CSCI 446 Artificial Intelligence Project 4 Final Report

Roy Smart Nevin Leh Brian Marsh

December 14, 2016

1 Introduction

2 The Racetrack Problem

3 Value Iteration

3.1 Description

3.2 Implementation

3.3 Experimental Design

4 Q-Learning

4.1 Description

Q-learning is a model-free reinforcement learning method for determining optimal action-selection policies [Russell and Norvig, 2010]. An agent using this method leverages a quantity known as the Q-value to derive the optimal action a for each state s. The Q-value, $Q(s, a)$ describes the expected utility for every action in every state within the environment and is learned by the agent using temporal difference learning. An expression to calculate the Q-value is given by [Russell and Norvig, 2010] as

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{1}$$

where a is the action that was executed in state s that resulted in state s' and $R(s)$ is the reward function. The constants α and γ are known respectively as the learning rate and the discount factor. Equation 1 is used as an update rule to adjust the value of $Q(s, a)$ for each action-state pair in every time trial undertaken by the agent. Using this simple update rule and Q-values initialized to zero for every action-state, an agent can learn how to navigate an environment.

4.2 Implementation

We based the design of our algorithm on the function Q-Learning-Agent described in Figure 21.8 of [Russell and Norvig, 2010]. The table of action values, $Q[s, a]$ and the tabel of frequencies, $N[s, a]$ were stored in six-dimensional arrays representing every possible position, velocity and acceleration in both spatial dimensions.

The agent took an astonishingly long time to train. To lower the training time, we adopted strategy where the agent was started close to the finish line and trained until convergence. The starting line was then moved back by m squares and then again trained until convergence. This process was repeated until the

Parameter	Value
Base Reward	-1.0
Wall Reward	-2.0
Finish Reward	1.0
Learning Rate	1×10^{-7}
Discount Factor	0.9
m	2

Table 1: *Table of tunable values for the Q-Learning agent*

actual start line had been reached. In this way, our Q-Learning agent could be incrementally trained, without having to wait around for a few months to complete training.

To further improve training, we assigned a penalty to hitting the wall. We understand that this breaks the rules of the assignment slightly, but the training time was too preventative. This behavior could indicate a problem with our Q-Learning algorithm.

The traditional way to detect convergence in Q-Learning is to measure the rate of change of the Q-values across the table, and exit once the rate is sufficiently small. We did not adopt this approach as it would have been computationally expensive. We instead opted for a convergence test based off of a running average of the timesteps required for n trials.

The learning rate, α and the discount factor, γ were tuned by hand. The tuned parameters are outlined in Table 1;

4.3 Experimental Design

For our experiments, we measured the average number of time steps for the Q-Learning agent vs. the number of time trials. This has the effect of showing the rate of convergence of the Q-Learning algorithm and describing the overall performance of the algorithm as track complexity increases.

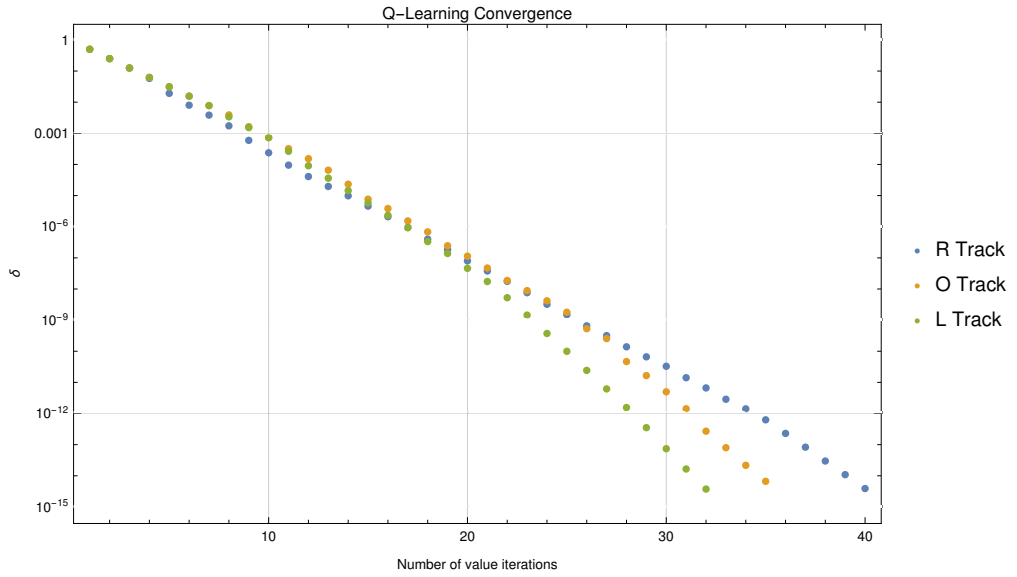

Figure 1

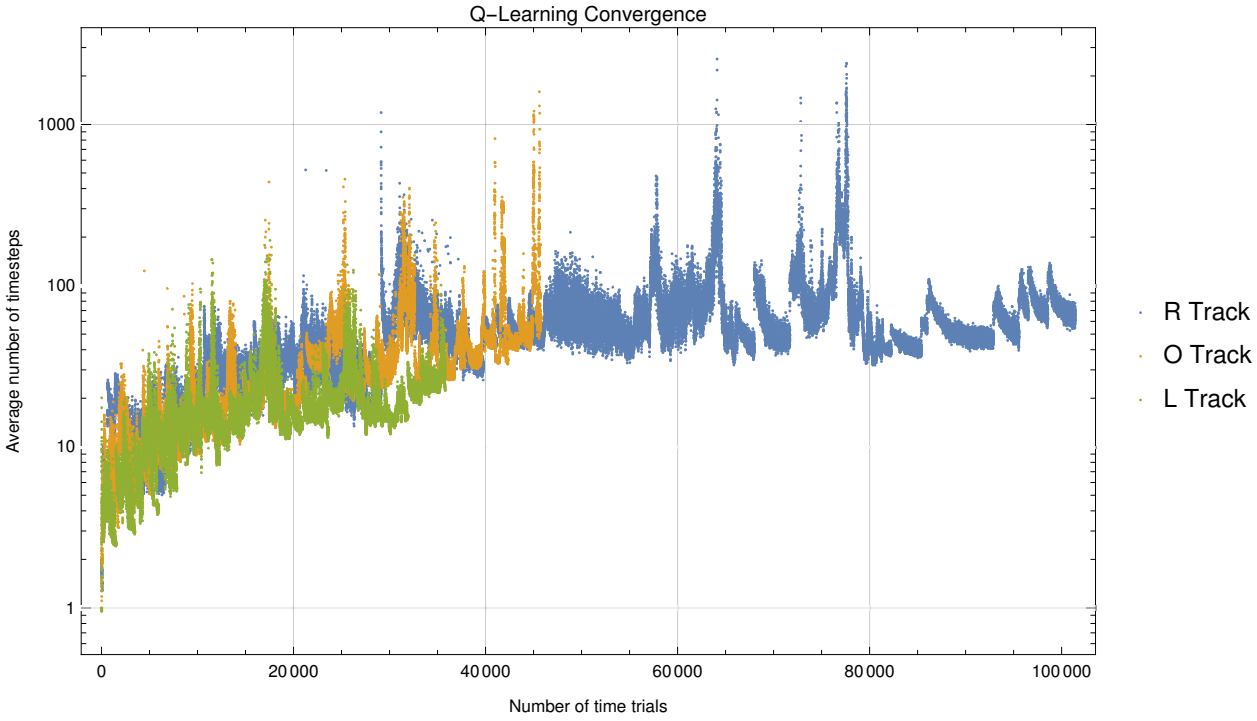

Figure 2

5 Results

5.1 L-Track

5.1.1 Without Restart

5.2 O-Track

5.2.1 Without Restart

5.3 R-Track

5.3.1 Without Restart