Online Control Policy Search for the Robotic All-Terrain Surveyor using Reinforcement Learning

Rolf Allan Luders
Robotics Institute
Carnegie Melloin University
Pittsburgh, PA 15213
rluders@andrew.cmu.edu

Krishnan Ramnath
Robotics Institute
Carnegie Mellon Univeersity
Pittsburgh, PA 15213
kramnath@cmu.edu

1 Introduction and Problem Formulation

The objective of this project is to devise an online control policy for a particular multi-legged robot. The conceptual robot named robotic all-terrain surveyor (RATS) has 12 legs equally distributed over a spherical surface, approximately the size of a soccer ball. The objective of this system is to have the capability to circumvent complex obstacles through rough terrain. Each leg consists of a pneumatic piston that has enough power to make the whole robot hop. The concept of this robot was made by Boeing Corporation, and it is being designed and developed by the Robotics Institute at Carnegie Mellon University.

The twelve legged version of the robot is still on its design phase and no prototype has been built yet. Since the system is highly complex a planar version of the robot has been built with only five legs in a wheel shaped body (see Figure 1.) The idea of the planar version is to decompose the system into a simpler form to eventually understand better the three-dimensional version.

This work focuses mainly in trying to implement a motion control strategy for the planar robot based on unsupervised learning that will allow us to understand better the timing and action space of the final system. Specifically, we wish to find an efficient way to start RATS from an initial rest position and speed it up as we proceed. More often than not, conventional deterministic control strategies prove to be a very bad starting point in terms of the amount of time taken to get the robot to achieve a necessary speed in the right direction. A natural consequence is to use an unsupervised learning algorithm such as Reinforcement Learning to learn an online control policy to do exactly this.

Reinforcement Learning (RL) refers to a class of problems in machine learning which postulate an agent exploring an environment in which the agent perceives its current state and takes actions. The environment, in return, provides a reward (which can be positive or negative.) Reinforcement Learning algorithms attempt to find a policy for maximizing cumulative reward for the agent over the course of the problem.

There are three fundamental parts of a Reinforcement Learning problem: the environment, the reinforcement function, and the value function. Every RL system learns a mapping from situations to actions by trial-and-error interactions with a dynamic environment. For the control task at hand, the feedback from the environment is obtained through the optical encoders that measure the angle tilt and angular velocity of the robot. The "goal" of the RL system is then defined using the concept of a reinforcement function, which is the exact function of future reinforcements the agent seeks to maximize. In other words, there exists a mapping from state/action pairs to reinforcements; after performing an action in a given state the RL agent will receive some reinforcement (reward) in the form of a scalar value.

In our case, the reinforcement function will be in terms of the angular velocity, which we try to maximize. We give the robot a positive reward if it produces an angular velocity greater than a certain threshold, in a particular direction (clockwise or anti-clockwse) and a negative reward in all other cases, for each trial. Also, one of the interesting aspects of this problem is that the action space is discrete and well-defined. For RATS the actions consist of firing one of the five pistons (1-5) or none (0), thereby providing thrust to any one of the five legs of the robot or doing nothing at each step.

Since the action space is low-dimensional and discrete for the task at hand, we find that a majority of Reinforcement Learning techniques can be applied to this problem, as it will become clear in later sections.

The third element in the Reinforcement Learning process is to address the issue of how the agent learns to choose "good" actions, or even how we might measure the utility of an action. For this we define two terms - policy and value. A policy determines which action should be performed in each state; a policy is a mapping from states to actions. The value of a state is defined as the sum of the reinforcements received when starting in that state and following some fixed policy to a terminal state. The value function therefore is a mapping from states to state values and can be approximated using a variety of Reinforcement Learning techniques that can be used to perform this task.

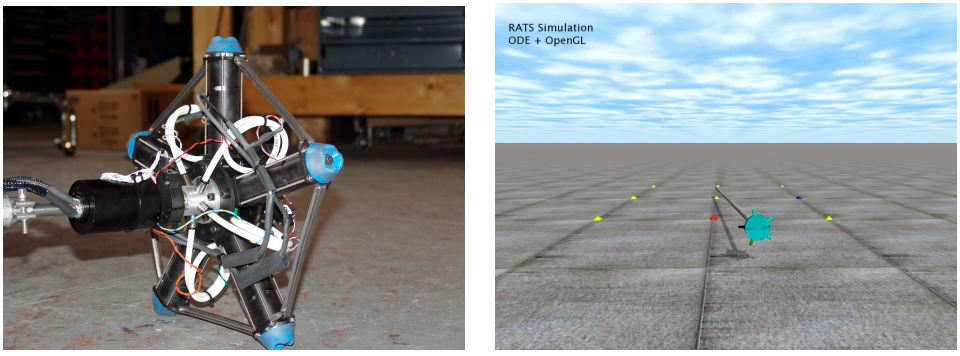

Figure 1: Actual Robot Robot Simulator

Choosing the best one of the numerous techniques available to us is important and we focus a major part of this report on that. We wish to find an optimal learning algorithm that is tailored to work best for the definition of the environment, state/action space and reward function for RATS. In this report we consider four different Reinforcement Learning algorithms[1] - SARSA (Single-Step), Q-Learning (Single-Step), SARSA (Lambda - Eligibility Traces) and Q-Learning (Lambda - Eligibility traces.) We implemented each of these algorithms in C and tested them on a RATS simulator, which we built using the Open Dynamics Engine (ODE.) We found that both SARSA and Q-Learning algorithms with eligibility traces converged the fastest as compared to the other two. However, the One-Step Q-Learning algorithm performed the best in terms of achieving the highest angular velocity of the robot. We discuss the algorithms and the experimental results in detail in the next sections.

2 Methodology

2.1 Intuition

The task at hand is to get an all-terrain robot to perform navigation tasks with minimal or no supervision. A natural choice of learning algorithm would be to use one that is completely unsupervised and works based on some optimality condition. The algorithm should learn by acting and correcting itself based on feedback from the environment. Reinforcement Learning techniques help us to do exactly this and hence were primary candidates for our task. Also the results obtained from our implementations of these algorithms are highly encouraging and stand testimony to the fact that these algorithms actually work well for the task that we have. We also wish to point out that application of Reinforcement Learning techniques to the RATS framework has never been done before, to our knowledge; we consider our work novel in that aspect. Hence, there is no state-of-art technique that we can compare our methods to. Rather we perform a comparison of various existing RL techniques for this particular task.

2.2 Proposed method and Algorithms Description

The objective of this project is to devise an efficient way to come up with a suitable starting point for RATS robot so that the robot quickly tends towards achieving its maximum angular velocity. We

[1]We narrowed down our choices of the learning algorithms after a careful analysis of previous work in Reinforcement Learning and the problem formulation. For a detailed analysis of related work, please refer to the midterm report.

wish to summarize here the environment, actions, reward function and goal of the Reinforcement Learning problem that we are trying to formulate for the control of RATS:

1. **Environment**: Feedback from the environment is obtained through the optical encoders that measure the angle tilt and angular velocity of the robot.
2. **Goal**: Maximize the angular velocity of the robot.
3. **Action**: Open each one of the piston valves (1-5) or none (0), thereby providing thrust to only one of the five legs of the robot or doing nothing at a particular time instant.
4. **Reward**: Positive reward if it produces an angular velocity greater than a certain threshold, in a particular direction (clockwise or anti-clockwse) and a negative reward in all other cases.

Having briefly formulated the RL framework for RATS, we wish to now analyze which of the popular RL techniques can be applied to this particular problem. We begin by noting that our action space is low-dimensional and discrete and hence well-defined. Also, we wish to have an algorithm that learns quickly and hence provides a suitable starting point for the robot that is better than the manual control initialization.

After careful survey of existing literature we have narrowed down the choice of learning algorithms to SARSA and Q-Learning algorithms. They are both popular learning techniques and have been widely used in many control tasks. Also, these algorithms work really well when the space of all possible actions is low-dimensional and discrete, which is exactly the case for RATS.

SARSA and Q-Learning are both similar algorithms except that Q-Learning is off-policy (not model based.) It is based on value-iteration[2] (truncates policy evaluation at each iteration) rather than policy-iteration (perform policy evaluation at each iteration.) SARSA is a variant of Q-Learning that is based on policy-iteration. Both algorithms converge to optimum although SARSA can be faster in case the action space has high cardinality. We implement two different variants of each one of these and compare their performances for the task at hand. We also implemented a version of on-policy Actor-Critic algorithm based on [8], but found its performance to be poor on our task as compared to the SARSA or Q-Learning algorithms and its variants, hence we exclude it from our comparisons. In summary, the four algorithms that we compared were :

1. SARSA (One Step TD Control)
2. Q-Learning (One Step off-policy TD Control)
3. SARSA (Lambda - Eligibility Traces)
4. Q-Learning (Lambda - Eligibility Traces)

We explored the SARSA and Q-Learning techniques with eligibility traces to obtain more general methods that may learn more efficiently. An eligibility trace (given by the lambda parameter) is a temporary record of the occurrence of an event, such as the visiting of a state or the taking of an action. The trace marks the memory parameters associated with the event as eligible for undergoing learning changes. When a Temporal Difference (TD) error occurs, only the eligible states or actions are assigned credit or blame for the error. Thus, eligibility traces help bridge the gap between events and training information. Like TD methods themselves, eligibility traces are a basic mechanism for temporal credit assignment.

In the next few sections we shall illustrate the algorithmic steps for each method described above.

2.3 SARSA Algorithm

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initilaize s
  Choose a from s using policy derived from Q (e.g., using ϵ-greedy)
  Repeat (for each step of episode):

[2]For more information on value and policy iterations, see [1]. We omit these details in the report due to lack of space.

Take action a, observer r, s'
Choose a' from s' using policy derived from Q (e.g., using ϵ-greedy)
$Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma Q(s',a') - Q(s,a)]$
$s \leftarrow s'; a \leftarrow a'$
until s is terminal

In the algorithm ϵ-greedy stands for a greedy policy search where it chooses the previous best action each time except during ϵ percent of the time where it chooses a new random action.

2.4 Q-Learning One-Step Off-Policy Algorithm

One of the most important breakthroughs in Reinforcement Learning was the development of an off-policy TD control algorithm known as Q-learning (Watkins, 1989.) Q-Learning is perhaps the most popular of reinforcement techniques that are being used till date. The One-step Q-Learning update rule is given by:

$$\mathbf{Q}(s_t,a_t) \leftarrow \mathbf{Q}(s_t,a_t) + \alpha [r_{t+1} + \gamma max_a \mathbf{q}(s_{t+1},a) - \mathbf{Q}(s_t,a_t)] \quad (1)$$

In this case, the learned action-value function, $\mathbf{Q}$, directly approximates $\mathbf{Q}^*$, the optimal action-value function, independent of the policy being followed. This dramatically simplifies the analysis of the algorithm. The policy still has an effect in that it determines which state-action pairs are visited and updated. However, all that is required for correct convergence is that all pairs continue to be updated. The Q-Learning algorithm described here can be obtained by replacing the learning rule described in the SARSA algorithm with Eq 1 and performing some other minor modifications. Please see [1] for details.

2.5 SARSA (Lambda)

In order to change the One-step version of SARSA to the one with eligibility traces, we need to define a trace for each state/action pair which we denote as $e(s,a)$. The update equation becomes:

$\mathbf{Q}_{t+1}(s,a) = \mathbf{Q}_t(s,a) + \alpha\delta_t e_t(s,a)$, for all s, a

where $\delta_t = r_{t+1} + \gamma \mathbf{Q}_t(s_{t+1},a_{t+1}) - \mathbf{Q}_t(s_t,a_t)$

and

$$e_t(s,a) = \left\{ \begin{array}{ccc} \gamma\lambda e^{t-1}(s,a) + 1 & \text{if} & s = s_t \text{ and } a = a_t; \\ \gamma\lambda e_{t-1}(s,a) & \text{otherwise} & \end{array} \right\}$$

2.6 Q-Learning (Lambda)

The update rule for Q-Learning with eligibility traces (a combination of the Q-Learning One-step algorithm with the eligibility traces inclusion described in the previous section) is given by:

$\mathbf{Q}_{t+1}(s,a) = \mathbf{Q}_t(s,a) + \alpha\delta_t e_t(s,a)$, for all s, a

where $\delta_t = r_{t+1} + \gamma max_{a'} \mathbf{Q}_t\left(s_{t+1},a'\right) - \mathbf{Q}_t(s_t,a_t)$

and

$$e_t(s,a) = I_{ss_t}\dot{I}_{aa_t} + \left\{ \begin{array}{ccc} \gamma\lambda e^{t-1}(s,a) + 1 & \text{if} & \mathbf{Q}_{t-1}(s_t,a_t) = max_a \mathbf{Q}_{t-1}(s_t,a_t); \\ 0 & \text{else} & \end{array} \right\}$$

where I_{xy} is an identity indicator function with is 1 if $x = y$ and 0 otherwise.
We shall discuss the implementation details of all four algorithms along with the results in the next section.

3 Experimental Results

We test all our algorithms on a robot simulator built on Open Dynamics Engine (ODE) running on Linux. We coded the RATS simulator on top of ODE and also implemented the four learning

Parameter	Discrete Steps	Range
Wheel Angle [rad]	31	from (-pi - 0.2) to (pi + 0.2)
Wheel Angle Rate [rad/s]	31	from (-10.1) to (21.0)
Height Angle [rad]	2	from (-2.0) to (0.1)
Height Angle Rate [rad/s]	2	from (-2.0) to (2.0)

Table 1: This table shows the parameters of the state space. We have four states as shown above that are discretized into fixed set of values within a specified range.

Algorithm	Steps	e-greedy Discount	Update Parameters
SARSA (One-Step)	[1 10 1000]	epsilon = 0.99*epsilon	α = 0.3, γ = 1.0
Q-Learning (One-Step	1000	epsilon = 0.99*epsilon	α = 0.3, γ = 1.0
SARSA (Lambda)	1000	epsilon = 0.95*epsilon	α = 0.3, γ = 0.8, λ= 0.9
Q-Learning (Lambda)	1000	epsilon = 0.95*epsilon	α = 0.3, γ = 0.8, λ= 0.9

Table 2: This table summarizes the parameter values that we chose for each algorithm that we tested. The SARSA (One-Step) algorithm was tested with varying steps to show the learning behavior. The e-greedy discount (given by epsilon) decides the trade-off between previous and random choices. We decrease the discount parameter to reduce choosing random actions as the iterations increase. The update parameters (α - learning rate, γ - discount factor and λ - traceability) are estimated empirically on a trial-and-error basis to give optimal outputs.

algorithms discussed in the previous section in C. In this section the experimental results of the simulation runs are presented. We wish to point out that RATS is governed by four parameters - the Wheel Angle, Wheel Angle Rate, Height Angle and Height Angle Rate. All these are continuous variables and hence our state space is a continuous space in four dimensions. To make it tractable we discretize each state into a certain number of steps as summarized in Table 1. The number of discrete steps for each state and the range of values are illustrated in the table.

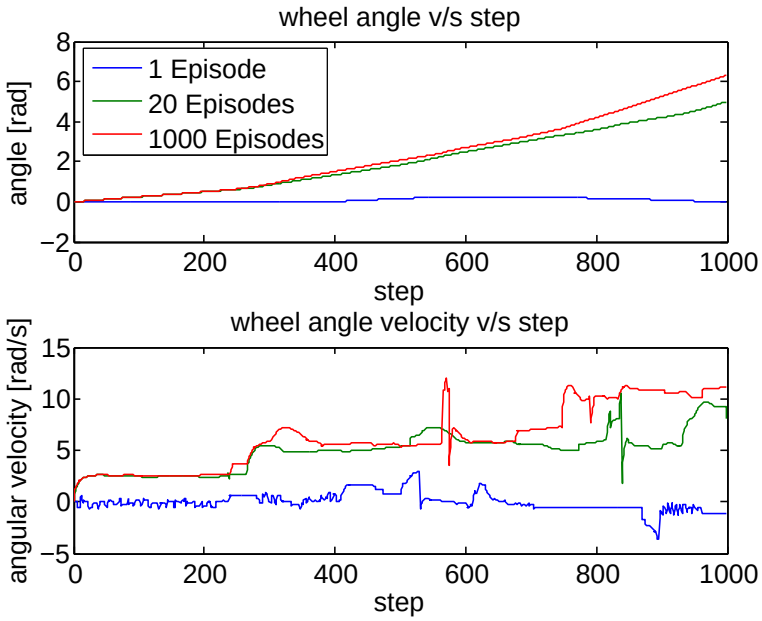

Figure 2: The angle of the wheel versus the step at different episodes using One-step SARSA algorithm.

The parameters for all four algorithms that were chosen after trying several different combinations are summarized in Table 2. To show how the learning algorithms improve with respect the number of episodes, three snapshots of the learning process are presented in Figures 2 and 3 using the One-step SARSA algorithm.

The top plot of Figure 2 compares the angle of the wheel versus the step at different episodes. With one episode (blue line), the wheel barely moves. Since the objective of the problem is to make the wheel rotate as fast as possible in one direction, this solution is not satisfactory. If we now allow the algorithm to run for 20 episodes, the solution presented improves dramatically (green line). Now it can be seen that the wheel rotates almost 5 radians. Finally, if it runs till the total reward per episodes converges, the result is even better (red line). The bottom plot from Figure 2, shows the angular velocity of the wheel, which is the value to optimize. It is clear that with 1000 episodes the solution is very good, and the initial acceleration is very fast.

Figure 3 shows which actions were performed at each step. Each of the three plots were generated after a different number of episodes. The Y axis is the number of the leg that was fired, where 0 means no leg was fired, and numbers 1 to 5 indicate which of the five legs was fired at each time step.

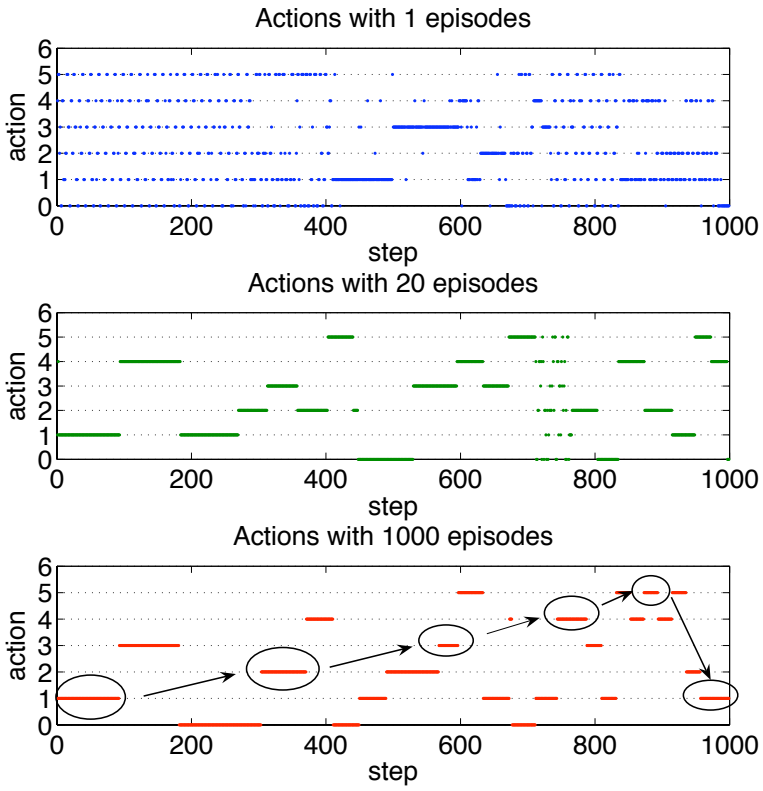

Figure 3: The actions performed at each step using One-step SARSA algorithm.

It is interesting to note that with one episode (the uppermost plot) the actions chosen look randomly distributed and do not make much sense. On the other hand, the bottom plot shows the best result obtained. The plot shows a more ordered pattern, where actions are grouped tightly to ensure a good thrust. The actions that have a circle around them actually represent the legs that are doing most of the work, since they are against the ground in that step. The firings outside the circles are legs fired in the air, probably to improve the behavior given that they might unbalance the wheel and help the wheel to rotate in the correct direction.

In order to compare the performance of the four algorithms, the average angular velocity of the wheel through a complete episode was taken as the benchmark. Figure 4(a) - left shows the average angular velocity for each episode for the one-step SARSA and Q learning algorithms during the whole learning process (from episode 0). Figure 4(b) - right shows the same results, but for the Q-learning and SARSA algorithms with eligibility traces.

The most relevant aspect of this comparison is that the algorithms augmented with eligibility traces converge much faster than their one-step counterparts. The one-step algorithms take almost 500

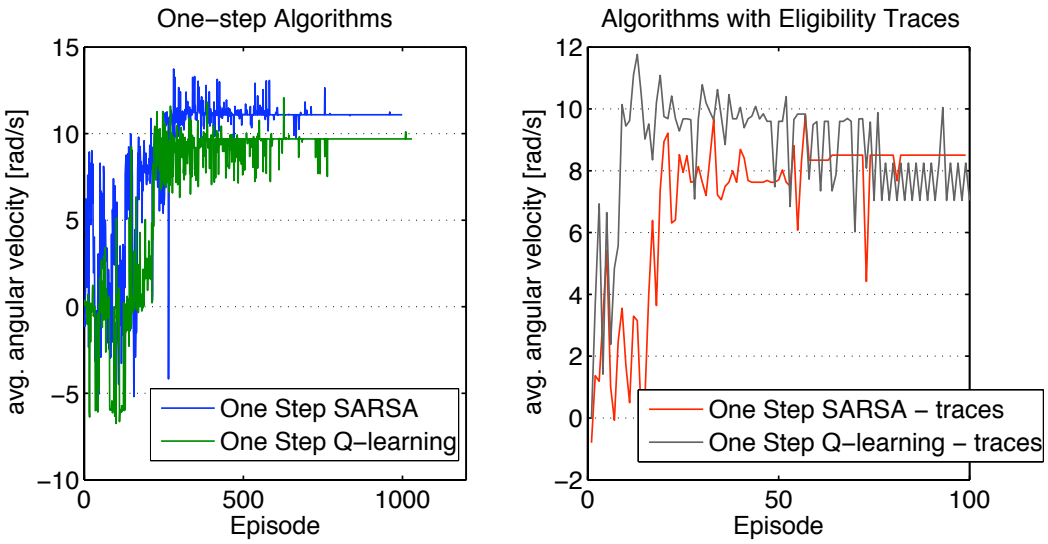

Figure 4: Average angular velocity for each episode for all four algorithms.

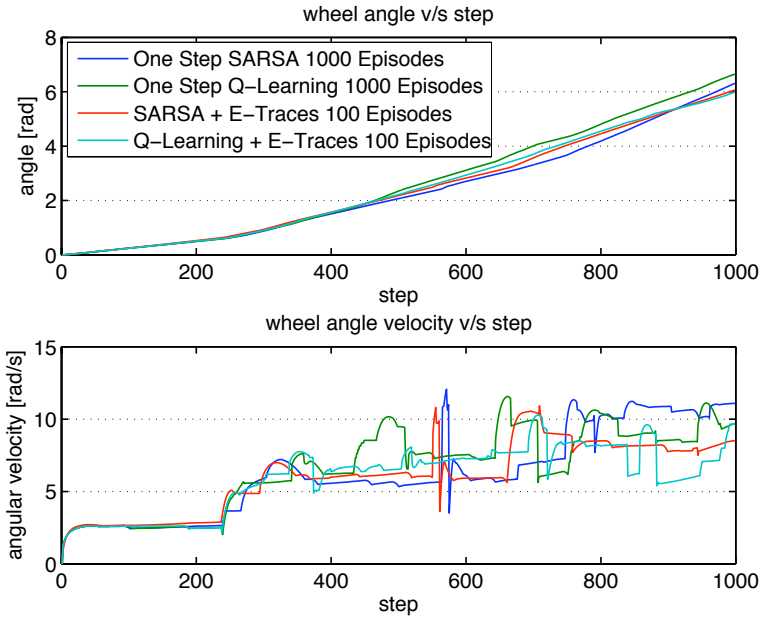

Figure 5: The angle and angular velocity of the wheel using the final learned policy of each of the four algorithms.

episodes to converge, while the ones with eligibility traces have satisfactory results with only 50 episodes (note that the X- axis scales of the plot are different). Although the algorithms with eligibility traces converge faster, they do have some drawbacks - the important one being that their final scores are not as good as the one-step methods. Also, it is important to note that the amount of computation required per episode is much higher than One-step updates.

Figure 5 shows the angle and angular velocity of the wheel using the final learned policy of each of the four methods. Again, the results are pretty similar for all four, however, there is small edge in performance for the one-step algorithms. Also, the two algorithms based on Q-learning converged to better final scores than the SARSA based.

The analysis we have provided above was drawn from our particular results, which may have some bias based on the parameters we chose for each method. However, the conclusion is that all four methods are robust, and prove that Reinforcement Learning is viable solution to control complex systems such as the one presented in this work. The eligibility traces based methods proved to learn very fast, and therefore it is highly probable that we will consider them for testing in the real robot in the near future.

References

[1] R. S. Sutton and A. G. Barto (1998) Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA.

[2] M. Harmon. (1996) Reinforcement learning: a tutorial.

[3] Bradtke, S. J. and Duff, M. O. (1995). Reinforcement learning methods for continuous-time markov decision problems. *In Advances in Neural Information Processing Systems,* MIT Press.

[4] Robotic All-Terrain Surveyor (RATS) videos:
`http://www.frc.ri.cmu.edu/~aluders/videos/RATS_4-23-06.avi`
`http://www.frc.ri.cmu.edu/~aluders/videos/RATS_4-23-06_single_hop.avi`.

[5] L.P. Kaelbling, L.M. Littman and A.W. Moore, Reinforcement learning: a survey, *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[6] T. Thorpe and C. Anderson. Traffic Light Control Using SARSA with Different State Representations. `http://www.cs.colostate.edu/~anderson/res/rl/`.

[7] Neumann Gerhard, The Reinforcemen Learning Toolbox: RL for optimal control Tasks `http://www.igi.tugraz.at/ril-toolbox/general/overview.html`

[8] A.G. Barto and R.S. Sutton and C. Watkins (1990) Learning and Sequential Decision Making, MIT Press, Cambridge, MA,

[9] Videos of RATS using SARSA learning algorithm:
`http://www.frc.ri.cmu.edu/~aluders/videos/RATS_RL_10.avi`
`http://www.frc.ri.cmu.edu/~aluders/videos/RATS_RL_1500.avi`
`http://www.frc.ri.cmu.edu/~aluders/videos/RATS_RL_3000.avi`