A Perception-Based Reward Function Back-Propagation Approach for Optimal Training of Reinforcement Learning Algorithms

1 Abstract

In this paper, I proposed an algorithmic approach integrating human perception to allow for the accelerated training of an intelligent agent, using the Reinforcement Learning Algorithm. Often times, designing a sophisticated reward function needs extensive knowledge of the problem domain or the environment. If the reward function design does not adequately represent the problem, it can lead to lengthy training of the Reinforcement Learning algorithm, and occasionally, an undesirable performance of the intelligent agent. To validate this algorithm, I implemented a variation on a form of the Reinforcement Learning algorithm, Approximate Q-Learning, and considered how one could utilize human perception in order to cater the agent's performance to the user's desire. This algorithm allows for accelerated training and a more accurate performance by utilizing the user's human judgment of the final outcome of a training episode, and recalculating the "learned" weights of the reward function accordingly. The approach proved to be a more optimal technique compared to the standard Approximate Q-Learning algorithm. The proposed algorithm demonstrates its capabilities by its ease in navigating an intelligent agent or robot within an unknown terrain without hitting any obstructions, while successfully reaching its destination. This paper discusses the algorithm's accelerated training capacity and improved accuracy, in comparison to the other well-known algorithm that it builds off of.

2 Introduction

This paper considers studying the training of intelligent goal-driven agents under partially observable and non-deterministic environments. An intelligent agent can be defined as anything that can comprehend its environment through sensors and solve a problem using an external controller, or an actuator. [1] The major attribute of an agent is its ability to learn the complexity of its environment to achieve a predefined goal. [2] The popular training mechanism, supervised learning, bases itself on a large volume of reputable data. But, for partially observable and non-deterministic environments, the Reinforcement Learning Algorithm (hereby referred to as RLA) remains one of the most effective agent training mechanisms, as it can learn from poor or limited information. The RLA operates on a reward function to motivate an agent towards a desired behavior. [3] However, according to the researchers at Max-Planck-Institute for Intelligent Systems, "Defining [reward functions] remains a hard problem in many practical applications… and often leads to undesired emergent behavior." The lack of a rigorous reward function typically prevents an accelerated training of RLA agents. [4]

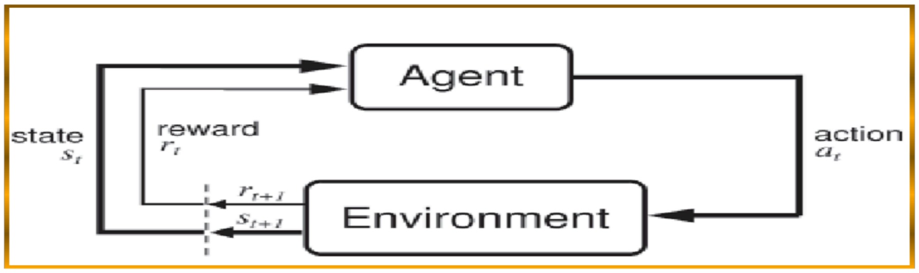

Figure 1– A diagram representing the high level design of the RLA.

The design of a reward function, which depends on the current state, the next state, and current action of an agent, often requires an extensive knowledge of the environment and the task at hand. Unfortunately, the designer of the reward function often does not have that privilege, and is forced to design the function using minimal or heuristic information about the environment, resulting in a lengthy training process of the RLA agent. The basic structure of an RLA agent is shown in Figure 1, where the agent carries out an action and the environment module determines the reward, based on the results of the previous action, and carries out the next state. [2]

This paper analyzes a form of the RLA, Approximate Q-learning, which can undergo optimal action selection for any given Markov Decision Process (hereby referred to as MDP). MDPs are used to solve dynamic optimization problems through the RLA. [1] In an MDP, at each time step, any action available in state ***s*** can be chosen. The MDP responds by randomly moving to a new state ***s'***, and returning a reward ***r***, for the previous action. Approximate Q-learning uses a well-constructed policy and the MDP to determine the optimal action (the action with the highest Q-value) to take during each time step. The environment is represented by a set of feature vectors and weight vectors. A feature vector is a numerical representation of the important properties of the environment, whereas a weight vector numerically represents whether a certain feature has a positive or negative impact on the goal. At each state transition of the agent, the system updates the weights based on the rewards, and recalculates the Q-values accordingly. The value function is the holistic representation of the success of the agent's training session. [5]

In response to the lack of a sufficient reward function, I decided to answer the question of how one can make a rigorous reward function. Since, eventually the performance of the agent implementing the RLA needs to work in accordance to a human evaluator's expectations, I hypothesized that if a framework accepts human judgment of the final outcome of a training episode, then the agent's training time can be accelerated, even while using a rudimentary reward function. In this paper, I propose using back propagation of the evaluator's final score/reward value to recalculate all the "learned" weights, and subsequently, the Q-values, for an agent through the Approximate Q-learning RLA. The above hypothesis cannot be found in existing literature and if proven, can potentially bring a successful human influence on the RLA training. The remainder of this paper discusses the experimental assessment of this hypothesis and its results. In order to ensure the validity of any positive outcome from my experiments for real world applications, I decided to experiment using a real environment instead of any simulated environment so that the partially observable and non-deterministic nature of the environment can be faithfully reproduced.

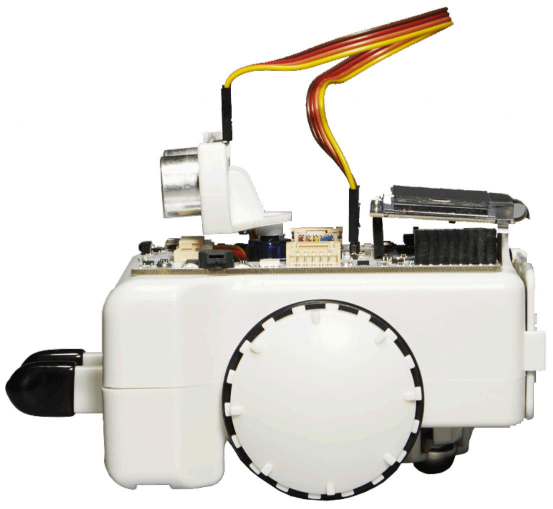

Figure 2– This is Sparki, the intelligent vehicle agent used to test both prototypes– Approximate Q-Learning and Reward Back-Propagation

3 Materials & Methods

(Note: all programming was performed in Python in the PyCharm CE IDE, using the sparki-learning Open Source library. All testing was carried out on a relatively flat surface (i.e– a wooden floor, or a black plastic table) using the Sparki toy robot manufactured by ArcBotics LLC. *The author of this paper has written all other Python classes or methods that do not belong to sparki-learning. Note – Sparki's speed was determined to be 1 cm/second.)*

My hypothesis validation consisted of the following stages:

Preliminary Stage: *Experiment Design*

I began by designing an experiment, which could test the results of the proposed reward back propagation algorithm alongside my hypothesis. In this experiment, the algorithm was implemented to help navigate a single agent in a stochastic environment from an initial destination to a final destination, given limited information about the terrain and a position vector from the point of origin of the agent. For convenience, I decided to use a small toy car, Sparki (as shown in Figure 2), which can be maneuvered through Python code using the sparki-learning library. Sparki also has imprecise ultrasonic sensors, which can give a rudimentary estimate of its distance to an obstacle. The "success" of any trip by Sparki would depend on whether Sparki avoided obstructions as it travelled, and whether Sparki reached relatively close (within 5 inches) to its destination. Additionally, it would be simple to track Sparki's progress during the training, by comparing the final scores (assessed by the same unbiased human evaluator), and determining whether there was any visible improvement in Sparki's training performance when the Reward Back-Propagation algorithm was used.

Stage 1: High-Level Design

I created a high-level design, which streamlined my solution to the problem. The solution is comprised of two major components: the software design components, and the hardware design components as illustrated in Figure 3.

The main software component, or the Navigator, as denoted in Figure 3, functions as the brain of the autonomous vehicular navigation system which leads Sparki to the final destination without hitting any obstructions. The Navigator accomplishes this goal by using the latest state information from Sparki and accordingly issuing a suitable action to Sparki. The Navigator consists of three primary components– the Environment module, and the Policy, and the Vehicle Agent.

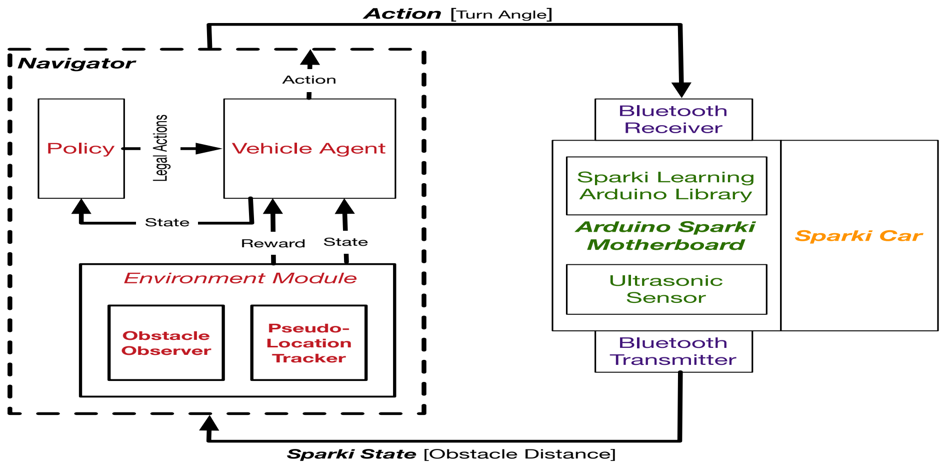

Figure 3– Overall System Design

The Environment module is the software representation of the actual environment within which Sparki is located. This module contains two sub-components– the "Pseudo-Location Tracker" and the "Obstacle Observer". The pseudo-location tracker represents the position at which Sparki is operating within a given environment. This is done through a pseudo-coordinate system, which estimates the current coordinate of Sparki based on the movements made from the original location (which is always assigned to (0,0)) at any given point in time. At each position, the pseudo-location coordinate $(currX, currY)$ of Sparki is calculated using the previously estimated co-ordinate observed for Sparki $(prevPos.x, prevPos.y)$, estimated distance travelled $(distTravelled)$ by Sparki from previous observation, and the vertical distance rotated by Sparki from its initial position (φ), as shown in Equation 1.

$$\boxed{\begin{aligned} currX &= prevPos.x + distTranvelled \times \sin\varphi \\ currY &= prevPos.y + distTravelled \times \cos\varphi \end{aligned}}$$

Equation 1– Pseudo-location coordinate calculation

The second sub-component of the environment module, the obstacle observer, models the distance of obstacles from Sparki. Sparki's distance from its nearest obstruction is periodically checked using its imprecise ultrasonic sensors. The pseudo-location and obstacle distance parameters together represent the current state of Sparki within the Environment module. This estimated "State" keeps track of the following properties: the current position of the vehicle, the final target position, the previous action taken by Sparki, the distance of the nearest obstacle, the vertical angle of deviation of Sparki from original direction (to aid in the pseudo-location tracking system), the score for the current state (to aid in computing the future rewards), and the current trip time (to determine whether the agent is taking more time than required).

The score for the current state is computed using a heuristic algorithm, which calculates the score for the current state using two types of sub-scores: the "Obstacle Distance" score which gives either a positive or negative score based on how far away Sparki is from an obstacle, and a "Trip Delay" score which assigns progressively negative scores if the overall travel time is delayed due to too many detours caused by obstacles. The sum of these two sub-scores is the overall score for the current state, ***s***, as shown in Equation 2.

$$\boxed{stateScore(s) = obstacleDistScore + tripDelayScore}$$

Equation 2 – State score calculation

Using the estimated current state and the previous action of Sparki, the Environment module also computes the "reward" for the previous action. The reward function is the change in scores (calculated through Equation 2) from the current state (*currState*) to the last state (*lastState),* as depicted in Equation 3.

$$\boxed{reward(currState) = stateScore(currState) - stateScore(lastState)}$$

Equation 3 – Reward function

The next component, the Policy module, takes the current state of Sparki and generates a list of "legal" actions for the Vehicle Agent. If no obstacles are found near Sparki, then there will only be one "legal" action: to continue to move Sparki in the same direction, with no directional change. However, if an obstacle is found within the threshold range of 5 cm, then a list of legal actions will be generated with different angles so that Sparki can avoid hitting the obstacle.

The Vehicle Agent uses the reward and Sparki's current state to direct Sparki towards the most optimal path to the destination. The vehicle agent takes the inputs from the "sensations" (pseudo-location tracker and obstacle observer) and the reward function, and gives Sparki the next action to take based on the RLA algorithm.

The hardware components consist of the Arduino Sparki motherboard, the imprecise ultrasonic sensors, the Bluetooth receiver/transmitter, and the Sparki car. The Sparki learning Arduino library (sparki_learning) facilitates Sparki, and can instruct it to move at a certain angle, or collect information about obstructions in the path. The ultrasonic sensors return a numerical distance in centimeters, giving a rudimentary approximation as to how far away the closest obstruction is. The Bluetooth receiver directs Sparki's movements, and the Bluetooth transmitter transmits information about Sparki's current location and nearby obstacles.

In this system design, the software components and the hardware components are in a constant feedback loop. The Navigator's Vehicle Agent uses the policy to determine Sparki's initial actions, which are then received by the Bluetooth receiver. The Bluetooth receiver processes the information, and instructs Sparki to carry out the actions. In the meantime, as Sparki is moving, it is also continually collecting data on obstacle distances and inferring its pseudo-location. The Bluetooth transmitter transmits this data to the environment module. Then, the environment module sends the Vehicle agent a current state-action pair. The Vehicle agent then determines Sparki's next actions, and so on and so forth.

This essential experimental design was used to create two different prototypes for comparing the results from these prototypes.

Stage 2: Prototype Design Using Approximate Q-Learning

For the first prototype, I decided to take a more traditional route to solve the earlier stated problem using approximate Q-learning. The Vehicle Agent in Figure 3 implements the Approximate Q-Learning algorithm in this prototype. The algorithm also allowed for online training, where Sparki learned as it moved.

The Approximate Q-Learning algorithm assigns values to Q-values for any state-action pair, based on inputs from the environment and prior experiences. It extracts the features of the environment for any given state-action pair. Features are functions that maps states to numbers. [5] I found the chosen problem has essentially has two key features, as depicted in Equation 4.

$$\boxed{\begin{aligned} \boldsymbol{f1} &= \frac{\boldsymbol{1}}{\boldsymbol{distance\ to\ destination}} \\ \boldsymbol{f2} &= nearest\ obstacle\ distance\ fron\ Sparki \end{aligned}}$$

Equation 4– Equations representing features for Sparki environment

The algorithm assigns a weight vector to these features. The goal of the Approximate Q-learning algorithm is to "learn" appropriate weights through multiple training episodes. The weight vector is initially initialized to [10,1]. For each feature within a state-action pair, Equation 5 is used to update the weight based on the previous Q-values. [5]

Weight update equation:

transition = (s, a, r, s'), where s = state, a = action, r = reward, s'= next state

w(i) = weight for i-th feature

f(i) = value of i-th feature for state-action pair (s, a)

$difference = \left(r + (\gamma \times maxQ)\right) - Q(s,a)$

$w(i) \leftarrow w(i) + (\propto \times difference \times f(i))$

Q-value update equation for state-action pair (s, a):

$Q(s,a) \leftarrow (1-\propto) \times Q(s,a) + \propto \times (r + (\gamma \times maxQ))$

α = Learning rate = 0.75; r = Reward value; γ = Discount Factor = 0.8

Equation 5– Approximate Q-Learning update equation

Stage 3: Prototype Design Using The Reward Back Propagation Algorithm

For this prototype, I chose to redefine the reward function to be more human perception-based for accelerated training of Sparki. This model was implemented by creating an additional step after Sparki reaches its destination, where the program asks for human feedback on how well Sparki performed in the current training session. In this step, and during all of Sparki's previous training sessions, the same unbiased human evaluator is expected to assign a numerical reward score within [-1000,1000] range for the whole trip. The system then uses this human feedback for the training episode and adjusts their reward value in relation to the other intermediate rewards for all of the state-action pairs that were computed in the current training episode. Figure 4 describes the algorithm for the Reward Back-Propagation Algorithm.

```
i.    Take final reward finalRewardVal from human judge
ii.   judgedRewardScale = 2000 [1000-(-1000) = 2000]
iii.  Find range of actual rewards value rewardsRange for the training episode
iv.   rewardsScaleFactor = rewardsRange/judgedRewardScale
v.    scaledRewardsVal = finalRewardVal/rewardsScaleFactor
vi.   Access the actualRewardsMap containing actual rewards by key (state, action,
      nextState) that was encountered within the training episode
vii.  for each (state, action, nextState) key within the actualRewardsMap:
              actualRewardVal = actualRewardsMap[((state, action,nextState)]
              adjustedRewardVal = actualRewardVal + scaledRewardsVal
              call update(state,action,nextState,adjustedRewardVal) as per Equation 5 for
              updating weights and Q-values
```

Figure 4– Reward Back-Propagation Algorithm

4 Results and Analysis

In order to systematically compare the Approximate Q-Learning (which acts as the control in this situation) and the Reward Back-Propagation algorithms, I devised a uniform, unbiased approach to score the results from both prototypes. This scoring mechanism took Sparki's final distance from the destination and the severity of its contact with obstacles into consideration in accordance to what I expected of Sparki.

Score Range	**Assessment of Training episode**
-1000 to -751	Sparki did not reach anywhere near the destination, or did not go towards the destination
-750 to -501	Sparki directly hit an obstacle on its way to the destination
-500 to -251	Sparki reached near the destination, but came

	into direct contact with an obstacle along its route
-250 to -1	Sparki ended far away from destination point, but didn't hit any obstacles
0 to 250	Sparki reached the destination within 3-4 inches of the destination, without hitting any obstacles
251 to 500	Sparki reached the destination within 2-3 inches of the destination, without hitting any obstacles
501 to 750	Sparki reached the destination within 1-2 inches of the destination, without hitting any obstacles
751 to 1000	Sparki reached the destination within 0-1 inches of the destination, without hitting any obstacles

Table 1– The User's Scoring Criteria of Sparki's Performance

Using the scoring criteria, I used the following steps to test each prototype:

1) Set up a simplistic training environment on a 3' 9.5" by 1' 10.5" table. The training environment consisted of:
    a. A single 4" tall obstacle
    b. A fixed target position vector marked by a quarter
2) Run a training session. Repeat three times.
    a. Run a training episode. Repeat fifteen times.
        i. After each episode, record a score based on the scoring criteria in Table 1, regardless of whether the algorithm requires a human influence.
        ii. After the score was recorded, the 4" obstacle was randomly moved, but the final position vector remained the same.
    b. Clear the "learned" weights from the session.

3) Find and record the average score for each training episode over each of the three training sessions.

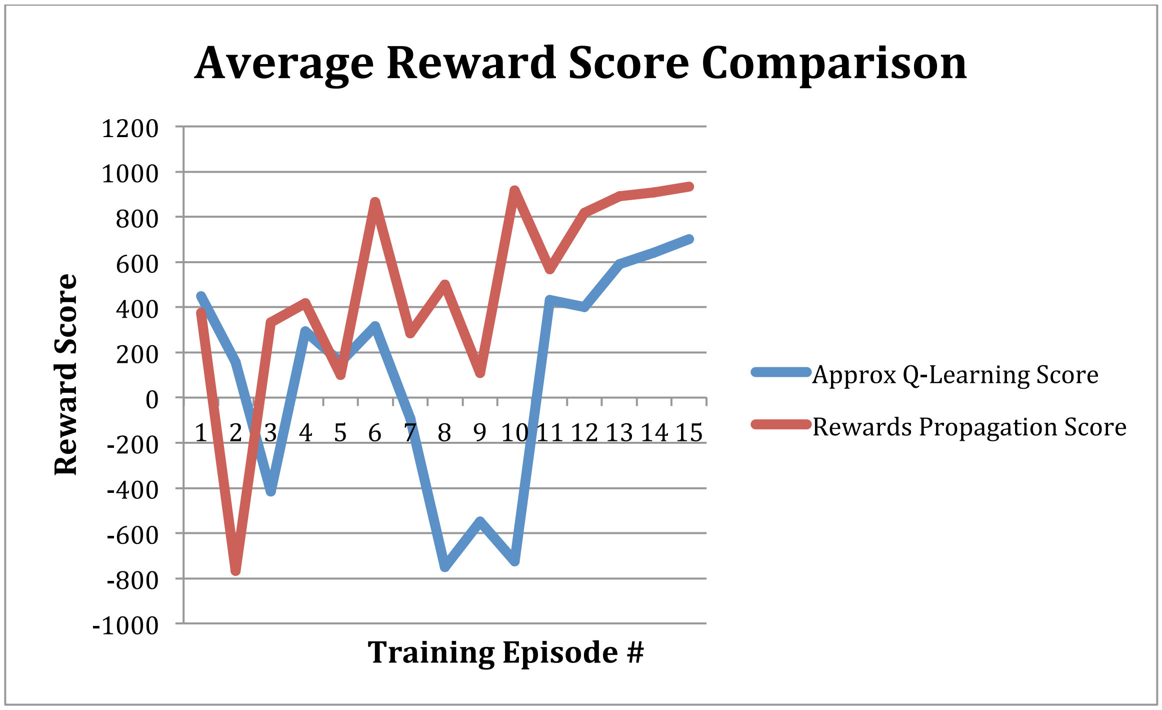

Figure 5– Average score from each training episode, over three training sessions

As the graph in Figure 5 depicts, the Reward Back-Propagation algorithm stabilizes much faster on average than the Approximate Q-Learning algorithm. Additionally, the Reward Back-Propagation algorithm generally provides higher (about 20% better) scores than the Approximate Q-learning algorithm by the end of a training session. The average score from all three training sessions (including all forty-five episodes) using the Rewards Back-Propagation algorithm is much higher than that of the Approximate Q-Learning algorithm (483.0 vs. 103.0).

After the training, I devised another test scenario to find out how well the "trained" algorithms from each prototype can work on an unknown terrain. The following procedure was carried out to test the "trained" prototype's capability on an unknown environment:

1) Set up a training environment on a 3' 9.5" by 1' 10.5" table. This environment contains:

    a. Two 4" tall obstacles

b. A new target position vector

2) Run a training episode. Repeat ten times.

a. The 4" obstacles were moved around randomly during Sparki's training episode, creating a dynamic environment.

b. After each training episode, a score was recorded in a table, in accordance to the scoring criteria in Table 1.

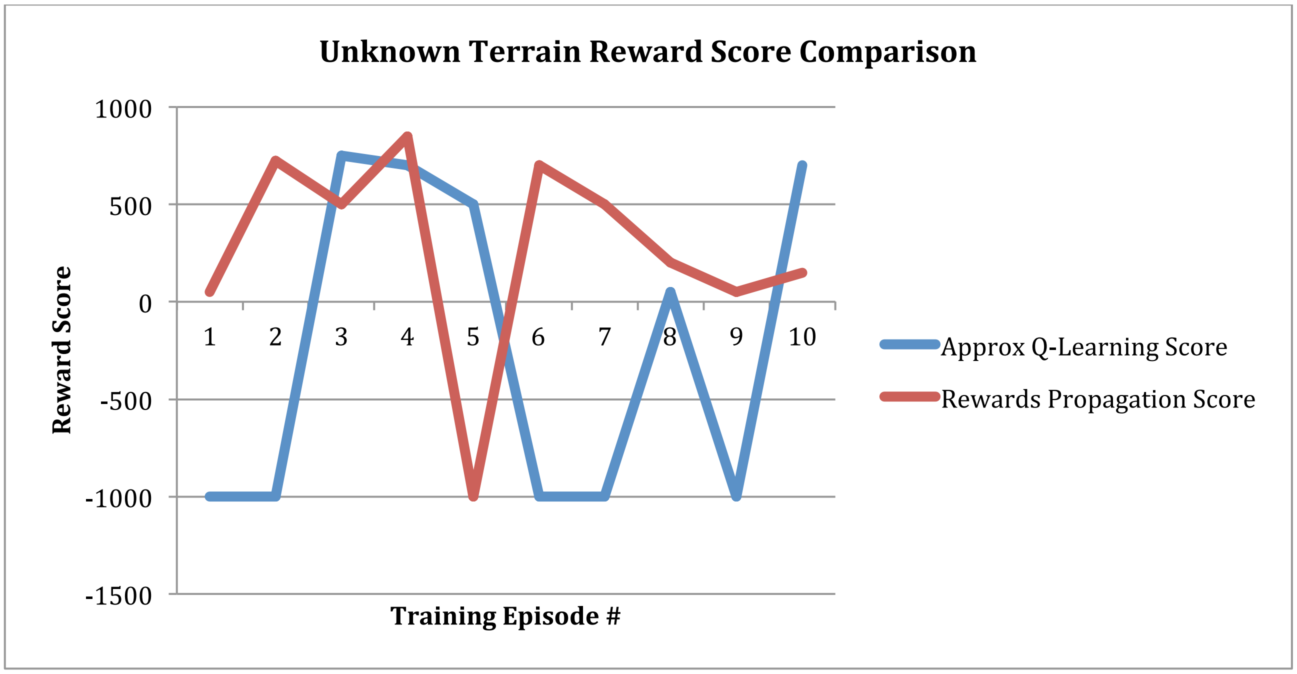

Figure 6– Results from experiment using unknown terrain

As the graph in Figure 6 indicates, the "trained" Reward Back-Propagation algorithm was generally much better at functioning in an unknown terrain, compared to the Approximate Q-Learning algorithm. The average training episode score for Approximate Q-Learning, -230.0, was 502.5 points less than that of the Reward Back-Propagation algorithm. The Reward Back-Propagation score only dipped down during the fifth training episode, but was otherwise steadily positive. In comparison, the Approximate Q-Learning score fluctuated several times through the course of the ten training episodes. However, both algorithms didn't perform as well as they had in "learned" their training terrain.

The Rewards Back-Propagation algorithm is a generic algorithm that can be applied to any RLA and not specific to the Approximate Q-Learning algorithm. However, this algorithm has a much bigger impact on specific types of problems. In the problem chosen for these prototypes, it was not possible for Sparki to know whether it definitely reached the destination accurately, as the environment was not capable to provide this feedback automatically. Additionally, the absence of a GPS and the partially observable nature of the environment made this mathematically complex. Plus, the estimates for position, orientation and velocity of Sparki required to determine its state were noisy, or imprecise. Hence, for such types of problems RLA couldn't find the optimized policy without any external feedback. For the types of problems that I prototyped, I found that the Reward Back-Propagation algorithm clearly establishes a model that not only accelerated training, but also allowed for a more favorable performance.

Moreover, the rewards function I used in my prototypes was based on a heuristic-based, rough understanding of Sparki's environment and was not necessarily the most appropriate for the problem. Also, I didn't spend much time establishing a mathematically optimal reward function either. Even with these handicaps on part of the robustness of reward function, my experiments established that reasonably accurate RLA based robotic system could be created without significant training overhead.

5 Conclusions

This research successfully found an effective algorithmic approach for the accelerated training of Reinforcement Learning algorithm through a rudimentary reward function. I developed a back-propagation algorithmic approach, which builds off of the Approximate Q-

Learning RLA by incorporating a human evaluator's perceived score of the agent's performance to update the actual reward function's score and consequently recalculate the Approximate Q-Learning weights. After the agent undergoes a training session, the user/human evaluator is prompted to give the agent a perceived score of its performance. The "learned" weights are recalculated in accordance, and subsequently, the Q-values are changed.

This algorithmic approach was tested using an autonomous toy car, Sparki, so that the evaluator could easily assess the agent's performance in a realistic partially observable and non-deterministic environment. As a result of this approach, the agent did not require a substantial amount of training and had consistently better performances, in comparison to the Approximate Q-Learning RLA control. I also found that the performance of my proposed algorithm is much better than the Approximate Q-Learning algorithm, when used in a generic terrain after training. And, I found this to be true, even though the estimates for Sparki's position, orientation, or speed are, at best, very noisy. Unfortunately, the agent's performance through both algorithms could have been much better, as Sparki often grazed past the obstructions, or in other words couldn't "see" obstructions unless they were directly in front of Sparki. This is due to the fact that Sparki did not have enough knowledge of its own width, and could not "view" the obstacles that it grazed, because of the small angular range of its ultrasonic sensors.

The testing of the proposed algorithm was limited to one specific vehicular agent, Sparki at this time. However, this algorithm could be applied to train other types of agents, through extension of the research in the following direction.

As a first step in extending the capability of the algorithm, I plan to make a more robust policy, where the agent knows some basic information about itself (i.e. its width). Alternatively, I would also make the "Path Planning" component of the algorithm, which is responsible for detouring Sparki after it encounters an obstacle in a close distance, more robust so that “obstacle-grazing” problems can be solved algorithmically. I would also test this algorithm on several other situations, to allow this approach to solve any problems requiring a long period of training. One situation where this algorithm could be used is to modify/validate forecasting data. Modern forecasting algorithms are usually very mathematically robust. However, it is almost impossible to encode all the knowledge of the external environment’s impacts within the forecast algorithm, as the possibilities are infinite. If forecasting algorithms are used within a system that implements the Reinforcement Learning Algorithm, then the approach proposed in this paper could be used to take feedback from a human assessor on the algorithm’s performance, and effectively fine-tune the accuracy of the predictive mechanism in future forecasts under similar conditions.

Additionally, this proposed approach could be used to create obstacle-avoidance path algorithms for delivery/pickup drones in manufacturing companies. These kinds of drones would have the knowledge of a path to the final destination, and would be likely to face unknown obstacles along the way. This approach would allow these delivery/pickup drones to quickly grasp the concepts of obstacle-avoidance during their training, thereby reducing the large amount of time and hefty cost it takes to train these machines.

Once the Reward Back-Propagation algorithm is enhanced to be more rigorous for handling non-deterministic, stochastic environments, it can have a wide variety of applications in the real world.

6 References

[1] Russell, Stuart, and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd ed., Prentice Hall, 2009.

[2] Sutton, Richard S., and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, 1998.

[3] Dandurand, Frédéric, et al. *Complex Problem Solving with Reinforcement Learning*. IEEE, 2007, www.psych.mcgill.ca/perpg/fac/shultz/personal/Recent_Publications_files/reinforcement07.pdf. Accessed 20 Sept. 2016.

[4] Daniel, Christian, et al. *Active Reward Learning*. www.roboticsproceedings.org/rss10/p31.pdf. Accessed 20 Sept. 2016.

[5] Klein, Dan, and Peter Abbeel. "CS 188: Artificial Intelligence." *EECS Instructional and Electronics Support*, University of California, Berkeley, 2011, inst.eecs.berkeley.edu/~cs188/fa11/lectures.html. Accessed 20 Sept. 2016.