Efficient Motion-based Task Learning

Nicholas Malone[1], Aleksandra Faust[1], Brandon Rohrer[2], John Wood[3], Lydia Tapia[1]

***Abstract*— Generating motions for robot arms in real-world complex tasks requires a combination of approaches to cope with the task structure, environmental noise, and hardware imperfections. In this paper we present an efficient framework for adaptive motion task learning on real hardware that consists of task transfer, probabilistic roadmaps (PRM), and an online reinforcement learning algorithm. Online refers to the agent making decisions and then receiving information about that decision immediately after the decision has been made, instead of receiving a complete training set. The task transfer jump starts training on the hardware with knowledge learned in simulation. To achieve faster trainings speeds we integrate a PRM with the learning agent. For motion-based task learning, we use a reinforcement learning algorithm loosely based on human cognition. We demonstrate the framework by applying it to two pointing tasks on a 7 degree of freedom Barrett Whole Arm Manipulator (WAM) robot. The first task has a stationary target and illustrates the ability of the framework to quickly adapt and compensate for hardware noise. The second task goes a step further and introduces a non-stationary target, demonstrating the framework's ability to adapt quickly to a new environment and new task.**

I. Introduction

In order to perform tasks, robots must be able to adapt to a changing environment and problems. In order to process real world information, online planning has to process higher volumes of data with tighter deadlines at every time step. The planning is subject to hardware imperfections and errors in reading sensory information. Machine learning techniques, especially online reinforcement learning (ORL) is a useful tool for robotics motion learning and planning. It provides a closed-loop feedback system continuously incorporating current environment information into the planning and producing the motions required to perform a task. However, online reinforcement learning comes with several challenges that make it potentially problematic to use on real hardware.

[1]Nicholas Malone, Aleksandra Faust, Lydia Tapia are with the Department of Computer Science, University of New Mexico, Albuquerque, NM 87131, {nmalone, afaust, tapia}@cs.unm.edu

[2] Brandon Rohrer is with the Intelligent Systems, Robotics, and Cybernetics Group, Sandia National Laboratories, PO Box 5800, Albuquerque, NM, 87185-1010, brrohre@sandia.gov

[3]John Wood is withThe Manufacturing Engineering Program, University of New Mexico, Albuquerque, NM 87131, jw@unm.edu

BECCA development was supported by the Laboratory Directed Research and Development program at Sandia National Laboratories. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. Tapia is supported in part by the National Institutes of Health (NIH) Grant P20RR018754 to the Center for Evolutionary and Theoretical Immunology. Partial support was provided by SNL-3 PO #1110220, for Automated Systems Research, issued to the UNM Manufacturing Engineering Program.

Implementation of an ORL algorithm must be carefully designed to be safe for the robot both in terms of collision avoidance and producing motions that don't strain hardware. Training the ORL agent from scratch on real hardware can cause wear and tear to the hardware and thus change the dynamics of the system. Furthermore, motions take longer time to execute on hardware than in simulation, and the training phase could become impractically lengthy. Lastly, the sheer size of real world state spaces and physical laws of motion that need to be processed at every time step in real-time could make ORL prohibitively computationally expensive.

We propose a framework based on ORL that successfully overcomes the challenges above and learns motion-based tasks suitable for a real robot. To jump start the learning on hardware, and avoid a lengthy training phase, we transfer the knowledge from a task trained in simulation. To achieve performance suitable for a physical system and ensure the safety of the system, we rely on probabilistic roadmaps (PRM) for dimensionality reduction. The state space information reduced by the PRM is passed to our learning agent, which learns to produce efficient motion plans. We use a Brain Emulation and Cognition Architecture (BECCA) [8] agent. It is an adaptive online reinforcement learning algorithm paired with an unsupervised hierarchical feature creator. BECCA's algorithm contains a decay feature, allowing the agent to forget features and motion plans over time. This feature is especially useful for changing environments, as the agent continuously learns and updates plans based on the current feedback from the environment.

To demonstrate the framework, we implement it on a pointing task on a 7 DoF WAM using all 7 degrees of freedom. The robot needs to autonomously learn how to point at a target location in its environment regardless of the start position. In the first series of the experiments, the target location is stationary. In the second series of experiments the target location moves. We assess the performance of the framework by measuring how well the agent adapts to hardware imperfections and measurement noise. We also examine the performance of the framework by looking into time savings obtained by using transfer learning.

Our results show near-identical performance between simulation and transferred hardware runs. We show between 100 to 600 time steps of savings obtained by using transfer learning, and demonstrate an agile agent that quickly adapts to the new environment within 500 time steps. The work here extends our previous work in [4]. Previously we utilized transfer to accelerate our experimental procedures without

much discussion of the exact transfer process. Here we delve deeper into the ramifications and possibilities of transfer learning for robotics and reinforcement learning.

The rest of this paper is organized as follows: section II gives an overview of the related work. Section III discusses the hardware in more detail. Section IV discusses our methodology, and section V presents our experimental results. Finally, section VI concludes the paper with the framework's benefits to online, reactive motion-based learning.

II. Related Work

Taylor and Stone defined a taxonomy of transfer learning in the reinforcement learning domain in [10]. Using that terminology, our source task is a simulated pointing task. We have two target tasks. In one, the target task has the same goal and algorithm in both simulation and hardware runs. In the other the target is moved but it still has the same algorithm. The transferred knowledge is a set of feature groups and a Q-function.

There is active research on WAM training through human demonstration. A WAM system is represented as a canonical system of motor primitives [6]. The direct policy search class of reinforcement algorithms learns the parameters of the canonical system, while using the demonstration as an initial policy [6]. This line of research has produced a WAM capable of playing table-tennis [5], performing a ball-in-a-cup task [2], and flipping a pancake [3].

Unlike the above approaches which approximate the WAM model, the BECCA agent is agnostic to the type of the system and environment. At every time step the agent receives two signals: a sensory vector and a reward signal. The sensory vector is passed to a hierarchical feature creator. The resulting features and the reward signal are passed to the reinforcement learner.

PRMs are a method for solving complex path planning problems [1], and they tackle these complex problems by working in *conformation space* (C-space). PRMs work by building a roadmap. A roadmap is a graph where configurations are nodes and connections are edges. First, a set of configurations are sampled. Then, for each sampled node a set of candidates nodes are selected to form connections. PRMs have been extended to work in a wide variety of environments, ranging from simple open environments, to complex narrow passageways [1]. They have also been used in environments with moving obstacles [11].

III. Hardware Platform

The Barrett Whole Arm Manipulator (WAM) platform is a 7 degree of freedom (DoF) robotic arm. It is cable-driven and controlled with position encoders and torque estimation. The WAM has been connected to a GE Intelligent Platforms reflective memory network in a hub design that allows multiple computers to share memory at speeds ranging from 43 MB/s to 170 MB/s. The reflective memory networks allow remote computers to handle the planning and learning processing, while leaving a small and fast computer on-board the WAM to handle simple motion control.

IV. Methods

We propose a framework for online motion-based task learning that includes knowledge transfer from simulation to hardware. Subsection IV-A discusses transfer methodology, and subsection IV-B explains the agent in more detail. Subsection IV-C contains details on PRM implementation in our framework.

To test the performance of the framework, we implement it on a WAM and use two series of tasks: with a stationary target which is described in IV-D, and with a changing target described in IV-E.

A. Transfer Learning

Transfer learning typically refers to utilizing information learned in the past on a task in the present [10]. This past learning can be transferred to a new task or to the same task under different constraints. Transfer learning has also been utilized in transferring knowledge from one robot to another robot that may have a different internal architecture to represent the world [10]. Taylor and Stone [10] define jump start and time to threshold performance as two metrics for transfer learning. Jump start defines the amount of gain an agent initially recieves from transfered knowledge. Time to threshold performance defines the amount of time it takes an agent to reach the threshold performance, which is the best the agent can do at a given task.

In this paper, we consider a much narrower version of transfer learning. We transfer learned knowledge of a single task between a perfect simulation of a robot to imperfect robotic hardware. In simulation the robot always receives the exact same joint angles for a particular state, but in hardware the joint angles are subject to small error so re-entering the same state will not have the exact same state information. The source task uses the same learning agent, parameters, and reward function as the target task. The only difference is that the source task interacts with the WAM simulator while the target task interacts with the WAM hardware.

The WAM simulator is a simple kinetic simulator, representing the arm with seven points each corresponding to one degree of freedom. The arm moves in the simulator by simply adding the state and action vectors. The simulator does not inject noise, and performs perfect movements. The WAM arm, on the other hand, performs the movements as described in III. The resulting motion is subject to error in performing the movement.

When performing the transfer, we transfer the entire agent with all its internal states and accumulated experience. We only change the world model that it interacts with from the simulator or the WAM interface.

B. BECCA

BECCA is a general reinforcement learner [4]. It takes an input sensory vector as well as a scalar reward from the

world and then produces an output action vector. Internally BECCA combines an unsupervised feature creator with a reinforcement learning agent. Figure 1 shows an overview of the architecture.

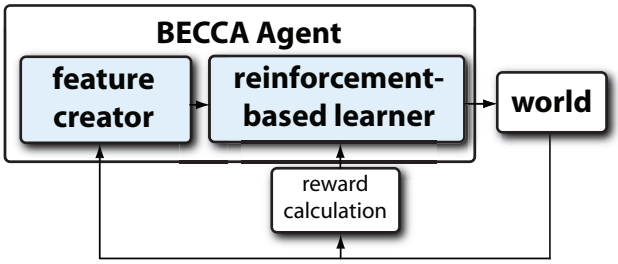

Fig. 1: BECCA's Architecture - At every time step, BECCA makes an observation in the world, extracts features from the sensory input, performs an action in response to the input, and receives a reward.

The feature creator identifies repeated patterns in the input vector and groups loosely correlated elements [7] [8]. The groups are considered to be subspaces, and the unit vectors of these subspaces are the extracted features. New inputs are projected onto each feature and the single feature, in each group with the greatest response magnitude is added to an active feature vector [8] [7] [9]. All other features with smaller response magnitudes are not added to the active feature vector. The active feature vector is then passed to the reinforcement learner, and on the next time step is fed back into the feature creator so that more complex features can be generated.

The reinforcement learner consists of a cause-effect table. The cause is the working memory, and the effect is the current active feature vector. Working memory is simply the sequence of actions that the agent has chosen in the last few time steps. The cause-effect pairs are then associated with an experienced reward. To use this model, the reinforcement learner compares its current working memory and the current active feature vector to the elements in the table. It then chooses the entry which is associated with the highest reward and takes the next action in the cause sequence.

In terms of traditional Markov Decision Process (MDP)-based reinforcement learning, the cause-effect pairs are equivalent to action-state pairs. The cause-effect table with the working memory and its expected reward roughly corresponds to a Q-function in traditional MDP-based reinforcement learning. However, BECCA's model does not assume the Markovian property and might depend on more than one previous state.

As time progresses, less frequently observed cause-effect transitions fade from the memory and the cause-effect table. This makes BECCA inherently able to adapt to new situations and environments at the cost of a steeper learning curve.

While BECCA is mostly automated, an engineer must design a task to interface with BECCA via sending sensory vectors and interpreting action vectors. Such an interface is called a task. A task simply defines what information from the world will be sent to the agent, and in what format. Note that BECCA is agnostic to the format. The task also defines how to read an action vector and move the robotic actuators. Again, note that BECCA is agnostic to how this is defined, and it will learn whatever format the engineer devises.

C. Probabilistic Roadmaps

In this paper, we use the PRMs combined with learning agent techniques from our previous work [4] to build a roadmap for the reinforcement learning agent to navigate. The learning agent is provided with the configurations and the adjacency information. It is constrained to making straight line movements along the edges in the adjacency matrix, thus constraining the reinforcement learner to learn how to navigate the roadmap. Each experiment generates a new random roadmap, except for when a transfer occurs. During a transfer, the previously learned roadmap is preserved. The PRM is the underlying state space provided to the learning agent.

D. Pointing Task with Stationary Target

The sensory vector is a n element binary vector, since the PRM contains n nodes. Each node represents a particular configuration of the robotic arm. When the robot is at a particular configuration the corresponding element in the sensory vector is set to 1.

Algorithm 1 shows how the pointing task is constructed. The action vector is a 4 element long binary vector and is parsed by the $interpret$ function. In this task, we have constrained BECCA to only return a single 1 in the action vector. The $interpret$ function in Algorithm 1 does the following: The 1 in the action vector represents BECCA selecting to move to one of the 3 neighbors, and the 4th element is interpreted as staying at the current configuration. For example the action vector $[0, 1, 0, 0]$ is interpreted by the task as selection to move to the second neighbor of the current configuration in the roadmap. The function then returns the configuration of the selected neighbor.

E. Pointing Task with Non-stationary Target

The formulation and the setup of the non-stationary target task is the same as in Section IV-D. The reinforcement

Algorithm 1 Task Step

Require: Task
1: $Task.agent.action$ = $[0, 0, 0, 0]$
2: **while** $not\ coverging$ **do**
3: $newLocation \leftarrow$ interpret($task.agent.action$)
4: $sendToWAM(newLocation)$
5: $task.currentPosition \leftarrow$ read current WAM location
6: $task.SensoryInput \leftarrow task.currentPosition$
7: $task.reward \leftarrow task.calculateReward()$
8: $task.agent \leftarrow agent_{step}(SensoryInput, Reward)$;
9: **end while**

learner is trained on an initial pointing task and then transferred to hardware, however upon being transferred the goal state is changed. Thus, the learning agent must compensate for the changed goal, while learning to adapt to the dynamics of the hardware system. Specifically, for this task the goal state is moved to one of the neighbors in the roadmap of the simulation goal state. The reward structure is changed so that the new goal state is reward 100 and the neighbors of the new goal 10 and the neighbors of the neighbors 0.1.

V. Experiments

We perform two experiments. One experiment is stationary target pointing task and the other is a non-stationary pointing task. All experimental results are averaged over five executions. Throughout the experiments, we measure the performance on the learning agent by measuring its cumulative reward. When the learning agent is transitioned from simulation to physical hardware, it is placed in a configuration that is as far as possible from the goal configuration.

We present the performance of the learning agent on hardware compared to performance in simulation. The agent executes in time steps but the graphs are shown in blocks, where 1 block equals 100 time steps. We look at the time savings brought on by using transfer learning, and the initial boost of performance that was obtained by knowledge transfer. In case of the non-stationary task, we will look at the time it takes the agent to react to a change in environment and recover to the previous level of performance

Each experimental run is executed on a new roadmap of 50 configurations generated using PRMs. Each configuration is connected to 3 neighbors and itself. A random point in the 50 configurations is chosen as the goal. The goal node is given a reward of 100. The neighbors of the goal are given a reward of 10 and the neighbors of the neighbors are given a reward of 0.1. All other configurations are given a reward of 0.

A. Pointing Task with Stationary Target

Figure 2 shows the cumulative reward of the pointing task with the stationary target in simulation and on hardware. The vertical line indicates the transition from the simulation to the hardware. The results show near-seamless transition, and the average performance of the agent on hardware very close to the performance in the simulation.

Table I shows the average cumulative reward for each experiment after stabilization, before and after transition to real hardware. Stabilization in simulation occurs at 20 blocks. The performance of the agent on the hardware outperforms the agent in simulation by 154 units of reward.

To better demonstrate the advantages of using the transfer learning in our framework, the pointing task with stationary target experiments were run again in a different manner. Five completely untrained learning agents were run on hardware for 20 blocks and the results averaged together. Then five agents which were trained for 100 blocks in a simulation were run on hardware for 20 more blocks and averaged together. Figure 3 shows the comparison of the stationary pointing task using transfer to the same task without using transfer. The advantages of using transfer are seen primarily in the jump start and the time to threshold metrics. Table II shows the transfer metrics for the three experiments. Jump start shows the immediate gain from using the transfer. The pointing task starts very close to the threshold performance using the transfer and has a jump start gain of 5716. In all random runs the transfered learning agent outperforms the non-transfered learning agent (Table II). Furthermore the transferred task reaches the threshold performance in 2 blocks compared to 7 blocks without transfer (Figure 3). It is important to note the time saved by using transfer learning. Table III shows the run times for simulation versus hardware for 20 blocks. It is clear that simulation is faster by up to 1 hour and 55 minutes. Using transfer learning it takes significantly less physical time on the robotic hardware for the agent to perform the given task as near optimal levels. This not only saves valuable time but it also saves valuable wear and tear on the hardware.

It is important to note that the learning algorithm is not executing pre-planned paths. It learns from experience which paths lead to highest reward and attempts to follow those paths. The paths learned in simulation provide BECCA with a strong foundation to work from, however each execution of the learning problem finds different paths due to the randomness of exploration. Thus, it is possible to witness executions of BECCA on the same underlying roadmap with slightly varying performances.

TABLE I: Average cumulative rewards in simulation and on hardware after the stabilization for 7DoF task with a stationary target and 7DoF task with a non-stationary target

Task	Simulation	Hardware
Stationary Target	7460.3	7614.8
Nonstationary Target	7460.3	7491.5

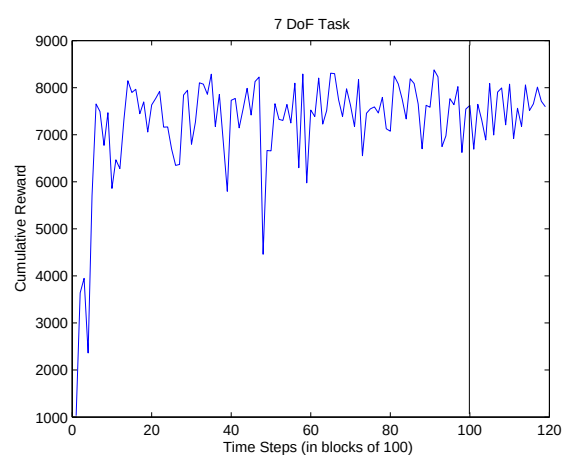

Fig. 2: Cumulative reward for the pointing task with stationary target per time step. The vertical line indicates where the learning agent was transitioned from simulation to real hardware.

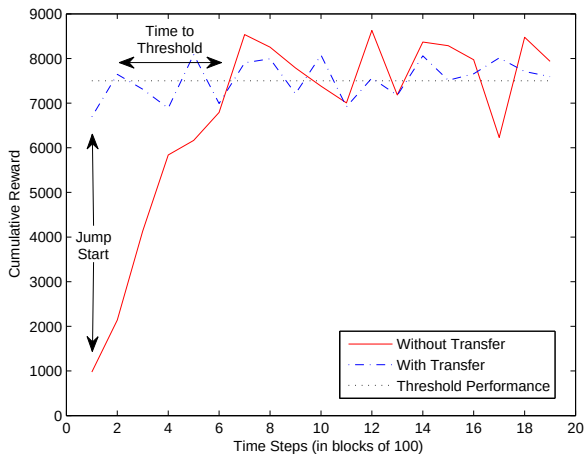

Fig. 3: Cumulative reward for the pointing task running on hardware with stationary target task with transfer and without transfer per time step. Transfer is when an agent trained in simulation is transferred to hardware. Jump start shows the initial gain obtained by using the transferred knowledge. Time to threshold indicates the time that the task without the transfer needs to achieve the same level of performance as the task with the transfer.

TABLE II: Transfer Metrics for stationary and non stationary tasks. Jump start shows the gain from using transfer. Threshold gain shows the reduction in time steps needed to reach the threshold performance

Task	Metric	Average	min	max
Stationary	Jump Start (reward)	5716	2757	9280
	Threshold Gain (steps)	500	200	700
Non-stationary	Jump Start (reward)	1313	364	1702
	Threshold Gain (steps)	100	100	400

B. Pointing Task with Non-stationary Target

In this experiment the reinforcement learner is trained on an initial pointing task and then transferred to hardware. However, upon being transferred, the goal state is changed. Thus, the learning agent must compensate for the changed environment. The goal state is moved to one of the neighbors in the roadmap of the simulation goal state. The reward structure is changed so that the new goal state is reward 100 and the neighbors of the new goal 10 and the neighbors of the neighbors 0.1.

Figure 4 shows the results of 100 blocks of simulation and then 20 blocks of running on hardware where the goal has changed. Initially there is a steep performance drop, but the reward does not drop to zero. The agent quickly recovers and learns the new reward structure within 6 blocks. This shows the online nature of the BECCA algorithm. It is able to first learn one environment and then be placed into a slightly different environment but able to compensate for the change quickly.

TABLE III: Average time in minutes to run 20 blocks in simulation and on hardware for 7DoF task with a stationary target and 7DoF task with a non-stationary target

Task	Simulation (min)	Hardware (min)
Stationary Target	23	122
Non-stationary Target	24	121

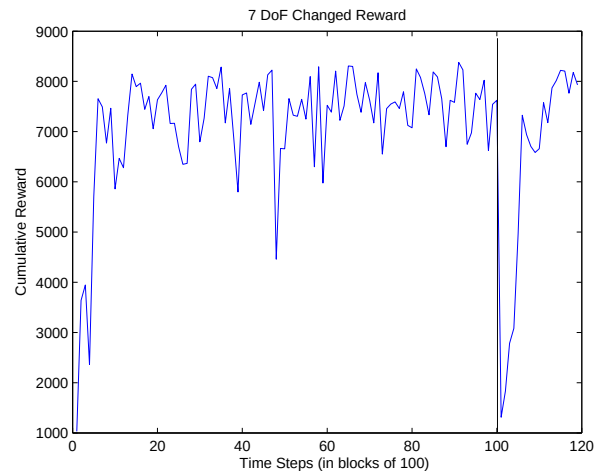

Fig. 4: Cumulative reward for running in simulation and then transferring the task to hardware. The transfer occurs at 100 blocks.

Figure 5 is a comparison between the agent having previously learned a pointing task to an agent without any prior knowledge. However, the agent with knowledge has learned to point to a different goal in simulation before being run on hardware. The untrained agent is also run on hardware but has a stationary target. Thus, the transfered agent has some information about the structure of the environment but it does not have the exact reward structure as the goal was moved before being placed on real hardware. The figure shows that the agent with prior knowledge has a small jump start of 1313 units of reward and reaches the threshold performance 1 block faster than the agent without transferred knowledge.

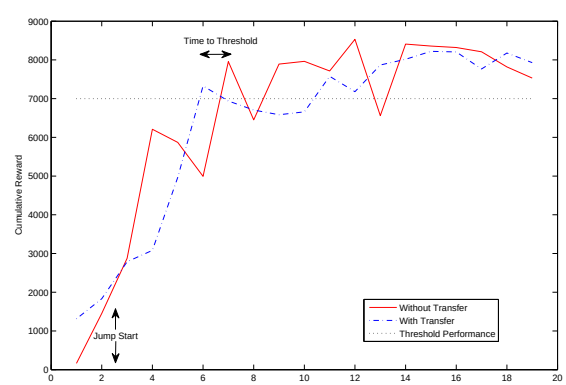

Fig. 5: Cumulative reward for the pointing task running on hardware with a non-stationary target task with transfer and without transfer per time step. Jump start shows the initial gain obtained by using the transferred knowledge. Time to threshold indicates the time that the task without the transfer needs to achieve the same level of performance as the task with the transfer.

VI. Discussion

We demonstrated an efficient online motion-based task learning framework based on reinforcement learning that works in high-dimensional spaces in real-time, is reactive to changes in the environment, performs safe hardware motions, and efficiently learns on hardware. We demonstrated the framework by implementing it on a 7 DoF WAM using all joints to produce pointing motions with both stationary and non-stationary targets. The framework is robust and extensible to other robotics systems as well as with different model formulations, and for a large variety of tasks as well.

Dimensionality reduction and collision checks can be handled through PRMs for any motion-based task. When PRMs are used in this manner, they impose hard limits on the system. For example, self-collision states tend to be invariant to the type of environment or the task, and are good candidates to be precomputed ahead of time. When there is error in the model used for simulation caused by noisy sensor data, the robot can explore the validity of the simulation's roadmap and learn how to efficiently navigate in the physical environment.

Transfer learning can be used to avoid early learning phases when the agent's performance tends to be erratic, to reduce wear and tear on robot, and to speed up the learning process on the real hardware. It can be a powerful techniques to mitigate the long convergence times of reinforcement learning. Combining transfer learning, reinforcement learning and probabilistic roadmap methods produces a powerful framework for solving complex robotic tasks. By harnessing each method's strengths, the weaknesses of the other methods can be mitigated.

An online reinforcement learning algorithm is a suitable candidate for a planner when paired with the above techniques. Such a reinforcement learner continuously learns and updates its policy by incorporating most recent experience from the environment and produces motion plans that are adaptive, real-time, and reactive. Hardware soft limits can be implemented through the reward function.

VII. Acknowledgmetns

We would like to thank Dr. Dave Vick for his help with the robotic hardware setup.

References

[1] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.

[2] J. Kober and J. Peters. Policy search for motor primitives in robotics. *Advances in neural information processing systems*, 21:849–856, 2009.

[3] P. Kormushev, S. Calinon, and D. G. Caldwell. Robot motor skill coordination with em-based reinforcement learning. In *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, pages 3232–3237, Taipei, Taiwan, October 2010.

[4] N. Malone, B. Rohrer, L. Tapia, R. Lumia, and J. Wood. Implementation of an embodied general reinforcement learner on a serial link manipulator. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 862 –869, may 2012.

[5] K. Mülling, J. Kober, and J. Peters. A biomimetic approach to robot table tennis. *Adaptive Behavior*, 19(5):359–376, 2011.

[6] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4):682 – 697, 2008.

[7] B. Rohrer. Biologically inspired feature creation for multi-sensory perception. In *BICA*, 2011.

[8] B. Rohrer. A developmental agent for learning features, environment models, and general robotics tasks. In *ICDL*, 2011.

[9] B. Rohrer. Becca: Reintegrating AI for natural world interaction. In *Submitted to AAAI Spring Symposium*, 2012.

[10] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.*, 10:1633–1685, Dec. 2009.

[11] Y. Wu. An obstacle-based probabilistic roadmap method for path planning. Master's thesis, Department of Computer Science, Texas A&M University, 1996.