Reproduction and Demonstration of The Crawler Reinforcement Learning Robot

Griswald Brooks

***Abstract*—Reinforcement learning is a diverse topic with many concepts and strategies. Using guidance from [1] a crawling robot was constructed and simulated as a platform for exhibiting reinforcement learning algorithms. The value iteration algorithm with an ε-greedy exploration policy was implemented in simulation and the algorithm produced a stable crawling gait.**

***Index Terms*—EL9223, project, report, reinforcement, learning, robot, crawler.**

I. Introduction

REINFORCEMENT learning (RL) is an approach used to find the optimal set of actions to perform in a given situation based on experience. RL is used in a broad variety of applications including economics [2], biology [3], and robotics [4].

The structure of RL algorithms has a number of components that it uses to accomplish its goal. The algorithm requires the system interact with the environment through action to gain experience. This experience is conceptualized as a reward/cost of interaction. After collecting sufficient experience the algorithm computes the optimal policy based on this experience. Finally, the algorithm needs a mechanism to ensure sufficient experience is gathered, usually requiring some non-optimal actions to be performed.

The goal is stipulated in the contest of observable rewards. These rewards for example could be reducing the distance to a point or accumulation of financial gain. The counter-part of cost could be the reduction of energy use. At times, there may be some ambiguity as to how the reward achieves the goal. The rewards could be minimizing distance to a point while reducing energy use to get there but the goal is to reach a certain point and maintain that position. The algorithm under a naïve reward structure could decide the best action is to do nothing. As a result a technique known as "reward shaping" is used to coerce the algorithm into planning towards the goal. These reward structures can sometimes be quite elaborate in order to produce an effective policy for complicated systems.

To compute the optimal action, a number of different techniques are available. Various techniques compute while the system is running (on-line), while in others the computation is done only after the rewards have been collected (off-line). Augmenting the on-line category is the idea of on-policy or off-policy. Off-policy techniques compute the optimal policy regardless of what the system is doing, so long as it sufficiently samples the state-space. On-policy techniques, require that the optimization computation affect the system behavior while the system is gathering experience.

G. Brooks is with the Department of Electrical and Computer Engineering, NYU Polytechnic School of Engineering, Brooklyn, NY, 11201 USA e-mail: griswald.brooks@nyu.edu

Manuscript received May 19, 2015.

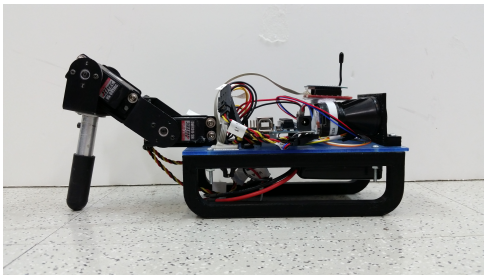

Fig. 1. Profile photo of the Crawler robot. The sled structure can be seen as the black rails and the two-link manipulator on the left for pulling the robot. The laser distance sensor for measuring distance to a wall can be seen on the right.

Some of the techniques used to compute the optimal action are Value Iteration, Policy Iteration, Q-Learning, and SARSA. Value Iteration is an off-line technique whereby the rewards for each of the states and actions is propagated to adjacent states (called values) as an incentive to get the surrounding states to prefer actions that lead to the high reward state. Policy Iteration is an off-line technique where the values for each state are computed while following a fixed policy. Then the policy is updated using the values. Q-Learning is an on-line, off-policy technique which observes the rewards for states, actions, and the proceeding predicted state and action as the system is running. It uses these values (called Q-values) to find the optimal policy. SARSA is an on-line, on-policy algorithm. Unlike Q-learning, it modifies the policy that it is executing while it is executing it in order to compute the optimal policy.

Lastly, the algorithm requires a strategy to sample the state and reward space, or rather, gain sufficient experience. Initially, the algorithm, by definition, does not know the optimal policy. Despite this, the system still has to perform actions in order to gain experience. At some point the algorithm will produce some policy that is optimal or nearly-optimal that should then be followed. The question becomes, at what point should the system prefer doing the perceived optimal action for gaining experience. This problem is called the "exploration-exploitation" trade-off. There are a number of strategies to tackle this problem, including ε-greedy, soft-max, and optimism-in-the-face-of-uncertainty. ε-greedy chooses between a random action and the perceived optimal according to a uniform distribution that thresholds the choice by some number ε. Softmax does something similar but chooses the actions with the higher probable reward preferentially. Finally, optimism-in-the-face-of-uncertainty gives unexplored states a

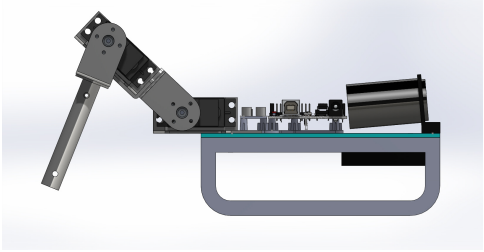

Fig. 2. CAD drawing of the assembled Crawler platform. The sled rails can be clearly seen in the lower part of the figure. The two-link arm can be seen on the far left with the laser distance sensor on the far right. The middle of the figure depicts the Arduino microcontroller used for onboard computation.

high initial reward to encourage the algorithm to explore them more often.

There are many elements that go with RL, which is essentially experience-based planning, each with many subcomponents. In order to provide an easily understandable example that illustrates all of the major concepts, a small mobile robot was built based on the work of [1]. It consists of a mobile base which is constructed as a sled with a small two-link manipulator used to pull the sled forward. The robot gets its reward via a laser distance sensor mounted on the vehicle that measures the distance to a wall. As the robot moves forward the distance reported from the sensor decreases. The reward structure on the robot considers this to be a positive reward. A picture of the implemented system can be seen in Figure 1. A simulation of the system implemented a form of the Value Iteration algorithm using the ε-greedy exploration scheme. The simulation was able to find a stable crawling gait that moved the robot forward.

This paper is organized as follows. Section II details the Crawler robot platform. Section III discusses the Value Iteration and ε-greedy algorithms. Section IV describes the simulation model. The algorithm implementation is reviewed in Section V with resultant policies discussed in Section VI. Finally, improvements are suggested in Section VII.

II. Crawler Platform

The Crawler platform was constructed with inspiration from Tokic's Crawler platform [1]. The platform was designed in CAD software and built using a combination of 3D printing and traditional 2D fabrication techniques. A schematic depiction of the assembled system can be seen in Figure 2. Parts that could be readily assembled were utilized when feasible, such as the brackets that form the arm. The system features power regulation, onboard microcontroller, wireless communication, and laser distance sensing. The system is powered by a 12 V lithium polymer (LiPo) battery. The total system cost is about $\$500$, though the price is dominated by the laser distance sensor which could be replaced by a less capable sensor, bringing the price down to less than $\$250$. The assembled vehicle weight is 1.2 lbs.

The construction has three major sections, the sled, the arm, and the electronics payload.

A. Sled

As the intent of the platform was the implement a robot whose only means on locomotion was an actuated arm, the platform needed to have as its base a form that was conducive to a sliding motion. Casters or passive wheels could have been used to implement this motion, but as the robot was not to use wheel encoders to implement the reward signal (as in Tokic's Crawler), runners were chosen for their mechanical simplicity and ease of construction. They were designed to be rounded on their ends in order to avoid sharp edges that might catch on ground surfaces such as carpets or table cloths. Their height was designed to be sufficient as to clear any payloads mounted on the underside of the main platform, but not so high as to raise the center of mass of the robot to create lateral instability. They were 3D printed and attached to a flat plastic sheet to which all other components were mounted.

B. Arm

The arm of the Crawler was designed as a two-link manipulator whose axis of motion were parallel. The joint axes are aligned as to actuate forward motion only of the platform and to minimize any turning behavior that might corrupt the distance readings of the laser sensor as an indication of the robot's forward motion. Such an action might cause the algorithm to reward unintended turning motions rather than the intended forward motion. This is an example of how rewards might not always reflect the intended goals of the system. The restriction of the number of joints to two, rather than a greater number that could produce more comprehensive crawling gaits, is to provide a simplified state space for the learning algorithm to work on. This is less for the benefit of the algorithm and more for demonstration purposes while observers watch how the value structure is formed. The state space for the system consists of the angles of the joints and therefore would be more difficult to visualize in dimensions greater than two. The joints are implemented using two HS-645MG hobby PWM servos. Each servo has a maximum torque of 133 oz-in. Each link length is 3 in and so can produce a force of 2.75 lbs easily actuating the 1.2 lb vehicle. Each servo has a range of 180 degrees and is controlled by a PWM (pulse-width modulated) signal produced my the microcontroller which is mapped to an angle command by the servo. The links are built using the Lynxmotion Servo Erector Set [5]. This is a commercially produced bracket system designed for making multi degree-of-freedom arms using hobby servos. It is inexpensive and easy to assemble and reconfigure.

C. Electronics Payload

The onboard electronics power and control the Crawler arm. It consists of a 12 V 2 Ah LiPo battery connected to a 5 V 5 A switching regulator to power the arm and onboard wireless system. The battery also powers the onboard Arduino Uno [6] microcontroller which sends PWM control signals to the arm in order to command it to joint angles. The Arduino has a 16 Mhz Atmel processor with 32 Kb of flash memory and 2 Kb of RAM. It is easily programmed via USB bootloader

and open-source IDE, The Arduino also provides regulated power to the LightWare SF10/A laser altimeter [7]. This is a time-of-flight laser rangefinder with a maximum range of 25 m and 5 cm accuracy and 1 cm resolution. The sensor can update at a rate of 32 Hz. The distance readings are reported via a 12-bit digital-analog converter that is connected to the microcontroller's 10-bit analog-digital converter. While this method is convenient for the current application, the readings are also available via USB, TTL UART, or I2C serial communication from the sensor. The microcontroller is also connected to a 2.4 GHz XBee Pro radio [8] via its TTL UART. This allows communication with an offboard computer equipped with a corresponding XBee at a rate of 56700 baud. With this complement of sensors, computation, and communication the platform is capable of producing joint angle commands that pull the robot forward, sensing the environment, and communicating with a ground station computer to report state information or command the platform.

III. RL Algorithms

While there are a rich and varied assortment of reinforcement learning algorithms that could be implemented on the Crawler robot, as the first purpose of the system is to demonstrate RL concepts the more basic strategies were chosen for this project. They are the Value Iteration algorithm for computing the optimal policy and ε-greedy exploration mechanism. What follows is a review of the two algorithms.

A. Value Iteration

The fundamental idea behind the Value Iteration algorithm is that each state should choose actions in order to maximize their future expected reward. In general, during the runtime of the system, a series of actions will be performed and a series of states will be visited. Depending on the system, rewards may come as a consequence of being in a certain state or from doing an action while in a specific state. As the series is performed, the system should seek to maximize the sum of the experienced rewards. This accumulation of rewards captured in equation 1.

$$V_0 = \mathbb{E}[\sum_{t=0}^{\infty} R_t] \tag{1}$$

Where R_t is the reward experienced at time t and V_0 is the accumulated reward starting from time $t = 0$ Typically, rewards are not random or only a function of time or iteration, but rather a result from performing some action in some state $r_t = r(s_t, a_t)$. Here r_t denotes the reward experienced at time t, while in state s_t and doing a_t according to some reward function r. This reward function is typically not known and is a function of both the dynamics and the environment whose state is usually not directly observable. In this case rewards from state-action pairs must be experienced via the system and then remembered for future use in computation. Given this and manipulation of equation 1, a new equation can be written as equation 2

$$V_0 = r(s_0, a_0) + \mathbb{E}[\sum_{t=1}^{\infty} r(s_t, a_t)] \tag{2}$$

For equation 2, we assume that the system is in some initial state s_0 and performs some action a_0, in which case, the reward looses its expectation as it has been observed. Inspecting the part of the equation still under expectation, we can see it is simply V_1. In which case the equation can be rewritten as:

$$V_0 = r(s_0, a_0) + V_1 \tag{3}$$

If we seek to maximize equation 3 through action, we arrive at equation 4.

$$V(s_0) = \max_{a \in A} \Big[r(s_0, a) + V(s') \Big] \tag{4}$$

Where we now perform different actions a from a set of possible action A to try and maximize the expected cumulative reward by sampling the reward function and expected future rewards. The set A in equation 4 is fixed but can be a function of the state. The subscripts from V_t have been modified to be functions of the state, as the are no longer a function of time but affected by the action performed which achieves a state. s' is the state achieved by doing an action a in state s_0. As this equation is recursive and looks the same from which ever state the system is currently in, its final form is:

$$V(s) = \max_{a \in A} \Big[r(s, a) + V(s') \Big] \tag{5}$$

This is known as the Value Iteration equation and we called the expected cumulative reward V the value for a state s. So has been achieved the basic idea. We choose actions based on the current reward and what we expect the future rewards to be.

B. ε-Greedy

The fundamental problem with performing the optimal action initially is that very little has been experienced by the system, so the initial policy is almost guaranteed to be suboptimal. Under these conditions, it is better to choose actions randomly and gain experience than to follow a suboptimal policy which is likely to not improve system experience. The idea behind the ε-greedy algorithm is to choose actions randomly according to if some random number $\psi \in [0, 1]$ is less than some number $\varepsilon \in [0, 1]$ and perform the perceived optimal action otherwise. The hopes with this strategy is that by random chance enough of the reward space will be explored such that the computation strategy can deduce the optimal policy. The algorithm can be seen as follows:

$\psi \leftarrow rand([0, 1])$
if $\psi \leq \varepsilon$ **then**
  $a \leftarrow rand(A)$
else
  $a \leftarrow \arg\max_{a \in A} \Big[r(s, a) + V(s') \Big]$
end if

Simple implementations of this strategy keep ε constant, but it can also start with a value close to one and then decay to a small constant value. The intuitive reasoning behind this is that when the system has experienced very little it should act very randomly to gain more experience and after it has gained more experience take advantage of its knowledge and planning.

IV. Simulation Model

In order to test out the different learning algorithms in simulation a system model had to be constructed. When modeling the robot, it can be seen that it has two major dynamics. It has one kind of dynamics when the end effector is in the air and a different dynamics when is it touching the ground. Therefore, the model used in this implementation is a piecewise kinematic manipulator model with two modes, an open chain kinematics mode and a closed chain kinematics mode. A diagram of the two modes can be seen in Figure 3. In this figure, it is shown that the hypotenuse of the sled itself can be modeled as a link $l_1$that goes between the base of the arm and the opposite corner of the vehicle. In the simulation, the conditions that prescribe which model the simulation is following is based on whether the angle that l_1 makes with the ground would in reality put the sled into the ground. In this case, switch to the open chain mode. The second condition is based on if the new state would attempt to put the end effector into the ground. If so, then switch to the closed chain mode. The implemented model uses only manipulator kinematics rather than dynamics as the system moves at a bandwidth that is largely static and therefore can be approximated by its simpler kinematics rather than its more detailed dynamics. Equation 6a and 6b show the kinematic equations for the two different modes.

$$\begin{aligned} x_3 &= x_0 + l_1 cos(\theta_1) + l_2 cos(\theta_{12}) + l_3 cos(\theta_{123}) \\ y_3 &= l_1 sin(\theta_1) + l_2 sin(\theta_{12}) + l_3 sin(\theta_{123}) \end{aligned} \tag{6a}$$

$$\begin{aligned} x_3 &= x_0 + l_1 cos(\theta_1) + l_2 cos(\theta_{12}) + l_3 cos(\theta_{123}) \\ y_3 &= l_1 sin(\theta_1) + l_2 sin(\theta_{12}) + l_3 sin(\theta_{123}) \\ \theta_1 &= atan2(-b, a) \\ b &= l_2 sin(\theta_2) + l_3 sin(\theta_{23}) \\ a &= l_1 + l_2 cos(\theta_2) + l_3 cos(\theta_{23}) \end{aligned} \tag{6b}$$

Equation 6a is the typical kinematic equations for a three-link manipulator, where θ_{12} refers to the sum of $\theta_1 and \theta_2$ and θ_{123} refers to the sum of $\theta_1 \theta_2 and \theta_3$. The first two lines of equation 6b are the same as 6a as the closed chain must observe these constraints, but the last three lines put a constraint on the value of θ_1 to values that produce an end effector vertical position y_3 equal to zero. These were derived by solving the inverse kinematics for equation 6a.

V. Implementation

As per Tokic's implementation, the state space was taken as the different joint angles the Crawler could take on. It was limited to five possible angles, with each joint spanning the same magnitude of range, with different starting points. The joint angle ranges implemented can be seen in Table I

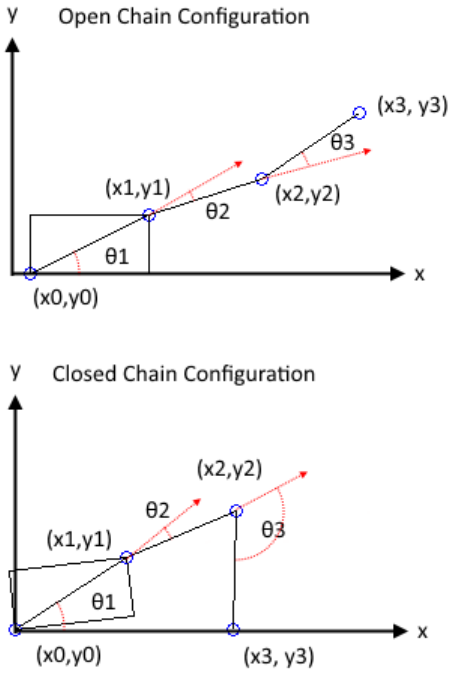

Fig. 3. Kinematic configurations of the Crawler robot. The upper image shows the platform in the open chain mode where the end effector is not touching the ground. The lower image shows the robot in the closed chain mode. In this case the robot is pushing itself up against the ground and hinging on its back corner. The blue dots signify joint positions parameterized via points on the plane (x_i, y_i). The angles of each joint are formed between the current link and the projection of the previous link. They are parameterized as θ_i.

TABLE I
Joint Angle Limits in Degrees

Joint	Min Angle	Max Angle	Angle Increment
1	-20	60	16
2	-180	-100	16

This range of joint angles and angle increment proved to be good fit for this configuration. The state space for the system was implemented as a 5 x 5 matrix with the row indices representing the angles for joint 1 and the column indices for joint 2. The system was modeled to be deterministic and so no probabilities were explicitly introduced.

The policy map was implemented as two 5 x 5 matrices, each representing the actions to be taken for each of the joints.

The action space for this robot was state dependent as when the robot was in an edge state (a state whose index was in either the first row or first column) it was not valid for the robot to explore past that space or to wrap around to the opposite edge. A function was written that computed the action set for each state. The basic action set allowed the state to transition along the cardinal direction, up, down, left, or right, but not both simultaneously. For example, the system could not transition diagonally.

The reward map was implemented as a 25 x 4 matrix. While this produces 100 state action pairs and not 80 as is in the Crawler system, the ease of implementation vs the cost of unused memory meant the extra states went unused. The

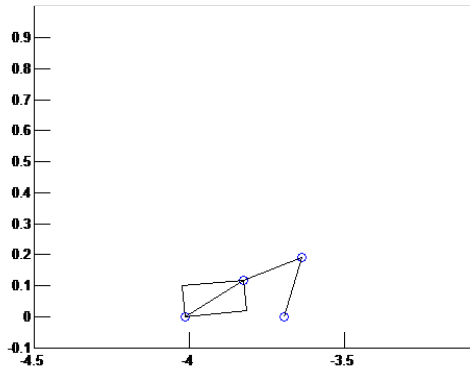

Fig. 4. Sample image from the crawler simulation. In this image, the value iteration algorithm has converged on a gait that moves the robot forward.

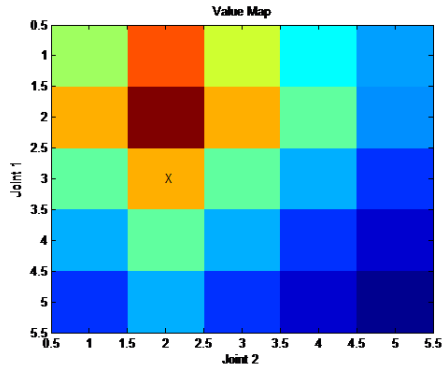

Fig. 5. Value map of the converged gait. In this representation, redder colors have a higher value and bluer colors have a lower value. The algorithm found a limit cycle between joint 1 angles $-20°$ and $-36°$ while keeping joint 2 at a constant $-164°$. That moved the robot forward.

rewards were updated according to the following formula:

$$r(s,a)_{n+1} = (1-\alpha)r(s,a)_{n+1} + \alpha r_t \tag{7}$$

Where r_t is the current reward sample and α was chosen to be 0.3. This basic form of low-pass filtering was implemented to reduce the impact of possible spurious reward samples. The rewards in the simulation were taken to be the change in the x position of the robot from a state s_t to a state s_{t+1}. The reward function was chosen to be:

$$r = -(50(x_{t+1} - x_t))^5 - 1; \tag{8}$$

The small negative constant discouraged the robot from staying in states where it did not move. Even if the neighbors or that state had values negative compared to the current state, eventually this negative constant would integrate into a value that would cause the system to transition to a different state.

The Value Iteration algorithm implemented updated all of the states once after every sample. In this way it was an on-line on-policy algorithm closer to SARSA. It was a discounted algorithm with γ equal to 0.75;

The ε-greedy was implemented to be time varying. It was initially 0.995 and decayed according to ε^i where i is the iteration time step, limited to a final ε of 0.1. This seemed to successfully encourage the robot to visit many states early on, while converging to an amount of randomness that allowed it to be successful once a policy had been learned.

VI. Results

The simulation was set to run over 5000 iterations in order to observe robot behavior. After about 200 iterations and a $\varepsilon = 0.3$ the robot had converged on a value map that produced a successful gait. The visualization of the value and policy map was displayed while the algorithm was learning, so anecdotally it was clear to see that after even 100 iterations the crawler had produced a value map that was very close to the final map. Figure 5 shows a depiction of the final map. It shows a very high value in the region of joint 1 angles $-20°$ and $-36°$ and joint 2 at $-164°$. All other states are directed to these states and they are directed to each other, forming a limit cycle that moves the robot forward.

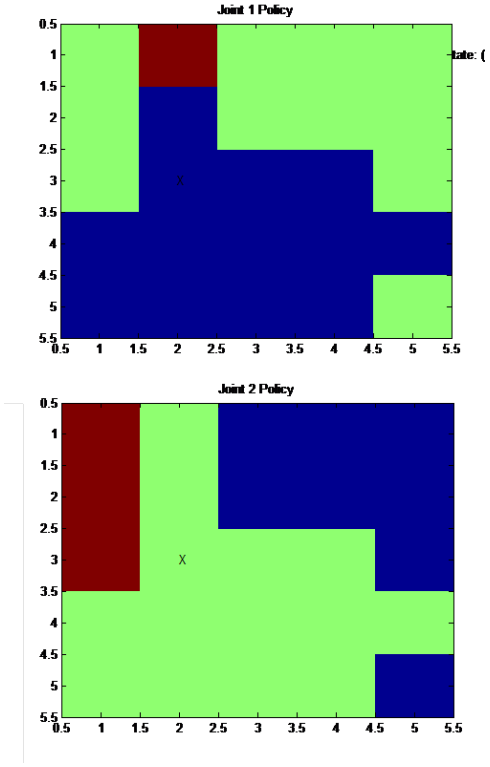

Fig. 6. Policy for the converged gait. In the figure, the green color indicates no change in state, while the red means to increase state value and the blue to decrease state value.

Figure 6 shows the policy that results from the value function. For the policies, blue means decrease joint angle, red means increase joint angle, and green means to make no change to the joint angle. As can be inferred from the value map, joint 2 is held at a constant in the limit cycle region and joint 1 switches between two states.

VII. Conclusion

The intent of the project was to implement a simple system that could demonstrate the basic concepts of reinforcement

learning. The result is a system that has a state space and policy map that is easy for an observer to read and interpret while still having a system requiring search and iteration to solve. The basic concepts of reward, value, and policy are shown while the importance of initial explorations are given. As an implementation exercise, a number of points were learned.

Tuning of the joint angle ranges and increments were important to allow the algorithm to find a successful policy. Given that the implemented Value Iteration algorithm can only gain rewards from one future action, and not the sequence of actions required to perform more complex gaits (whose intermediate states would come without reward), the state space had to be restricted so the scheme would only search in that space. The discretization being coarse allowed each action to have a bigger positive or negative effect on the system and differentiate state values more clearly.

Crafting the reward function to have a small negative reward when not movement of the robot was otherwise occurring was instrumental to getting the algorithm out of spurious local minima and encourage the robot to explore more states. As times, while exploring, the robot would receive a positive reward for a state action pair that later would not be reproduced. This reward would persist in the value map as it was larger than its neighbors. This would encourage the robot to perform actions that would produce no movement. Adding a small negative reward helped solve this problem.

In order to get a robot to exhibit a more complex behavior, algorithms that consider a whole sequence of actions, such as SARSA(λ) should be tried. This freedom to reward sequences more directly would reducing the amount of state space and reward shaping required to implement a complex system. Finally, while a discretized state space is a convenient learning tool for such systems, when exploring higher dimensional state spaces with a broader range of possible values, value tables will have to give way to function approximation in order to make computation tractable.

ACKNOWLEDGMENT

The authors would like to thank Professor Yezekael Hayel from the University of Avignon, Avignon, France, and Professor Quanyan Zhu from NYU Polytechnic School of Engineering, Brooklyn, NY for teaching the EL 9223 Reinforcement Learning class, for which the principles of reinforcement learning might have otherwise been hard gained and not allowed this project to come to fruition.

REFERENCES

[1] M. Tokic, J. Fessler, and W. Ertel, "The crawler, a class room demonstrator for reinforcement learning," in *Proceedings of the 22th International Florida Artificial Intelligence Research Society Conference FLAIRS'09*, C. Lane and H. Guesgen, Eds. Menlo Park, California, USA: AAAI Press, 2009, pp. 160–165. [Online]. Available: publikationen/papers/flairs09.pdf

[2] J. Rust, "Using randomization to break the curse of dimensionality," *Econometrica*, pp. 487–516, 1997.

[3] G. Czibula, M. iuliana Bocicor, and I. gergely Czibula, "A reinforcement learning model for solving the folding problem."

[4] J. Kober, J. A. D. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *International Journal of Robotics Research*, July 2013.

[5] Lynxmotion - servo erector set. [Online]. Available: http://www.lynxmotion.com/c-73-servo-erector-set.aspx

[6] Arduino - home. [Online]. Available: http://www.arduino.cc

[7] Sf10/a (25 m) - lightware. [Online]. Available: http://www.lightware.co.za/shop/en/rangefinders-and-altimeters/33-sf10a.html

[8] Xbee(r) digimesh(r) 2.4 - digi international. [Online]. Available: http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/zigbee-mesh-module/xbee-digimesh-2-4