Implementation of an Embodied General Reinforcement Learner on a Serial Link Manipulator

Nicholas Malone[1], Brandon Rohrer[2], Lydia Tapia[3], Ron Lumia[4] and John Wood[5]

***Abstract*— BECCA (a Brain-Emulating Cognition and Control Architecture software package) was developed in order to perform general reinforcement learning, that is, to enable unmodeled embodied systems operating in unstructured environments to perform unfamiliar tasks. It accomplishes this through automatic paired feature creation and reinforcement learning algorithms. This paper describes an implementation of BECCA on a seven Degree of Freedom (DoF) Barrett Whole Arm Manipulator (WAM) undergoing a series of experiments designed to test the reinforcement learner's ability to adapt to the WAM hardware. In the experiments, the following is demonstrated, 1) learning to transition the WAM between states, 2) learning to perform at near optimal levels on one, two and three dimensional navigation tasks, 3) applying learning in simulation to hardware performance, 4) learning under inconsistent, human-generated reward, and 5) combining the reinforcement learner with Probabilistic Roadmap Methods (PRM) to improve scalability. The goal of the paper is to demonstrate both the scalability of the BECCA reinforcement learning approach using different formulations of the state space and to show the approach in this paper operating on complex physical hardware.**

I. Introduction

Robotic path planning and control is a challenging task and often requires intimate knowledge of a specific platform to build a path planner for it. Reinforcement learning is an alternative approach to hand designing a path planner. There are many different reinforcement learning techniques but they all have the machine learn how to find a path which maximizes a metric called the reward by exploring state-action sequences [1], [11], [5], [4], [3], [2], [12]. Abtahi et al. in [1] describe a reinforcement learning technique which combines deep belief networks with a function-based reinforcement learner. [11] combines a traditional reinforcement learning algorithm with additional input from a human trainer. The work in [5] deals with combining the approximation value function approach with the discretization approach to reinforcement learning. Cuccu's work in [4] uses a type of reinforcement learning for artificial neural networks operating on the mountain-car benchmark. Clouse's work on reinforcement learning in [3] is about combining a traditional reinforcement learner with apprentice learning or mimic learning. Finally, the work of Legenstein et al. discuses a technique for handling high dimensional inputs [12].

The Barrett Whole Arm Manipulator (WAM), as seen in Fig 1, is a 7 DoF robotic system. Instead of handcrafting a control and path planning algorithm for the WAM, machine learning techniques can be used to learn how to control and path plan the WAM. BECCA is a general reinforcement learner designed to learn how to control arbitrary robotic platforms. In this paper BECCA is implemented on subsets of the WAM platforms DoFs. The goal of this work is to first provide a proof of concept on a physical platform and then to investigate the scalability of different approaches.The reinforcement learner is combined with the Probabilistic Roadmap Methods (PRMs) in order to improve scalability.

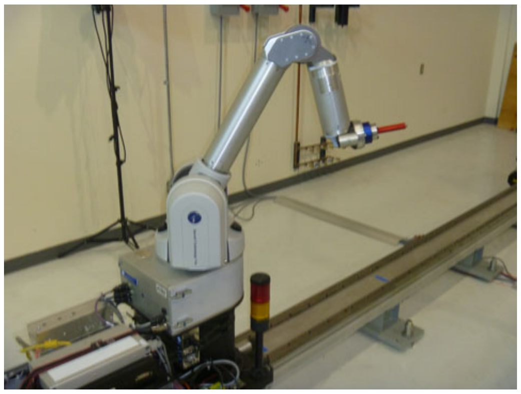
Fig. 1. Whole Arm Manipulator (WAM).

II. Related Work

Creating a general learning machine has been one of the grand goals of artificial intelligence (AI) since the field was born. Efforts to achieve this goal may be divided into two categories. The first category uses a depth first approach, solving problems that are complex, yet limited in scope, such as playing chess. The assumption underlying these efforts is that an effective solution to one problem may eventually be generalized to solve a broad set of problems. The second category emphasizes breadth over depth, solving large classes of simple problems. The assumption underlying these efforts is that a general solution to simple problems may be scaled

This work was supported by Sandia National Laboratories PO# 1074659

[1]N. Malone is a student at the University of New Mexico namlone@cs.unm.edu

[2]B. Rohrer is with Sandia National Laboratories brrohre@sandia.gov

[3]L. Tapia is with the Faculty of Computer Science, University of New Mexico ltapia@cs.unm.edu

[4]R. Lumia is with the Faculty of Mechanical Engineering, University of New Mexico lumia@unm.edu

[5]J. Wood is with the Faculty of Mechanical Engineering, University of New Mexico jw@unm.edu

up to address more complex ones. An example of the first category would be a master level chess playing agent, while an example of the second category would be an agent with the capabilities of an ant worker. The work described here falls into the second category, focusing on breadth. The motivating goal for this work is to find a solution to natural world interaction, the problem of navigating, manipulating, and interacting with arbitrary physical environments to achieve arbitrary goals. In this context, environment refers both to the physical embodiment of the agent and to its surroundings, which may include humans and other embodied agents. The agent design presented here is loosely based on the structure and function of the human brain and is referred to optimistically as a Brain-Emulating Cognition and Control Architecture (BECCA) [16], [17].

A Brain-Emulating Cognition and Control Architecture agent interacts with the world by taking in actions, making observations, and receiving reward (see Fig. 2). Formulated in this way, natural world interaction is a general reinforcement learning problem, [19] and BECCA is a potential solution. Specifically, at each discrete time step, it performs three functions:

1) reads in an observation, a vector $o \in \Re^m \mid 0 \le o_i \le 1$.
2) receives a reward, a scalar $r \in \Re \mid -\infty \le r \le \infty$.
3) outputs an action, a vector $a \in \Re^n \mid 0 \le a \le 1$.

Because BECCA is intended for use in a wide variety of environments and tasks, it can make very few assumptions about the environments beforehand. Although it is a model-based learner, it must learn an appropriate model through experience. There are two key algorithms to do this: an unsupervised feature creation algorithm and a tabular model construction algorithm.

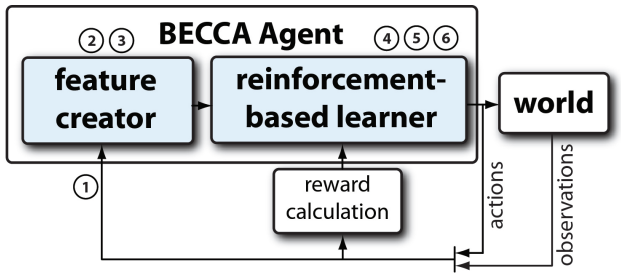

Fig. 2. At each timestep, the BECCA agent completes one iteration of the sensing-learning-planning-acting loop, consisting of six major steps: 1) Reading in observations and reward, 2) Updating feature set, 3) Expressing observations in terms of features, 4) Predicting likely outcomes based on an internal model, 5) Selecting an action based on the expected reward of action options, and 6) Updating the model.

The feature creator component identifies repeated patterns in the input vector [15]. It then groups loosely correlated elements of the input vector. The groups are treated as subspaces and unit vectors of these subspaces are features [15]. New inputs are also projected onto existing features and the single feature in each group which has the greatest response is turned on while all others in that group are turned off [14] [15] [18].

The reinforcement learning component receives feature activity, reward, and direct input from the environment. Each feature is associated with an approximate reward. It keeps track of recent actions and recent features in working memory which is then used to update the model. The actual model is a table of cause-effect pairs. The cause is the working memory and the effect is the current feature. Considering this in standard reinforcement learning language, the model can be thought of as a sequence of state-action pairs. Entries in the table which are rarely observed are deleted from the model [14] [15] [18].

To chose an action the reinforcement learner compares the current working memory to the entries in the model and selects the entry which both matches the current working memory and which has the highest recorded reward. With a set probability, an exploratory action is chosen instead [14] [15] [18].

III. Methods

The methods section first describes the WAM hardware, then the interface with the WAM, and finally, how a general task is built for BECCA to utilize the WAM interface. The reinforcement learner is designed for general purpose learning on any platform but there is still a small amount of configuration needed to allow it to interface with a specific hardware. A task is a framework for incorporating the specific requirements for any hardware. A task is a translator for the action vector produced by BECCA to the hardware, and a translator for the sensory information into an input vector for the reinforcement learning approach. The task also is responsible for calculating the reward for any given state.

A. Robotic Hardware

The Barrett Whole Arm Manipulator (WAM) platform used in the experiments is a seven degree of freedom (DoF) robotic arm as seen in Fig. 1. It is a cable driven system controlled with joint position encoders and torque sensors. For the experiments in this paper, the WAM has been connected to a GE Intelligent Platforms reflective memory network in a spoke design that allows multiple computers to share memory at speeds ranging from 43 MB/s to 170 MB/s. The reflective memory network allows remote computers to handle the planning, learning processing, and sensor processing, while leaving a small and fast computer on-board the WAM to handle simple motion control.

B. WAM Interface

The WAM is connected to a xPC Target Kernel running Matlab Simulink 7.7.0 R2008b [13]. The controller for the WAM is written in Simulink and interfaces with remote computers via the reflective memory network. The Simulink code responsible for directly issuing commands to the WAM, henceforth the WAM controller, receives a command vector by reading a specific block of reflective memory. The command vector is a length seven vector containing the desired joint angles in radians of each for the seven WAM joints.

The WAM controller, upon receiving a command vector, places the command vector into a buffer, which only stores one move. The command vector is first sanitized so that each entry is within the WAM's joint limits. If the WAM is not executing a move, it compares its current location to the command vector buffer. If the command vector buffer is sufficiently different from the current location, the WAM controller computes a linear interpolation in joint space between the two joint angles and executes the path within the allowable WAM workspace. However, the velocity follows a fifth-order smooth polynomial as seen in Fig 3, and is used both for safety and for mimicking biological motion [6]. Slow beginnings and endings to moves provide safe joint torques. In the current architecture a move cannot be interrupted.

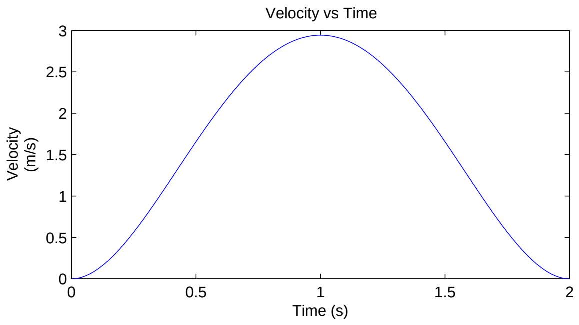

Fig. 3. Example Velocity Profile for a Single Joint.

C. Tasks

BECCA is designed to be a reinforcement learner for general robotic applications. Thus, there is a simple interface between the external world and its internal representation of the world. At the simplest level, it takes a vector of inputs and transmits a vector for outputs. The structure of the inputs and outputs is intended to be irrelevant to BECCA and so it must learn the structure of both. Therefore, a designer must define an input vector and an action (output) vector as well as a reward structure for each task. The task must also define how to interpret an action vector, and how to translate a sensory information into an input vector. The manner in which the task interprets outputs and delivers inputs to BECCA is completely at the discretion of the task designer. Algorithm 1 is a pseudocode example of an interpretation of an action vector, for a 1 DoF task.

For the 1 DoF task the joint space is partitioned into equally spaced bins and BECCA is constrained to move between bins. Algorithm 1 calculates the direction and number of bins the action vector specifies to move. Line 1 first calculates the offset move for a given action vector. For this task the action vector is treated much like a stepper motor move. The vector has 8 components that are either 0 or 1. Vector component 1 corresponds to a move of one bin to the right while component 2 corresponds to a move of one bin to the left. Components 3 and 4 represent a move of two to the right or two to the left and so on for the rest of the components. A total move is calculated by summing together all of the components to determine how many bins to move and in which direction. Lines 2 and 3 calculate

Algorithm 1 interpretActionVector(action)

```
1: $move \leftarrow 1 * action[1] - 1 * action[2]$
         $+2 * action[3] - 2 * action[4]$
         $+3 * action[5] - 3 * action[6]$
         $+4 * action[7] - 4 * action[8]$
2: $currentPos \leftarrow getJointPositionFromWam()$
3: $actualMove \leftarrow (move * 0.3142 + currentPos[4])$
4: if $actualMove < jointMinLimit$ then
5:     $actualMove \leftarrow jointMinLimit$
6: end if
7: if $actualMove > jointMaxLimit$ then
8:     $actualMove \leftarrow jointMaxLimit$
9: end if
10: return $actualMove$
```

the current position of the arm and then calculate the joint angle that the arm should be at given the desired bin move. Lines 4 through 9 then sanitize the actual move to be within the joint limit constraints. It is important to note that at initialization, BECCA does not know the transition function between states, where a state corresponds to a particular input vector, and an input vector is a vector of all zeros except for the bin the WAM is currently in is a 1. It must learn what each action vector does in a given state.

A task is a specific instance of a problem for BECCA to learn. Algorithm 2 outlines the basic steps in a simple task. In line 1 the action vector is retrieved from BECCA and then interpreted by the task in line 2 by calling Algorithm 1. In line 4 the move is sent to the WAM via the WAM interface discussed in section III.B. An input vector is then generated given the new state and sent to the reinforcement learner in lines 5 through 10. Finally, a reward is calculated based on the new state by lines 11 through 17. In the Algorithm 2 example, there is a single reward bin, which is given a reward of +10, the edge bins are punished by a reward of -10 and any other bin is punished by a reward of -1.

IV. Experiments

The experiments section starts with a simple binning formulation for a 1-DoF pointing task. It then progresses to a 2-DoF pointing task to demonstrate the scalability issues reinforcement learners have with specific problem formulations. The manual training section then demonstrates that BECCA can be used in a real time human-trained environment for simple tasks. This is important since in order for learning robots to be useful they will need to be trained by human operators in specific tasks. Finally, the PRM section proposes a solution to the scaling issues presented by the simple binning formulation.

A. 1-DoF Task

The first experiment is a 1-DoF task. On the WAM joint 4, the elbow joint, is used for the single DoF. Joint 4 has a range of motion from 0 radians to π radians. For simplicity, the joint space is divided into 10 equally spaced bins, such that any angle between 0 and 0.3142 radians is bin 1, any

Algorithm 2 Task Loop

```
1:  loop
2:     $action \leftarrow BECCA.getAction$
3:     $move \leftarrow interpretActionVector(action)$
4:     $sendMoveToWAM(move)$
5:     $waitForMoveToFinish()$
6:     $currentPos \leftarrow getJointPositionFromWam()$
7:     $bin \leftarrow findBin(currentPos, numBins)$
8:     $binVector = [0, 0, 0, 0, 0, 0, 0, \ldots, 0]$
9:     $binVector[bin] \leftarrow 1$
10:    $BECCA.inputVector = binVector$
11:    if $bin == rewardBin$ then
12:       $BECCA.reward \leftarrow 10$
13:    else if $isEdgeBin(bin)$ then
14:       $BECCA.reward \leftarrow -10$
15:    else
16:       $BECCA.reward \leftarrow -1$
17:    end if
18: end loop
```

angle greater than 0.3142 and less than 0.6284 radians is in bin 2 and so on until π radians. The binned joint space becomes the state space and is used as the input vector to BECCA. The input vector to BECCA is then of length 10, and has zeros in every position except for the current bin. For example if the WAM's joint 4 is at angle 0.1125 radians then it is in bin 1, and the input vector is: [1,0,0,0,0,0,0,0,0,0].

The action vector sent by BECCA for the 1-DoF task is a length 8 vector and each entry in the vector is constrained to be either a 0 or a 1. For the 1-DoF task, the action vector is interpreted by Algorithm 1. The action vector for the 1-DoF task was chosen to be a simple design without significant regard for how well the reinforcement learner would handle it. Algorithm 1, shows how the action vector for the 1-DoF task is interpreted. Once the action vector and the input vector operations have been specified they can be used in the 1-DoF task. Lastly, a reward function must be specified. In this experiment, the basic algorithm (Algorithm 2) is altered so that bin 5, the center bin, is given a reward of +10, and every other bin is given a reward of -1 except the edge bins 1 and 10 which are given a -10 reward.

BECCA is first trained in simulation and then ported to hardware partially through the experiment. Each data point is the cumulative reward the reinforcement learner receives after 100 iterations (100 iterations is a block).

Fig. 4 is an average of 10 runs on the 1-DoF task. BECCA was first trained in simulation and then ported to hardwar at block 25. Averaging the runs slightly smoothes out the learning curve for BECCA and better illustrates the climb to optimal performance at 700 units of reward. Fig. 4 is a proof of concept for interfacing with a specific task. It appears that BECCA does poorly, reaching only approximately 700 units of rewards, however there is a 30 percent exploration rate at minimum, thus on the simple 1-DoF task 700 units of reward is approximately optimal due to the -10 edge punishment and the fact that some exploratory actions have no effect. Here optimal is receiving maximal reward.

It should be noted that even on the relatively small state space of 10 bins and an action vector of length 8, BECCA still takes on average 3,000 iterations to converge on optimal behavior. 3,000 iterations with an average move time of 2 seconds on the WAM is approximately 1.7 hours of operation time on the WAM, which presents a problem for larger state/action spaces. The design of the WAM controller interface allows for BECCA to be run in simulation mode until convergence and then be connected to the WAM. A simulation of the WAM is not a perfect matching to the actual hardware due to idiosyncrasy of the hardware differing minutely from the theoretical. However, BECCA is a general learning program and is able to compensate for the difference between simulation and real hardware after exposure to the hardware. Thus, the experiments can be safely run in simulation, which operates significantly faster than hardware and reduces the time it takes BECCA to learn a task. An iteration in simulation takes a fraction of second, while in hardware a single iteration takes approximately 2 seconds. In Fig. 4 the structures, which have been learned, are ported to the WAM hardware at block 25.

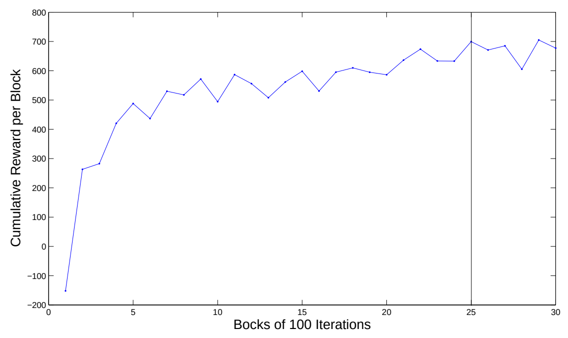

Fig. 4. Average of 10 1 DoF Task Learning Curves. The black vertical bar shows the transition from simulation to the WAM hardware.

B. Manual Training

BECCA is also capable of being trained manually. In the current version, version 0.3.9, manual training can be tedious given the number of iterations required for convergence. To compensate for the convergence time, a very simple 3 bin task with an action vector of length 4 is used during the manual training experiment. A trainer can reward or punish BECCA at any time, however only the last reward or punishment is registered per move, where a move is a single loop through Algorithm 2. Thus, this task differs from the normal training task in that the reward function is at the discretion of the trainer. The trainer can choose to change how BECCA is rewarded at will. Standard training tasks have a reward function which is coded directly into the task. The 1-DoF manual training task is running on the WAM robot without simulation bootstrap training. The trainer only rewards BECCA when joint 4 is in the middle bin, which places the elbow joint at $\pi/2$.

Fig. 5 shows the results of a typical manual training task. In Fig. 5 BECCA is only rewarded for being in state 2. The trainer sometimes intentionally fails to give reward for being in state 2, to demonstrate that BECCA can handle inconsistent rewards which it will likely receive in practice from a human trainer. The trainer never gives punishment during the experiment. Fig. 5 is slightly different from the previous cumulative reward figure in that the block size is only 5 iterations, and the reward is at most +10 per iteration making the maximum 50 units of reward. The cumulative reward seen in Fig. 5 shows that BECCA has converged at around 50 iterations (block 10 on the independent axis). The rapidity of convergence is due to the simplicity of the task, but it demonstrates that BECCA can learn a task with inconstant reward and can be trained by hand.

Fig. 6 shows the percentage of time spent in each bin. The blue vertical striped bar shows the overall percentage of time BECCA has spent in a bin, while the red horizontal striped bar shows the percentage of time spent in a bin for the last 5 iterations. On average 70 percent of its time is spent in bin 2, which is the only bin it receives reward in. 70 percent is optimal due to a 30 percent exploration rate. The red bar being at 100 percent in bin 2 indicates that BECCA chose to spend all of its time in bin 2 over the last 5 iterations. The results of both Fig. 5 and Fig. 6 show that BECCA is correctly learning to select bin 2 over the other two bins during the manual training task.

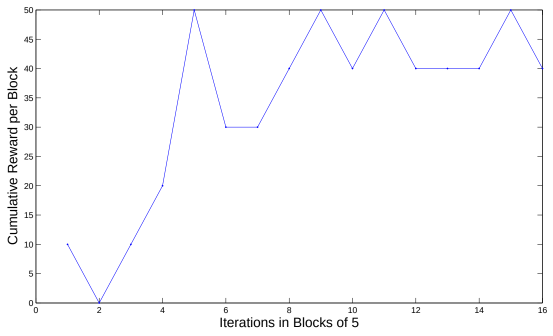

Fig. 5. Manual Training 1 DoF Task Learning Curve.

C. 2-DoF Task

The previous two experiments have been on a single degree of freedom and have had relatively small and simple state spaces and action vectors. The 2-DoF task has an exponentially larger state space than the 1-DoF task. The 2 degree of freedom task uses joint 2 and joint 4. Joint 2 has a range of motion from -1.99 radians to 1.99 radians. Both joint 2 and joint 4 are divided into 10 bins each, meaning that the state space is now a 10 by 10 matrix of states. The action vector must also be enlarged to 16. The first 8 components control joint 2 and the second 8 components control joint 4 in the same way as the first 1-DoF task. Thus the state spaces from the 1-DoF and 2-DoF task can be compared. The state space for the 1-DoF task had 10 states with an action vector of length 8, resulting in a state/action space of $10 * 2^8$ because BECCA can send an arbitrary bit string of length 8. The state space for the 2-DoF task has 100 states and an action vector of length 16 giving a state/action space of $100 * 2^{16}$. The reward function has been modified to $Reward = rJ2(Joint2Bin) + rJ4(Joint4Bin)$ where rJ2 and rJ4 are defined as:

$$rJ2(x) = \begin{cases} 10 & \text{if } x = 5 \\ -10 & \text{if } x = 0 \vee x = 10 \\ -1 & \text{otherwise} \end{cases}$$

$$rJ4(x) = \begin{cases} 10 & \text{if } x = 5 \\ -10 & \text{if } x = 0 \vee x = 10 \\ -1 & \text{otherwise} \end{cases}$$

In rJ2 and rJ4 bin 5 corresponds to the middle bin, and bins 0 and 10 correspond to the first and last bin, so the robot will be in a right angle when it is in the correct location.

Fig. 7 shows the cumulative reward for the 2-DoF task. After approximately 5000 iterations the reward stabilizes to an oscillation between 500 and 1500 units of reward. Based on the reward structure the maximum reward is 2000 per block, which indicates that the 2-DoF task is performing under optimum. The reason for the underperformance can be better seen by looking at Fig. 8 and Fig. 9, which show the percentage of time each joint spends in a particular bin. Fig. 8 indicates that BECCA spends the majority of its time in bin 5 for joint 2, but Fig. 9 indicates that not enough time is spent in bin 5 for joint 4. The underperformance stems from not fully learning the large state/action space. If BECCA correctly finds joint 2 it still must find joint 4 simultaneously in order to locate the optimal state. An action of length 16 has 2^{16} possible bit vectors, and there are 100 states, thus BECCA would have to visit $100 * 2^{16}$ state action combinations to fully explore the environment.

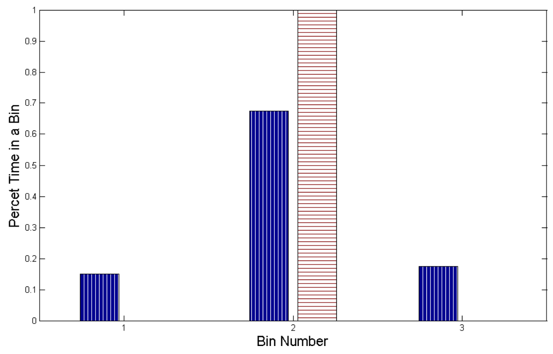

Fig. 6. Percentage of Time BECCA Spends in a Bin. This graph corresponds to block 15 in Fig 5. The red horizontal striped bar shows that BECCA chose to spend every step in bin 2, and thus received maximum reward for that block.

D. Probabilistic Roadmap Methods and BECCA

PRMs are a path planning technique used with robots with high DoFs to reduce the complexity searching in a high-dimensional and continuous space of possible conformations.

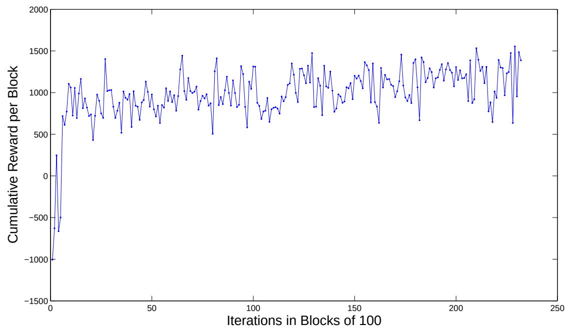

Fig. 7. 2-DoF Task Cumulative Reward.

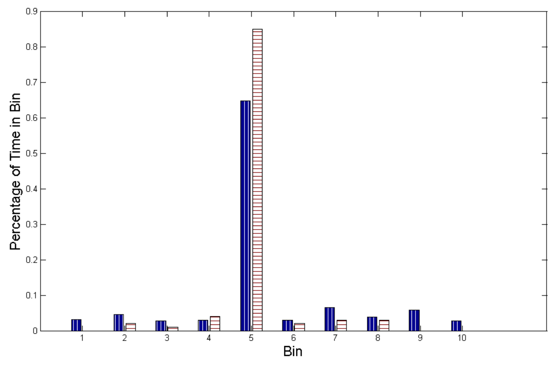

Fig. 8. 2-DoF Task Percentage of Time Spent in Each Bin for Joint 2. The blue vertical striped bar shows the cumulative percentage of time spent in a bin, while the red horizontal striped bar shows percentage of time spent in a bin during the last block.

They have been applied to a variety of complex robot types including manipulators [9], walking robots [8], and nonholonomic robots [7]. PRMs tackle the planning problem by working in *conformation space* (C-space) rather than the workspace. In regards to joints, this means that each DoF of a joint is mapped to a single dimension in C-space. Then, the path planning problem is reduced to finding a sequence of feasible states in this conformation space, those in C-free.

PRMs work by building a roadmap of possible feasible motions in C-free [10]. They do this by randomly selecting points in C-free. Then, nearby points are connected by simple connection methods. Nearby can be defined by low-cost Euclidean distance calculation to identify the k nearest neighbors. Connection can be achieved using straight-line interpolation.

The goal of incorporating PRMs was to help guide the searching. For example, BECCA was successful at automatically searching the large search space with 2 joints, however, it struggled. This would be magnified with a 3 joint problem moving from 100 states to 1,000 states, which would increase the complexity at least by a factor of 10. Incorporating PRMs allowed us to reduce the state space for 3-DoF tasks to the number of nodes in the roadmap. For the results shown, the state space has 50 nodes, but the number of nodes can be adjusted.

For the 3-DoF task, joints 1, 2, and 3, are mapped into a 3

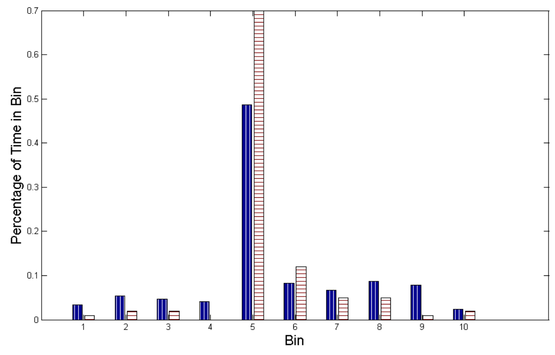

Fig. 9. 2-DoF Task Percentage of Time Spent in Each Bin for Joint 4. The blue vertical striped bar shows the cumulative percentage of time spent in a bin, while the red horizontal striped bar shows percentage of time spent in a bin during the last block.

dimensional C-space. Then, fifty random points are sampled in the C-space using a uniform distribution. The fifty points are then connected probabilistically based on the distance between the points, such that closer points have a higher probability of being connected. Fig. 10 shows an example of a PRM generated for the 3-DoF task.

Fig. 11 shows the cumulative reward per block for BECCA operating on the 3-DoF PRM task. The maximum reward that can be receive per iteration is 100, making the maximum per block 10,000 units of reward. The reward structure for the PRM task assigns a reward of 100 to the target node, a reward of 10 to all neighbors of the target node, and a reward of 1 to the neighbors of the neighbors. Every other node is given a reward of 0. The goal state is chosen at random from the 50 points, without loss of generality. BECCA's action vector is interpreted as which neighbor to transition to. Each node in the PRM is numbered and the input vector is 50 long, with each entry in the vector corresponding to a particular numbered node. All values in the vector are set to 0 except for the current node's number is set to 1. The PRM covers a wide area in the WAM's range of motion, but only takes 900 iterations to reach a very high cumulative reward. 900 iterations is significantly fewer than the 5,000 iterations required for the 2-DoF task to converge, which indicates that PRM's are very effective at reducing the convergence time of BECCA.

To better see the convergence speed of the PRM method compared to the engineering solution presented in the first 3 experiments, another experiment was performed as seen in Fig. 12. In this last experiment, two 3-DoF tasks are created, a simple and a hard task, using the same method as the 2-DoF task except a third joint is added. The simple 3-DoF task has 3 bins per joint, and an action vector of length 12. The hard 3-DoF task has 4 bins per joint, and an action vector of length 18. Both simple and hard tasks are rewarded by $+(100/3)$ for being in bin 2 and receive 0 reward for being in any other bin. The reward structure has been altered so that the simple and hard tasks have a reward structure more similar to the PRM task. The altered reward structure has a maximum reward of 100 per iteration

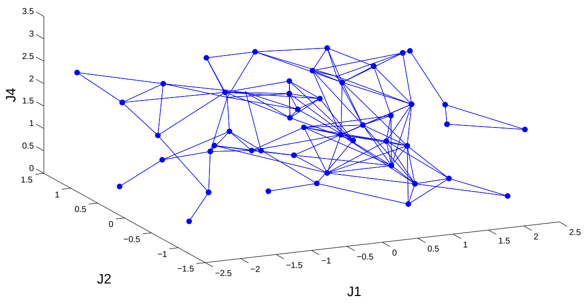

Fig. 10. Probabilistic Road Map for a 3-DoF WAM Task. Vertices are possible configurations. Edges are possible transitions between configurations.

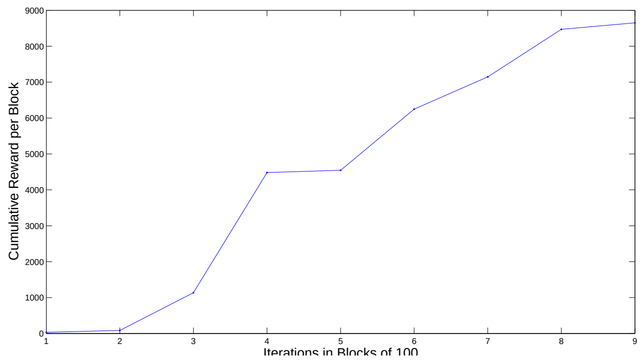

Fig. 11. PRM Cumulative Reward for a 3-DoF Task.

and thus 10,000 per block, just like the PRM experiment. The simple task has 27 possible states. The hard task has 64 possible states and the PRM has 50 states. Thus, the simple and hard tasks bound the PRM in number of states. However, it is important to note that the simple and hard tasks have larger action vectors than the PRM due to how they were engineered. Fig. 12 shows that the PRM method converges much faster than either the simple 3-DoF or hard 3-DoF task. The PRM method has reached the optimal of 7,000 units of reward by around 1,000 iterations while, the simple 3-DoF task has only reached approximately 6,000 units of reward by 7,000 iterations. The 3-DoF hard task has only reached approximately 3,500 by 7,000 iterations. Thus we can see that the PRM task converges much faster than either the simple or hard task.

To further show the scalability of the PRM approach we produce Fig. 13 which plots the average reward of 10 runs for each DoF from 1 to 7. This graph confirms that PRM-BECCA is unaffected by the Degrees of Freedom with a constant number of nodes. However, there is a problem with just testing the Degrees of Freedom and holding the number of nodes constant. By holding the number of nodes in the PRM constant, the density of nodes decreases as the DoF increases. Thus we must also test to see how BECCA scales with the number of nodes.

In the following experiments we vary the number of nodes from 60 to 200 in steps of 20, and the k neighbor parameter is

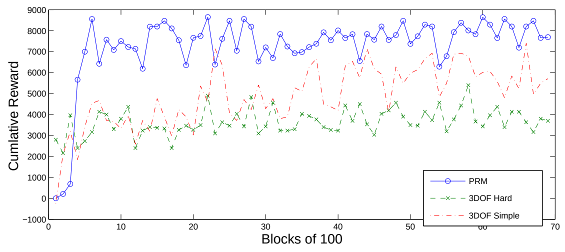

Fig. 12. Cumulative reward for PRM, 3-DoF Simple, and 3-DoF Hard tasks. The 3-DoF simple task has 3 bins per joint, giving a state space of 3^3. The 3-DoF hard task has 4 bins per and an action vector length of 18, giving a state space of 4^3. The PRM task has 50 points which correspond to 50 states.

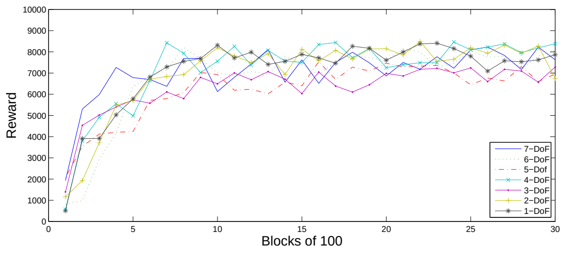

Fig. 13. Cumulative reward per block of 1-DoF to 7-DoF with PRMs.

set to 4. Since the previous experiment showed that BECCA would converge at the same number of steps regardless of DoF we chose to do this experiment with 3 DoF. Again 10 runs are done for each number of nodes and the results are averaged. Fig. 14 shows the average cumulative reward for each test. It shows us that BECCA also converges at the same time regardless of number of nodes in the graph. Fig. 14 is practically indistinguishable from Fig. 13, thus showing that BECCA converges at the same rate regardless of DoF and regardless of the number of nodes in the PRM.

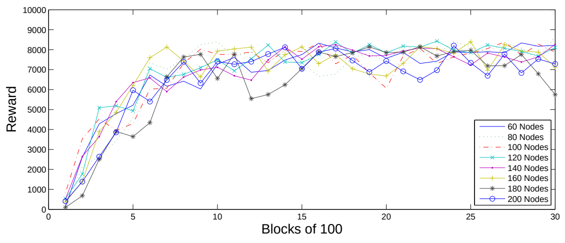

Fig. 14. Reward per Block for Varying Number of Nodes.

It is important to note that this is a novel use of PRMs. In previous work, they have been used to plan the motions for complex robot systems [7], [8], [9]. However, by integrating PRMs with BECCA, we are able to demonstrate automatic learning of controls to achieve motion in complex problems.

V. Conclusions

BECCA is intended to be a general reinforcement learning operating in unmodeled environments. The experiments in this paper demonstrate BECCA running on a single WAM platform under different constraints. BECCA performs very well on small state/action spaces as seen in the 1-DoF task, but struggles under larger state/action spaces as seen in the 2-DoF task. Large state/action spaces are a problem for many reinforcement learning algorithms. The experiments in this paper show that BECCA can learn how to operate a complex machine such as the WAM in state/action spaces with varying complexity. However, the experiments also show that the complexity of those environments has a large impact on the convergence time of BECCA. The hope is that BECCA can learn any unconstrained environment, but the complexity of the environment is such a large factor that realistically the state/action space of the environment has to be carefully engineered to insure feasible convergence times.

The PRM formulation of the problem demonstrates that careful construction of the state space allows the reinforcement learner to overcome the scalability issues. BECCA scales very nicely from 1 to 7 DoF and under varying numbers of nodes in the roadmap using the PRM formulation. However, it fails to scale using the binning formulation. Therefore, PRMs work to improve the efficiency of the BECCA reinforcement learning agent.

VI. Acknowledgments

This work was supported by Sandia National Laboratories PO# 1074659. Tapia supported in part by the National Institutes of Health (NIH) Grant P20RR018754 to the Center for Evolutionary and Theoretical Immunology. We also thank Dr. Dave Vick for his help with the robotic hardware setup.

References

[1] Abtahi, F., and Fasel, I. Deep belief nets as function approximators for reinforcement learning, 2011.

[2] Argall, B. D., Chernova, S., Veloso, M., and Browning, B. A survey of robot learning from demonstration. *Robot. Auton. Syst. 57* (May 2009), 469–483.

[3] Clouse, J. On integrating apprentice learning and reinforcement learning. Tech. rep., University of Massachusetts, Amherst, MA, USA, 1997.

[4] Cuccu, G., L. M.-S. J., and Gomez, F. Intrinsically motivated neuroevolution for vision-based reinforcement learnin. *International Conference on Development and Learning and Epigenetic Robotics* (August 2011).

[5] Fernández, F., and Borrajo, D. Two steps reinforcement learning. *Int. J. Intell. Syst. 23* (February 2008), 213–245.

[6] Flash, T., and Hogans, N. The coordination of arm movements: An experimentally confirmed mathematical model. *Journal of neuroscience 5* (1985), 1688–1703.

[7] Hashim, S., and Lu, T.-F. A new strategy in dynamic time-dependent motion planing for nonholonomic mobile robots. In *Proceedings of the 2009 international conference on Robotics and biomimetics* (Piscataway, NJ, USA, 2009), ROBIO'09, IEEE Press, pp. 1692–1697.

[8] Hauser, K., Bretl, T., claude Latombe, J., and Wilcox, B. Motion planning for a sixlegged lunar robot. In *The Seventh International Workshop on the Algorithmic Foundations of Robotics* (2006), pp. 16–18.

[9] Jung-Jun Park, J.-H. K., and Song, J.-B. Path planning for a robot manipulator based on probabilistic roadmap and reinforcement learning. In *International Journal of Control, Automation, and Systems* (2008), pp. 674–680.

[10] Kavraki, L., Svestka, P., claude Latombe, J., and Overmars, M. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *IEEE International Conference on Robotics and Automation* (1996), pp. 566–580.

[11] Knox, W. B., and Stone, P. Augmenting reinforcement learning with human feedback. In *ICML 2011 Workshop on New Developments in Imitation Learning* (July 2011).

[12] Legenstein, R., Wilbert, N., and Wiskott, L. Reinforcement learning on slow features of high-dimensional input streams. *PLoS Comput Biol 6*, 8 (08 2010), e1000894.

[13] MATLAB. *version 7.7.0 R2008b*. The MathWorks Inc., 2007.

[14] Rohrer, B. Biologically inspired feature creation for multi-sensory perception. *BICA* (2011).

[15] Rohrer, B. A developmental agent for learning feature, environment models, and general robotics tasks. *ICDL/Eprirob* (2011).

[16] Rohrer, B. A developmental agent for learning features, environment models, and general robotics tasks. In *Joint IEEE International Conference on Development and Learning and on Epigenetic Robotics* (2011).

[17] Rohrer, B. An implemented architecture for feature creation and general reinforcement learning. In *Workshop on Self-Programming in AGI Systems, Fourth International Conference on Artificial General Intelligence* (2011).

[18] Rohrer, B. BECCA: Reintegrating AI for natural world interaction. *AAAI Spring Symposium on Designing Intelligent Robots: Reintegrating AI 2012* (2012).

[19] Sutton, R., and Barto, A. *Reinforcement learning: An introduction*. Adaptive computation and machine learning. MIT Press, 1998.