spatial representations (like *"Top"* and *"Bottom"*). This results in failure of the approach when the robot has to learn to copy more than one stack on the table.

To address these issues it is required to further investigate what additional types of spatial representations and memory aggregations are required, and whether these systems require counting mechanism.

5. Acknowledgements

This work was supported in part by UTA REP- 14748719

References:

[1] S. Rajendran and M. Huber, Developing Task Specific Sensing Strategies Using Reinforcement Learning. In *Proc. FLAIRS*, Miami Beach, FL, 2004, 738-743.
[2] J. H. Piater, Visual Feature Learning. Ph.D. diss., Dept. of Comp. Sc., Univ. of Mass., Amherst, 2001.
[3] Baddeley A. D., G. Hitch, Working Memory, *Recent advances in learning and motivation*, *8*, Edition G. Bower, NY : Academic Press, 1974.
[4] A.D. Baddeley, Working Memory (Oxford Univ. Press, 1986).
[5] D.H. Ballard, M.M. Hayhoe, Feng Li, S. D. Whitehead, Hand-Eye Coordination during Sequential Tasks, *Phil. Trans. R. Soc. London B,* 337, 1992, 331-339.
[6] T.Jaakkola, S.P. Singh, and M. Jordan, Reinforcement learning algorithm for partially observable Markov decision problems, *NIPS*, Cambridge MA, 1995, 345-352.
[7] L.J. Lin and T. Mitchell, Reinforcement learning with hidden states. *In Proc. of the 2nd Int. Conf. on SAB*, Cambridge, MA, 1992, 271-280.
[8] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra, Planning and acting in partially observable stochastic domains, *CS-95-11*, Brown Univ., Providence RI, 1995.
[9] M.L. Littman, A.R. Cassandra, and L.P. Kaelbling, Learning policies for partially observable environments: Scaling up. *In Proc. of the 12th Int. Conf. on Machine Learning,* San Francisco, CA, 1995, 362-370.
[10] R. A. McCallum, Learning to use selective attention and short-term memory in sequential tasks. *In Proc. of the 4th Int. Conf. on SAB*, Cambridge, MA, 1996, 315-324.
[11] A. R. McCallum, Hidden state and reinforcement learning with instance-based state identification. *IEEE Trans. SMC, 26B* (3), 1996, 464 -473.
[12] Huber M. and Grupen R.A., A Feedback Control Structure for On-line Learning Tasks, *Robotics and Autonomous Systems*, *22*(3-4), 1997, 303-315.
[13] P.J.G. Ramadge and W.M. Wonham, The control of discrete event systems. *Proc. IEEE*, 1989, *77*(1):81-97.
[14] J.G. Thistle, W.M. Wonham, Control of infinite behavior of finite automata, *SIAM Journal of Control and Optimization*, 32(4), 1994, 1075-1097.
[15] C.J.C.H Watkins, Learning From Delayed Rewards. *PhD thesis*, Cambridge Univ., Cambridge, England, 1989.
[16] L.P. Kaelbling, M.L. Littman, A.M. Moore, Reinforcement Learning: A Survey, *JAIR*, 1996, (4):237-285.
[17] R.S. Sutton and A.G. Barto, *Reinforcement learning: an introduction* (Cambridge, MA: MIT Press, 1998)