Car Simulation Using Reinforcement Learning

Zhijin Wang*
Department of Computer Science
University of British Colombia
zhijin@cs.ubc.ca

Abstract

This project report presents the result of Reinforcement Learning (RL) experiments in a car simulation. Without any knowledge of the tracks in advance, the car can be trained to avoid bumping into the walls by learning from the given rewards. We have built a car simulation system in which the car can be trained and tested on the tracks with several RL algorithms, including Actor-Critic method, SARSA(0) and SARSA(λ). We have also compared the results and given some ideas of future work.

1 Introduction

Our project is motivated by the online Robot Auto Racing Simulation (RARS) [1]. It consists of a simulation of the physics of cars racing on a track, a graphic display of the race, and a separate control program (robot "driver") for each car. Programmers can create their own drivers by writing a module that takes a situation (state) as input and returns the desired steering angle and throttle. The existing robot drivers of RARS are provided with the information of the whole track which can be used to pre-compute the optimal path before the race. When the race starts, these drivers will stick to the pre-computed fixed trajectory, which often obtain excellent finishing time on empty tracks. However, their performance on passing other cars and running on a random track is quite poor due to the constancy of the output of their algorithms.

Our car simulator assumes that the drivers do not know the track information beforehand. At each time step, the driver must observe his current position and then takes appropriate action to avoid bumping into the wall by learning from the past experience and the given rewards. The learning ability, i.e. the convergent rate of the algorithm, becomes the most important critic of different Reinforcement Learning methods.

In the next section, a general background of RL will be given. Then we describe our models and simulation system in section 4, following by the implementation of RL algorithms in section 5. The experiment results are shown in section 6. The conclusions and future work is given in the last section.

*This project is a collaborated work by Zhijin Wang and Chen Yang. Wang designed and built the car and track models, set up the system framework and created the user interface, while Yang implemented the reinforcement learning algorithms.

2 Reinforcement Learning

2.1 General Model

Reinforcement Learning is an approach for an autonomous agent learning to choose optimal actions to achieve its goals by senescing and acting in its environment [2]. It's a simulation of the learning process in the nature. For example, when children do something right, the parents give positive approvals. Then the children tend to repeat the right thing in the same situation while they will avoid doing something if they have been criticized for doing that. The central idea of RL is to find the link between the cause and effect.

Figure 1 shows the learning process of the agent. At each time step, the agent receives a reinforcement feedback from the environment along with the current state. The goal for the agent is to create an optimal action selection policy π to maximize the reward. In many cases, not only the immediate reward but also the subsequent rewards – delayed rewards ? - should be considered when actions are taken.

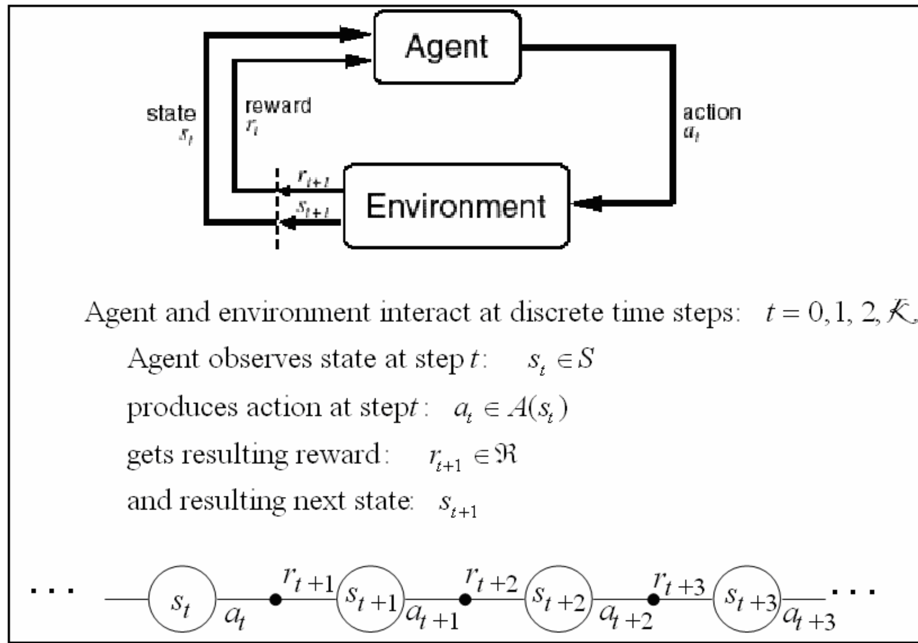

Figure 1: A general model of RL [3]

For every action selection policy π, a value function is estimated by mapping the inputs of the current state to the expected long-term reward that can be earned from the state.

$$V^{\mathbf{p}}(s) = E_{\mathbf{p}}\{R_t | s_t = s\} = E_{\mathbf{p}}\left\{\sum_{k=0}^{\infty} \mathbf{g}^k r_{t+k+1} | s_t = s\right\} \quad (1)$$

In each time step, we use the value function from next state to estimate the value of current state. In this way, the reinforcement learning algorithm learns by iteratively reducing the discrepancy between value function estimates for adjacent states.

$$V(s_t) \leftarrow V(s_t) + \mathbf{a}\left[r_{t+1} + \mathbf{g}V(s_{t+1}) - V(s_t)\right] \quad (2)$$

$$R_t^{(1)} = r_{t+1} + \mathbf{g}V(s_{t+1}) \quad (3)$$

2.2 Temporal Difference (λ)

In Equation 3, the reward is estimated in one step. Another more common approach is to use n-step reward estimation:

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}) \tag{4}$$

TD (λ) algorithm is such an example. It learns by reducing discrepancies between estimates made by the agent at different times. The estimate averages all the n-step estimates, each of which is weighted proportional to λ^{n-1}, where $0 \leq \lambda \leq 1$.

$$R_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)} \tag{5}$$

2.3 Components of Reinforcement Learning

A Reinforcement Learning system often consists of the following four parts [3]:

Environment model gives the feedback to the agent from a simulated environment. For example, given the current state and action, the model will give back the next state and next reward to the agent.

Policy is a mapping from states to actions to be taken. The policy determines the agent's behavior at given time. The agent finds the optimal policy by estimating value function.

Reward function is a mapping from states (or state-action pairs) to a single number as a reward presenting the intrinsic desirability of the state. It's an immediate reward and must be fixed. They are basically given directly by the environment.

Value function specifies what is good in the long run. Without rewards there could be no values, and the only purpose of estimating values is to achieve more reward. The values must be estimated and re-estimated from the sequences of observations an agent makes over its entire lifetime. An efficient method of estimating value functions is the most important component of RL algorithms.

3 Car Simulation System

3.1 Environment Model

Our car model is simplified as a particle on the track plane, so we don't have to consider the size of the car. The state of the car is represented by two variables: the distance of the car to the left wall of the track and the car's velocity to the right wall. Here we assume that the value of the car's velocity is a constant, only the direction is changing all the time. At each time step, the car driver only needs to know the distance from the car to the left wall of the track Dis2L, and the current moving direction α.

As seen in Figure 2, each track consists of a number of segments. All the segments can be divided into 3 categories: straight line, left-turn arc and right-turn arc, distinguished by their values of radius. For example, the right-turn arc has a radius which is defined to be less than zero, and its information is given in terms of the starting point of the left side – (L_X0, L_Y0), the angle between the starting direction and the positive x axis – ?, the center of the circle – (X0, Y0), the radian of the arc – Rad, and the width of the track – Width.

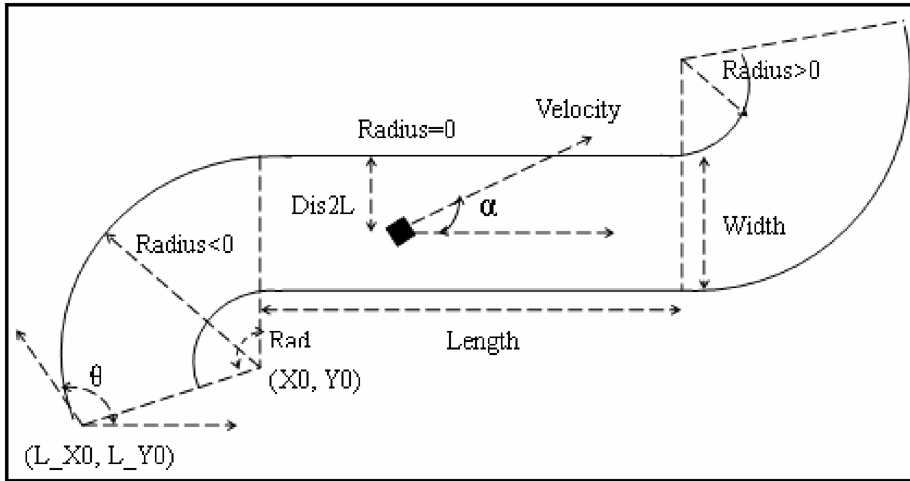

Figure 2: The car and the track model of our system

In order to provide the above information to the car driver, we must first find out which segment of the track is the car in. In other words, we want to know at any time step whether the car has left the current segment and entered the next one. Here we give our different equations to solve this problem, with respect to different types of track segments.

The left part of Figure 3 is a zoom-in of the straight line segment. At each time step, we first compute the relative position (dx, dy) between the car and the starting point of the left side of the track. Then we can use the equations (6) and (7) to find out the distance (D) from the car to the starting line of the segment, as well as the distance (Dis2L) to the left side of the track.

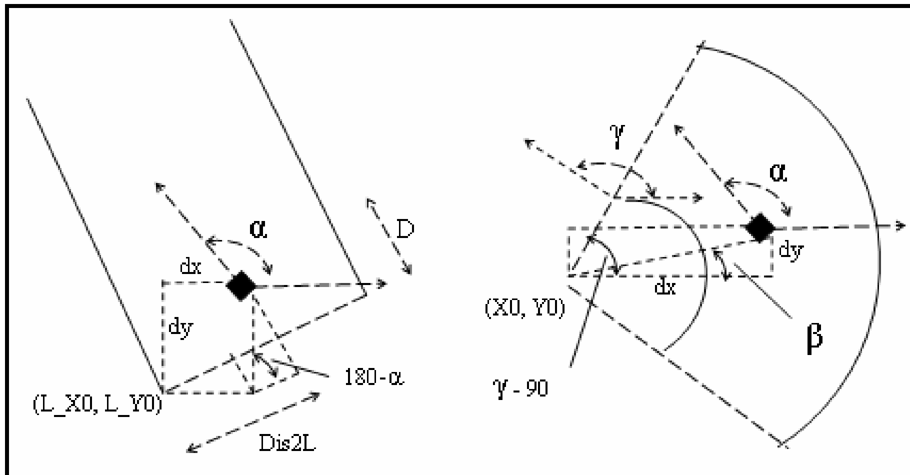

Figure 3: Two types of track segments

$$D = dy \cdot \sin(180\text{-}a) - dx \cdot \cos(180\text{-}a) = dx \cdot \cos(a) + dy \cdot \sin(a) \qquad (6)$$

$$Dis2L = dy \cdot \cos(180\text{-}a) + dx \cdot \sin(180\text{-}a) = dx \cdot \sin(a) - dy \cdot \cos(a) \qquad (7)$$

By comparing the distance D with the straight line segment's length, we can easily tell whether the car is still inside the straight line segment.

For the left-turn arc segment at the right of Figure 3, we also first find out the relative position (dx, dy) between the car and the center of the arc circle. Then we use the

angle ß to compare with the ending angle of the segment, ?. If ß > ?– 90, then the car has left the current segment. The equations are as follows:

$$ß = \arctan (dy / dx) \quad (8)$$

$$\text{Dis2L} = \sqrt{dx^2 + dy^2} - Radius \quad (9)$$

3.2 Reward Function

There are two types of situations where the car should be given a negative reward. One is the car bumping into the wall and the other one is the car going backwards. When any of these happens, the car will receive a (-1) reward and then restarts from the starting line of the track.

3.3 Value Function

The value function is to learn the long run reward to state-action pair of each time step given the immediate reward above. Then the optimal policy can be constructed based on this value function. Here we use a decision tree to find out the current state and the corresponding action to take, as in Figure 4.

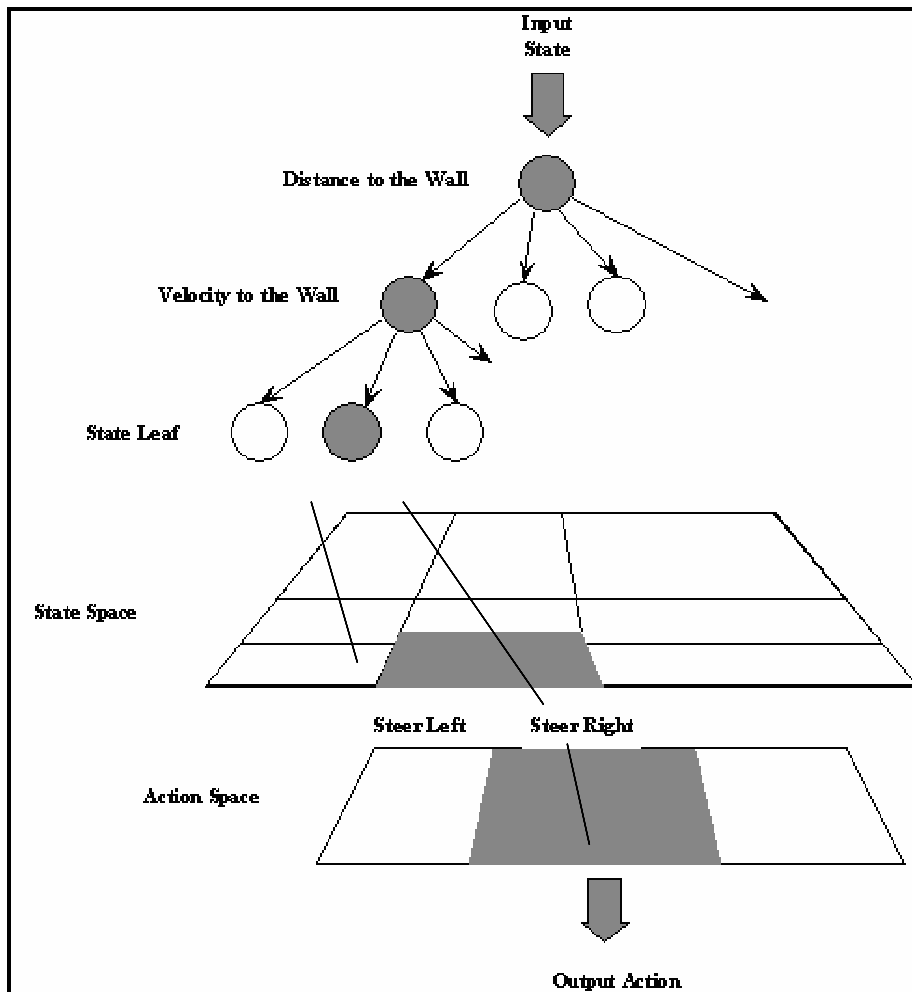

Figure 4: A decision tree to find the corresponding action for each state [2]

The state space and action space is both divided into several discrete boxes. First the state of the car in the track is given by the simulator and the position of the state in the state space is found by searching down the decision tree, then an action such as steering right is given to the car according to the correspondence between the state space and action spaces. This correspondence is given by the policy.

3.4 Policy

In our system, the car's throttle is set as a constant, so we only need to give the steering command as the action of the car at each time step. These actions are based on the value function of perceived states to avoid bumping into the wall or going backwards in a long run.

4 Reinforcement Learning Algorithms

In this project we have implemented three kinds of reinforcement learning algorithms. The basic ideas will be described in this section and we will give the comparison result in section 5.

4.1 Actor-Critic Algorithm

Actor-Critic Algorithm has a separate memory structure (actor) to represent the policy. The estimated value function is the critic, because it criticizes the actions by the actor. The pole balancing program is such a good example. We use the binary output of the pole cart controller to represent the actions of steering left and steering right. Also the reward function is modified to give negative rewards to the car's crashing and moving backwards.

4.2 One-Step TD

SARSA (0) is a one-step TD algorithm we have implemented in our system. The algorithm can be seen in Figure 5

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
    Initialize s
    Choose a from s using policy derived from Q (e.g., ϵ-greedy)
    Repeat (for each step of episode):
        Take action a, observe r, s'
        Choose a' from s' using policy derived from Q (e.g., ϵ-greedy)
        $Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma Q(s',a') - Q(s,a)\right]$
        $s \leftarrow s'$; $a \leftarrow a'$;
    until s is terminal

Figure 5: Algorithm 1- SARSA (0): value function of state-action pair

4.3 TD (*l*)

Since TD (0) only looks one step into the future, the frontier of nonzero Q values will creep backward from the goal state at the rate of one new state-action transition per episode. The convergence rate is slow even for a mid-size problem, so we decided to implement TD (λ) to see whether the converging rate will be faster.

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all s, a
Repeat (for each episode):
  Initialize s, a
  Repeat (for each step of episode)
    Take action a, observe r, s'
    Choose a' from s' using policy derived from Q (e.g. ε - greedy)
    $\delta \leftarrow r + \gamma Q(s', a') - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all s, a :
      $Q(s,a) \leftarrow Q(s,a) + \alpha \delta e(s,a)$
      $e(s,a) \leftarrow \gamma \lambda e(s,a)$
    $s \leftarrow s'; a \leftarrow a'$
  Until s is terminal

Figure 6: Algorithm 1- SARSA (λ): value function of state-action pair

5 Experiment Results

5.1 Parameters of experiments and convergence rate

We set time step τ=0.05, the typical parameters and some statistics over >20 runs are given in Table 1. α is the learning rate, ? is the discount factor for delayed reward, λ is the decay rate for eligibility traces, and ε is the probability for ε-greedy. For Actor-Critic, there are two sets of parameters for actor and critic. The convergence rate is given as the failures before one successful run over a complete lap of the track.

. Table 1: Parameters and comparisons of three RL algorithms

ALGORITHMS	PARAMETERS	CONVERGENCE RATE
SARSA(0)	α=1.0 ?=0.9 ε=0.05	>30
Actor-Critic	α=1000,0.5 ?=0.95 λ=0.9,0.8	5-15
SARSA(λ)	α=0.05 ?=0.9 λ=0.9 ε=0.05	5-15

5.2 Figures

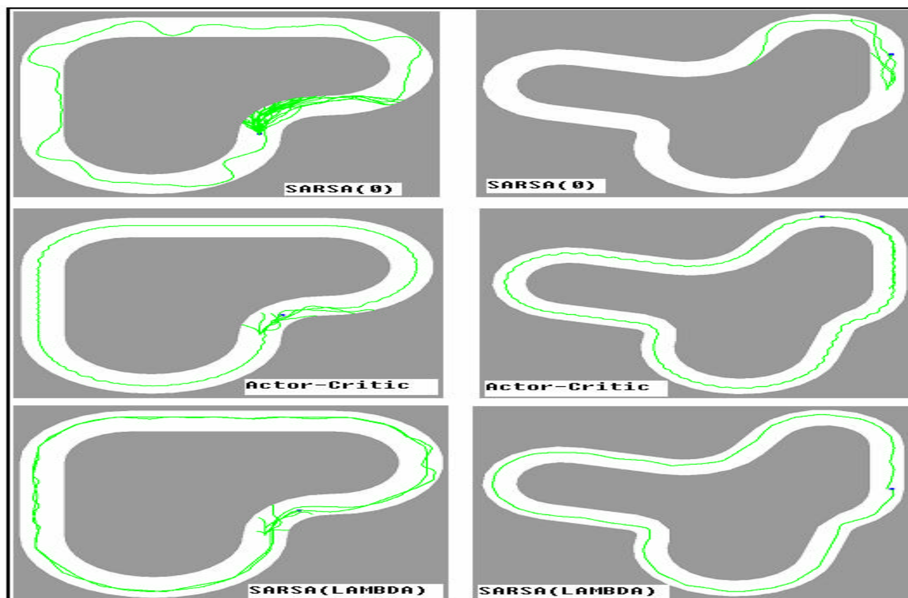

Figure 7: Train and test images captured from our simulator, the left column is for training, and the result of the right based on the estimated value function from the training track, both tracks are unknown to the car controller beforehand. It costs too much time for sarsa(0) to converge

5.3 Comparisons of Q-function

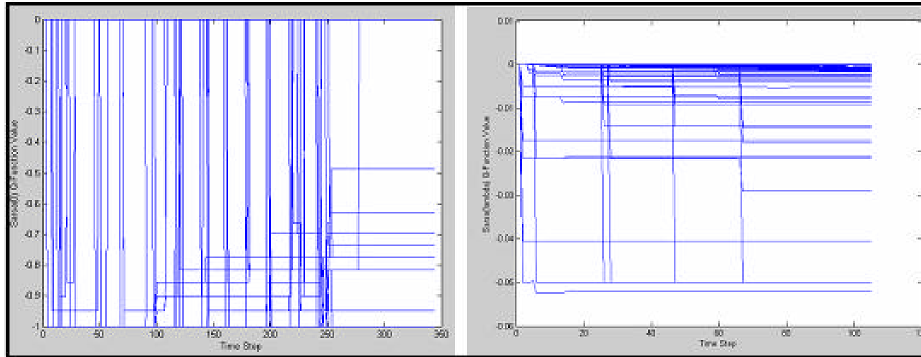

Figure 8: Estimated Q-function values: the left is SARSA(0), the right is SARSA(λ), the x-axis is time and y-axis is the estimated Q-function value, each line corresponds to Q-function value of one state-action pair. We can see the Q-values converges in SARSA(λ) at a rate much faster than SARSA(0)

6 Conclusions and Future Work

Our work has proved that Reinforcement Learning can be successfully applied to the car simulation. The car can learn how to avoid bumping into the walls and going backwards using only local information instead of knowing the whole track in advance. Such robot driver is similar to a human driver and it can work on an unknown track. From the experiment results we also proved that TD(λ) can converge at a faster rate than one-step TD methods.

There is much future work to do in this project. First, we can add additional control to the throttle of the car, so it can finish the lap as soon as possible. We have also tried to find the shortest way with the input of curvatures of the two segments ahead using a negative reward for each step before the end point. But it is very unstable to find the shortest way without bumping into walls. More complex approximated function such as neural network can also be used to improve the performance.

References

[1] Robot Auto Racing Simulator, http://rars.sourceforge.net/

[2] Mitchell, T. M. (1995) *The Book of Machine Learning* : McGRAW-HILL INTERNATIONAL EDITIONS.

[3] Sutton, R. S. & Barto, A. G. (1998) *An Online Book of Reinforcement Learning*, http://www-anw.cs.umass.edu/~rich/book/the-book.html