Reinforcement learning for bicycle control

Bruce Cam, Chris Dembia, Johnny Israeli

December 13, 2013

1 Introduction

In 1998, Randlov and Alstrom showed that reinforcement learning can be used to control a bicycle [7]. Since then, their work has enjoyed minor popularity as a benchmark problem [4, 3, 1, 6], in part because it is harder to solve than the popular cart-pole problem. We implemented Randlov's original work using the PyBrain [9] machine learning library. We were particularly interested in how difficult it is for a human to balance the bicycle model using keyboard inputs; how much of a feat is the reinforcement learning algorithm accomplishing? To this end, we created a small game using the Panda3D library.

As Randlov and Alstrom did, we also attempted to learn a controller that could drive to a goal destination. To extend their work on shaping, we tried our own suite of complex reward functions that would work well for arbitrary goal destinations.

Our code is available online at https://github.com/chrisdembia/agent-bicycle.

1.1 Background and previous work

Since the state of a bicycle is continuous, Randlov and Alstrom needed a method to *generalize* the learner's observations from specific states to new states the learner had not seen. They chose to discretize the five-dimensional state of their bicycle into 3456 bins; the learning that occurs for any state in a given bin is used for all other states within that bin (to evaluate the action value Q). This generalization method is called *linear function approximation with tile coding* [12], though Randlov and Alstrom use different words in their description. This rudimentary method would require a significant tuning of the generalization scheme (number of bins, etc) if a different bicycle model were used.

Lagoudakis, et al. used a linear function approximation with 20 basis functions that were nonlinear functions of the state (e.g., $\dot{\theta}^2$, $\omega^2\theta$, etc.) [4]. Their method, Least Squares Policy Iteration (LSPI), does not require the state to be discretized, but still requires a discrete action space. Lagoudakis and Parr have also used SVMs for the same bicycle in order to store $Q(s, a)$ [3]. Ernst, et al. use tree-based supervised learning methods [1]. Ng and Jordan learn a Markov Decision Process, which allows for a continuous action space (contrary to Randlov and Alstrom's work) [6]. Of course, many of the generalization methods (or reinforcement learning algorithms that come with a form of generalization) have not been applied to the bicycle problem. For example, neural fitted Q iteration (NFQ) is designed to be less sensitive to any specific experience, because it is offline and $Q(s, a)$ is stored using a neural network with a hidden layer [8]. We thoroughly explored both the LSPI and NFQ methods, but had limited success. In this report, we only discuss results from using Randlov's original method.

1.2 Controlling a bicycle

Why does it make sense to use reinforcement learning to control a bicycle? We see three reasons. As is evident in [7], reinforcement learning allows the learning of a control law that causes qualitatively different behavior depending on the state: the agent can learn to focus on riding towards a goal when the bicycle is upright, but must focus on balancing when the bicycle starts to tilt significantly. Secondly, reinforcement learning allows for a control law that achieves a much more high-level goal than that obtainable via typical optimal control methods. While others have studied lane-change maneuvers with conventional control methods [11], the task of driving to a goal that is at a right angle from the bicycle's heading would be a challenge for such methods unless the entire trajectory is specified That is, reinforcement learning can be used for trajectory optimization.

The last reason relates the bicycle problem to other problems typically approached with machine learning algorithms. The bicycle is not so different from the cart pole: in both cases we attempt to keep an unstable system upright. However, the control in the cart pole problem more directly affects the angle of the pole: move the cart so that it is under the pole. In the bicycle problem, we have the same rough objective: move the bike in the direction it is falling in order to ”catch” it. However, the control to move the bike under itself is not so simple. Indeed, to turn a bicycle to the left, one must first steer to the right. This is called counter-steer. Indeed, this is reminiscent of the mountain car problem, in which we have the tricky result that the car must first go in the direction opposite to its goal destination.

2 Randlov's Model

Using Randlov and Alstrom's bicycle model and environment, we trained a learner using the SARSA(λ) algorithm. We implemented Randlov's balance and go-to tasks with PyBrain, a machine learning library for Python [9]. The episodic learning framework is illustrated in Figure 1. To evaluate the execution of the task during learning, we will additionally define a *rehearsal* sequence. In one rehearsal sequence, the agent learns by executing 10 episodes, after which, the agent "performs" one episode greedily without learning. We use the discounted sum of rewards achieved over the performance episode as a success metric.

We used Randlov's reward function for both tasks. For the balance task, a reward of -1 was given when the absolute tilt angle exceeded 30° and 0 otherwise. For the go-to task, the following reward was given.

if $abs(\omega) > \frac{\pi}{12}$

$$r = -1$$

else

$$r = (4 - \psi^2) * 0.0004$$

Figure 2 shows the results from Randlov's balance task. Our learner was able to balance indefinitely after approximately 1500 trials. We observed cases in which the agent would fall travel in stable, circular trajectories. This was also observed by Randlov. Figure 3 shows the resulting implementation of Randlov's go-to task. The agent lfirst learns to balance, and it first reaches the goal after approximately 4500 trials. Both of the tasks are very sensitive to learning rate annealing. Randlov does not mention this, despite the fact that seems to be well-known in the field [10]. Without annealing, neither task will result in a converging policy because the current learning continually discards all the learning that had already been done. The go-to task required a much slower annealing rate than did the balance task because the agent first needs to gain experience balancing before attempting to navigate toward the goal.

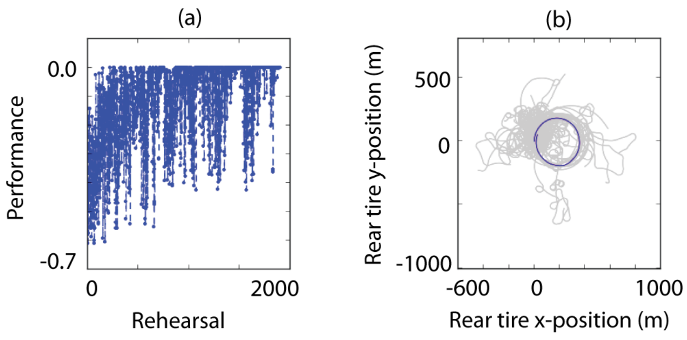

Figure 2: (a) Learning curve for the balance task. (b) The trajectories of the bicycle's wheel contact points in each of several thousand episodes. Simulation was stopped after 1000 seconds, so trajectories may appear to terminate. The agent also found several stable circular trajectories, as the one highlighted above.

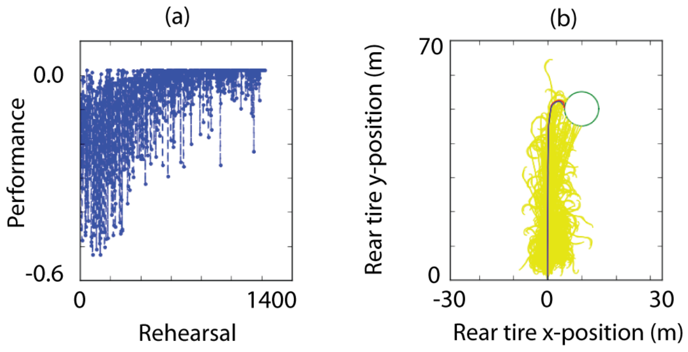

Figure 3: (a) Learning curve for the "go-to" task. (b) The trajectories of the bicycle's wheel contact points after several thousand episodes. Here the goal was placed at a position (x = 10m, y = 50 m) relative to the bicycle starting location. After about 4000 episodes (400 rehearsals), the bicycle reaches the goal with increasing frequency.

Now, building on Randlov's implementation, we explore our topics of interest–human ability to control a virtual bicycle, and shaping of the reinforcement reward function.

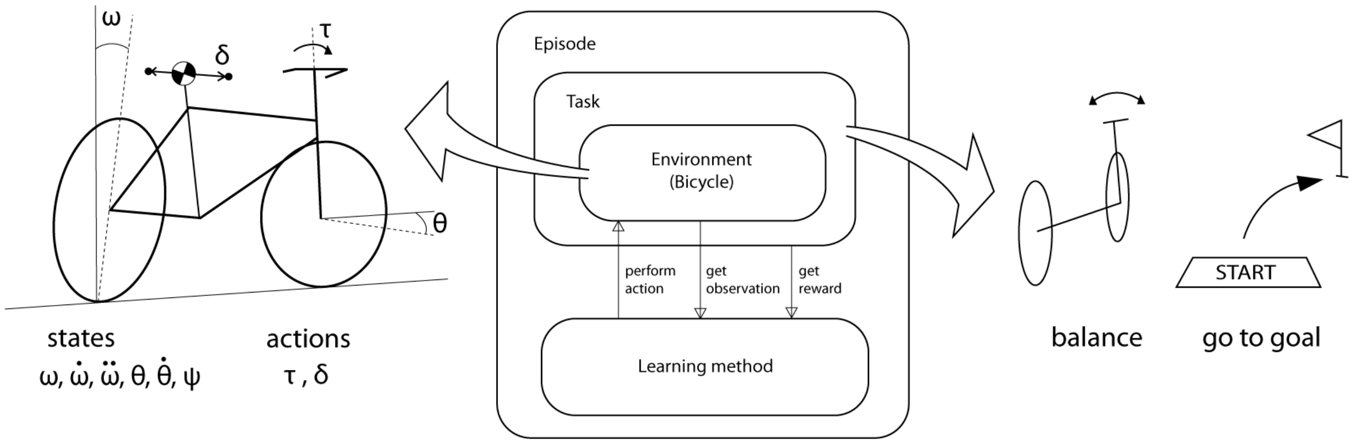

Figure 1: Learning methodology, in PyBrain's terminology.

3 Game

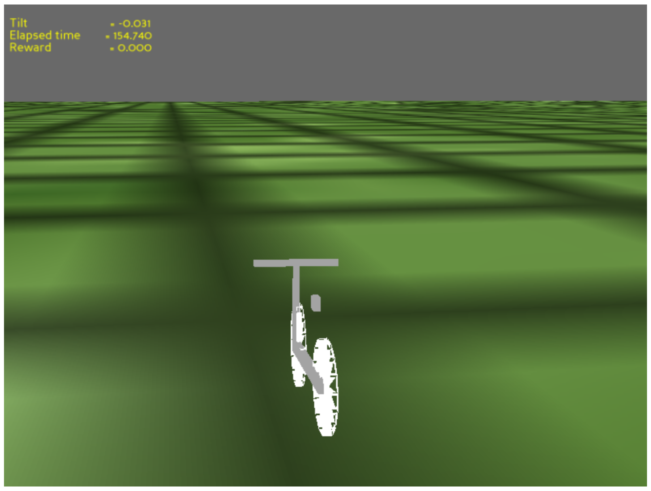

Figure 4: We simulated our model using a Python game engine, Panda3d [2]. Here, the user's may attempt to balance the bicycle by applying a torque to the handlebars or by shifting the center of mass via keyboard inputs. Additionally, the game engine was used to visualize the agent's learning process. This aspect was useful for demonstration and debugging.

To explore how humans control a bicycle, we created an interactive simulation environment using Panda3d, a game engine for Python. We simulated Randlov's bicycle model, and provided users control over the same actions available to the reinforcement learning agent. In testing the game, we quickly discovered the reflexive nature of human control of a bicycle. Successful coordination of body mass displacement and torque control was difficult to accomplish. In fact, we struggled mightily to balance the bike for more than a few seconds, far worse than what our trained agent was able to achieve. This indicates that the use of apprenticeship learning for this problem would require that the observations come from humans riding real bicycles; not from keyboard inputs.

During the poster session for this project, we considered allowing observers to try their own hand at balancing the bicycle. However, since we found it to be so challenging, we decided instead to allow observers to apply a perturbation to the bicycle's steering torque (using keyboard input). The learned controller was able able to reject disturbances that were five times greater than the steering torque it has available to itself for control!

In future studies, we would like to collect user input data for a simple balance task and compare this to an optimal reinforcement learning policy. Our preliminary experiments suggest that the simulation may need to be sufficiently slowed for usability. Further, we found great utility in visualizing the learning process using our game model. It helped illustrate behaviors that were being rewarded, and it was a valuable debugging tool. Games such as this can be used as educational or illustrative tools.

4 Shaping

One of the key factors in any reinforcement learning application is the reward function that is being maximized. To optimize the learning, the reward function provide positive reward for actions that help attain the goal and negative reward for actions that hinder the bike from attaining its goal. This process of reward engineering is called shaping. Here we describe a systematic shaping of reward functions for a go-to task.

4.1 Balance Task

In order to shape an appropriate reward function for balance, we first determine what behavior in the bike model can help avoid a fall. If the bicycle's tilt ω satisfies $|\omega| > \frac{\pi}{6}$, then we want to reward actions that reduce tilt and punish actions which increase tilt. For tilt ω at timestep t we define ω' as the tilt at time $t+1$ and a balance reward function as follows:

$$R_b(\omega, \omega') = \begin{cases} A(|\omega| - |\omega'|), |\omega| > |\omega'| \\ B(|\omega| - |\omega'|), |\omega| < |\omega'| \end{cases} \tag{1}$$

where A is defined as the reward factor and B is defined as the punishment factor.

4.2 Go-to Task

In the case of a go-to-destination task, the agent must both balance the bicycle and navigate to the goal. We measure navigation performance through ψ_g, the error in the heading at time t. As in balance, we define ψ_g', the heading error at time $t+1$ and a navigation reward function as follows:

$$R_n(\psi_g, \psi_g') = \begin{cases} A(|\psi_g| - \left|\psi_g'\right|), |\psi_g| > \left|\psi_g'\right| \\ B(|\psi_g| - \left|\psi_g'\right|), |\psi_g| < \left|\psi_g'\right| \end{cases} \tag{2}$$

Finally, we combine R_b and R_n for a composite reward function $R_t = R_b + cR_n$, where c determines the degree of emphasis on balance versus navigation, and A and B have the same values as in R_b. We suppose that there exists an optimal ratio between balance and navigation that is characterized by c^*. Now, we expect c^* to scale inversely with the expected converged values of R_n, implying the following behavior for c^*:

$$c^* \propto \frac{1}{Cum(R_n)} \sim \frac{1}{\arctan\left(\frac{y}{x}\right)} \tag{3}$$

where the target location is (x, y).

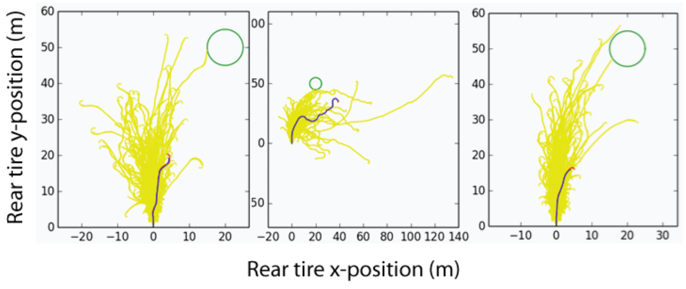

Figure 5: Path trajectories to a target centered at $(20, 50)$ after 5500 episodes for $c = 0.05$ (left), $c = 0.1$ (center), and $c = 0.2$ (right). Both $c = 0.05$ and $c = 0.2$ reached the goal once but $c = 0.1$ reached the goal twice. Thus, we expect $c*$ to be near $c = 0.1$.

To test this hypothesis we set $A = 5000$, $B = 3000$ and tried several c values for a target at $(20, 50)$ and concluded that $c^*_{(20,50)}$ is in the vicinity of $c = 0.1$ (Figure 5). Next, we kept the same A, B, and c values for a target at $(40, 60)$ and we found that $c = 0.05$ is the best estimate of $c^*_{(40,50)}$. This confirms the hypothesized behavior which predicts that $c^*_{(20,50)} > c^*_{(40,50)}$.

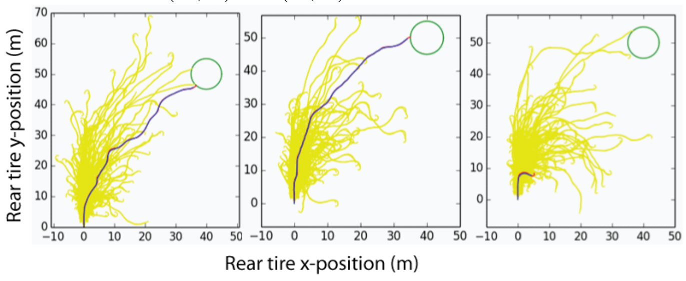

Figure 6: Path trajectories to a target centered at $(40, 50)$ after 5500 episodes for $c = 0.05$ (left), $c = 0.1$ (center), and $c = 0.2$ (right). Both $c = 0.1$ and $c = 0.2$ reached the goal once. $c = 0.1$ reached the goal three times and thus a better estimate of $c*$.

4.3 Future work

We have two major future goals in the area of shaping. First, we plan to more thoroughly characterize the behavior of c^* by repeating the trials presented here for statistical confidence and testing more perturbations. In particular, we plan to test a series of targets at with a constant $\frac{y}{x}$ and varying distance. Our second goal is to develop a more sophisticated reward function. The current reward system has produced functional controllers for various targets. However, it linearly combined R_n and R_b which implicitly treats them as orthogonal. But we know that this is not true since ω and ψ_g are coupled in our model. Thus, we plan to explore reward functions which include cross terms between ω and ψ_g. By developing more sophisticated reward systems and characterizing the behavior of the coefficients in the reward function we hope to develop a systematic method of deriving reward functions. This will eliminate most of the fine-tuning that is usually required in controller design.

Randlov and Alstrom's bicycle model is highly simplified, and does not capture all the interesting dynamics of bicycle motion. For instance, their steer axis is vertical. Realistic bicycles have a slanted steer axis that is designed so that the point where the front wheel contacts the ground in fact lies *behind* the point where the steer axis intersects the ground. It had long been supposed that this feature contributes to the stability of bicycles. In fact, bicycles are self-stable (they won't fall over if pushed sideways) between about 4 and 6 m/s (forward speed).

Thus, it seems the utility of their problem is as a toy problem for the comparison of reinforcement learning methods, with limited applicability for actually controlling a bicycle. Significant work has been performed to develop accurate models of bicycle dynamics [5].

References

[1] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-Based Batch Mode Reinforcement Learning. 6:503–556, 2005.

[2] Mike Goslin and Mark R. Mine. The Panda3D graphics engine. *Computer*, 37(10):112–114, October 2004.

[3] Michail G Lagoudakis, Parr Cs, and Duke Edu. Reinforcement Learning as Classification : Leveraging Modern Classifiers. 2003.

[4] Michail G Lagoudakis, Ronald Parr, and Michael L Littman. Least-Squares Methods in Reinforcement Learning for Control. pages 249–260, 2002.

[5] Jason Keith Moore. *Human Control of a Bicycle*. PhD thesis, University of California, Davis, 2012.

[6] Andrew Y Ng and Michael Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *Uncertainty in Artificial Intelligence*, 2000.

[7] Jette Randlov and Preben Alstrom. Learning to drive a bicycle using reinforcement learning and shaping. *Proceedings of the Fifteenth International Conference on Machine Learning*, 1998.

[8] Martin Riedmiller. Neural Fitted Q Iteration - First Experiences with a Data Efficient Neural Reinforcement Learning Method. In *16th European Conference on Machine Learning*, 2005.

[9] Tom Schaul, Justin Bayer, Daan Wierstra, Yi Sun, Martin Felder, Frank Sehnke, Thomas Rückstieß, and Jürgen Schmidhuber. PyBrain. *Journal of Machine Learning Research*, 2010.

[10] Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. 2012.

[11] Robin S. Sharp. On the Stability and Control of the Bicycle. *Applied Mechanics Reviews*, 61(6):060803, 2008.

[12] Richard S Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. The MIT Press, Cambridge, MA, 1998.