University of British Columbia
Department of Computer Science

Computer Animation

Project Report

Project Title:

Learning to Ride a Bicycle with Reinforcement Learning

Group Members:

Setareh Cohan - 98701162
Saeid Naderiparizi - 98712169

December 16, 2017

CONTENTS

1 Introduction

1.1 Problem

Starting to choose a project for this course, we decided to use reinforcement learning for riding a bicycle. We thought of breaking this problem into two subproblems in order to make it easier to tackle:

1. **Balance the bicycle**: This means that we will only look at the lean angel of the bicycle and try to keep it balanced by steering the handlebars, while it moves with a constant speed.
2. **Ride to a specific destination**: In this phase, we will determine the rotation angle that the bicycle should have in order to get to a desired destination.

We started from the first subproblem and were able to solve and implement only this subproblem and since we did not have enough time, we could not work on the second subproblem.

1.2 Why Reinforcement Learning?

In supervised learning, we have seen algorithms that tried to make their outputs mimic the labels y given in the training set. In that setting, the labels gave an unambiguous "right answer" for each of the inputs x. In contrast, for many sequential decision making and control problems, it is very difficult to provide this type of explicit supervision to a learning algorithm. For our problem here, initially we have no idea what the "correct" actions to take are to make it balance and ride to a destination, and so do not know how to provide explicit supervision for a learning algorithm to try to mimic.
For these types of settings, we need a reinforcement learning framework, which will instead provide our algorithms only a reward function, which indicates to the learning agent when it is doing well, and when it is doing poorly. One reward that we could think of for the task of keeping the bike balanced, was to give high rewards when the lean angle was small, and give low rewards when the lean angle was large.

We had many choices for the reinforcement learning method to use. Three of the main ones were *Model predictive control*, *Fitted value iteration* and *Policy search*. We chose to use *Fitted value iteration* algorithm and the details of it is discussed in the next subsection.

1.2.1 Fitted Value Iteration

In this method, we basically define a set of states S, a set of actions A, a reward function R for each state, and a value function V for each state. The algorithm for this method is as follows:

ALGORITHM 1: Fitted Value Iteration

Input: Set of states S, set of actions A, reward function R and value function V
Output: V and best actions to choose at each step.
For each state $s \in S$, initialize $V(s) := 0$;
repeat
  for *each state* $s \in S$ **do**
    update $V(s) := R(s) + \max_a \{\gamma \hat{V}(s')\}$
    update best action for state s to $\arg\max_a \{\gamma \hat{V}(s')\}$;
  end
until V *converges*;

This algorithm can be thought of as repeatedly trying to update the estimated value function using Bellman updates that we discussed in the class.
There are two possible ways of performing the updates in the inner loop of the algorithm. In the first, we can first compute the new values for $V(s)$ for every state s, and then overwrite all the old values with the new values. This is called a synchronous update. In this case, the algorithm can be viewed as implementing a *Bellman backup operator* that takes a current estimate of the value function, and maps it to a new estimate. Alternatively, we can also perform asynchronous updates which means updating $V(s)$ at each step and this is what we do in our implementation ([1], [2]).

2 IMPLEMENTATION

2.1 MODEL

In order to balance the bicycle as our first goal and then ride the bicycle to a destination as a final goal, we modeled the bicycle as an inverted pendulum as in [3]. We decided to use this model because it was easier to understand than other models we have seen([4], [5]).

In this setting, the inverted pendulum represents the bicycle such that the tilt angle of the pendulum corresponds to the lean angle of the bicycle. Figure below shows the essential components of the model which are explained further. The bicycle is moving along the z-axis with constant velocity v, and by changing the steering angle we move its direction in the $x-z$ plane. It leans about the z-axis and we should control this leaning angle by steering, so that lean angle would not grow larger than a certain threshold to keep the bike balanced.

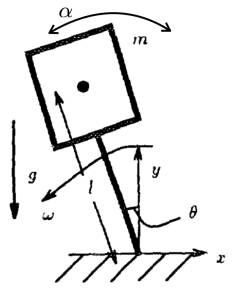

Figure 2.1: Inverted pendulum used to model the bicycle. [3]

- m: Combined mass of the bicycle and the rider.
- l: Height of the center of mass.
- I: Moment of inertia about the z-axis.
- θ: Lean angle.
- ω: Angular velocity of lean angle.
- g: Acceleration due to gravity.
- α: Steering angle.
- v: Constant velocity of bike along z-axis.

2.2 Setting of the Learning Algorithm

As explained in section 1, we use the *Fitted Value Iteration* algorithm for the reinforcement learning part of our project. Here, we will explain the details of the algorithm for each subtask of the problem separately in the following two subsections.

2.2.1 Balancing the Bicycle

For this task, components of the algorithm are as below:

- **States**
  We define each state S as $s := (\theta, \dot{\theta})$ which correspond to the leaning angle and the angular velocity about the z-axis. We constrained and discretized these values to be 17 equally spaced values in range of $[-12, +12]$ for θ and 17 equally spaced values in range of $[-3, +3]$ for $\dot{\theta}$.
- **Actions**
  Actions are defined to be the steering angles which are also constrained and discretized. We have defined them to be $a \in -6, -3, 0, +3, +6$ degrees.
- **Reward**
  We defined the reward for state $s = \{\theta, \dot{\theta}\}$ to be $R(s) = c^2 - \theta^2 - (0.1)\dot{\theta}^2$. c is a constant set to 0.5 in our code.

- **Gamma**
  The constant γ is set to be 0.9 in our code.

Now, we know all that we need to implement the *Fitted Value Iteration* algorithm just as the specifications of 1, except for how to get s' which is the next state after applying best action on the current state s and the $\hat{V}$ which is an estimate of the value function V for s'. We will obtain these as below:

- **Updating s to s'**
  In order to find the next state s' from the current state s and the action to perform a, we need a simulator that maps s, a to s': $a' \leftarrow simulator(s, a)$. What this simulator does, is mapping updating θ and $\dot{\theta}$ based on the steering angle α which is our best action. This is done as follows:

  - **Step 1**
    Define curvature C_f of the bike on the $x-z$ plane as:

    $$C_f := \frac{1}{r}$$

    where r is the instantaneous radius of curvature.[3]
    It is easy to show that r can be calculated from α and d as:

    $$r = \frac{d}{tan(\alpha)}$$

    where d is the distance between the front and rear wheel of the bike. We have set this value to $1m$ in our implementation.

  - **Step 2**
    Define a dynamic function that maps the radius of curvature r to angular acceleration of the lean angle $\ddot{\theta}$. In order to do this, we used equation 20 of [3] which is:

    $$\frac{I\ddot{\theta}}{ml} = -\frac{v^2}{r}cos(\theta) + lsin(\theta)cos(\theta)\frac{v^2}{r^2} - l\ddot{\theta} + gsin(\theta)$$

    Using this, we defined the dynamic function $D : r \rightarrow \ddot{\theta}$ as below:

    $$D(r) = \ddot{\theta} = \frac{-\frac{v^2}{r}cos(\theta) + lsing(\theta)cos(\theta)\frac{v^2}{r^2} + gsin(\theta)}{\frac{I}{ml} + l}$$

  - **Step 3**
    Update state $(\theta, \dot{\theta})$ using $\ddot{\theta}$ as below:

    $$\begin{cases} \theta \leftarrow \theta + \dot{\theta}\Delta T \\ \dot{\theta} \leftarrow \dot{\theta} + \ddot{\theta}\Delta T \end{cases}$$

    We defined the time step ΔT to be $0.4s$ throughout our implementation.

- **Defining $\hat{V}$**
  To estimate the value function for the new state s' ($\hat{V}(s')$), we use interpolation. The interpolation method we use is a very simple one in which we find the value function V of the nearest state s^* to s' and report it as $\hat{V}(s')$.

After we run the reinforcement learning algorithm for a specific state s, we end up with the best action to perform at this state. Now, we update the position and steering angle of the bike as below:

$$\begin{cases} P \leftarrow P + \begin{bmatrix} cos(\alpha) \\ sin(\alpha) \end{bmatrix} v\Delta T \\ \alpha \leftarrow \alpha + \omega\Delta T \end{cases}$$

where ω or the angular velocity of the steering angle is calculated as

$$\omega = \frac{v}{r} = \frac{v}{\frac{d}{tan(\alpha)}} = \frac{v\tan(\alpha)}{d}$$

2.3 Constants Used in the Code

Here is a list of all constant values used in the code:

- m: Combined mass of the bicycle and the rider.
  - We have set this value to $60kg$ in our implementation.
- l: Height of the center of mass.
  - We have set this value to $0.3m$ in our implementation.
- I: Moment of inertia about the z-axis.
  - We tried different values for moment of inertia and the value that worked best was to model the bike as a cylinder with moment of inertia equal to $\frac{2}{3}md^2 \times 10kg.m^2$.
- d: Distance between the front and rear wheel of the bike.
  - We set this value to $1m$ in our code.
- g: Acceleration due to gravity.
  - We have set this value to $9.8m/s^2$ in our implementation.
- α: Steering angle.
- v: Constant velocity of bike along z-axis.
  - We have set this value to $5m/s$ in our implementation.
- ΔT: step size that we use is $0.4s$.

2.4 Challenges

We had more challenges than we anticipated for this project. Some of the most important ones are listed below:

- The biggest challenge for us was first to understand the physical aspects of the problem and formulate the relations between the steering and lean angle of the bicycle.
  For instance, setting the constants was not trivial in some cases such as the moment of inertia. At the beginning, we set this value to 1 and then we updated it to $\frac{2}{3}md^2$ and it improved our results. However, our bike fell a lot quicker than we expected it to. Therefore, we increased the moment of inertia to $\frac{2}{3}md^2 \times 10$

- Another challenge for us was to understand reinforcement learning and study about different reinforcement learning methods available and choose one that is not too complicated and does not take a long time to train and use, and also gives good results for our project.
- We were new to OpenGL and it was challenging for us to use OpenGL and learn how to work with it.

3 Results

3.1 Code

Our code is available here: `https://github.com/saeidnp/LearningToRideABike.git`.

4 Conclusion and Future Work

Our results are actually acceptable. When the staring lean angle is big, the bike cannot keep itself balanced and it falls. However, the bike can keep it self balanced with small starting lean angles. We can definitely improve our results by:

- Use more precise and physically coherent constants such as moment of inertia.
- Use a better reward function.
- Improve the interpolation by more sophisticated methods.
- Try other reinforcement learning algorithms and see if they give better results.
- Improve the graphics and visualization of our project.
- Adding a 3rd dimension to our state model meaning using $(\theta, \dot{\theta}, \alpha)$ instead of $(\theta, \dot{\theta})$. α is the steering angle.

We should also add the second subproblem to the project which is to ride the bike to a goal. This task is not as straightforward as it may seem since we have to solve two problems at the same time: Balancing on the bicycle and driving to a specific place.

REFERENCES

[1] C. Szepesvári, "Algorithms for reinforcement learning," 2009.

[2] A. Ng, "Reinforcement learning and control,"

[3] M. Vandepanne, E. Fiume, and Z. Vranesic, "Physically based modeling and control of turning," *CVGIP: Graphical Models and Image Processing*, vol. 55, no. 6, pp. 507–521, 1993.

[4] J. Randlov and P. Alstrom, "Learning to drive a bicycle using reinforcement learning and shaping," in *Proceedings of the Fifteenth International Conference on Machine Learning*, pp. 463–471, 1998.

[5] M. Cook, "It takes two neurons to ride a bicycle," *Demonstration at NIPS*, vol. 4, 2004.